

Fuzzy Clustering and Data Analysis in Python

The `fuzzyclust` toolbox

János Abonyi

University of Pannonia, Veszprém



Fuzzy Clustering and Data Analysis in Python

The `fuzzyclust` toolbox

János Abonyi

University of Pannonia, Veszprém

Version 1.0 · November 2026

Fuzzy Clustering and Data Analysis in Python. The `fuzzyclust` toolbox

János Abonyi, University of Pannonia, Veszprém, Hungary

Version 1.0, November 2026

© 2026 János Abonyi. This book is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0, https://creativecommons.org/licenses/by/4.0/): it may be copied, redistributed, and adapted for any purpose, provided the source is credited. The `fuzzyclust` software described in the book is released under the MIT License.

Software: https://github.com/ComplexSystemsIntelligence/fuzzyclust
Documentation: https://complexsystemsintelligence.github.io/fuzzyclust
Installation: `pip install fuzzyclust`

The research summarized in this book was supported by the National Research, Development and Innovation Office of Hungary (NKFIH) through the OTKA 143482 project.

Typeset with LaTeX in Libertinus Serif, Raleway, and IBM Plex Mono. Every figure that shows a computed result was produced by an example script of the package. The cover shows a Gustafson–Kessel partition computed with `fuzzyclust`: each sample is drawn in the blend of its three membership colors, and the lines are contours of the memberships.

Preface

This book is about clustering with fuzzy partitions and about what can be built on it: validity indices that estimate how many groups a data set contains, projections that make a high-dimensional partition visible, prototypes that are lines, shells, or local models, graphs that replace the prototype altogether, and clustering-based identification of dynamic systems and segmentation of time series. The last chapters carry the same machinery over to relational, categorical, incomplete, very large, and image data, and to partial supervision, co-clustering, collaborative clustering, and cluster ensembles. All of it is implemented in `fuzzyclust`, an open-source Python package with a scikit-learn interface, and every figure in the book that shows a computed result is produced by one of its example scripts.

Who the book is for. It is written for engineers, data scientists, and graduate students who know k-means and want to understand what fuzzy clustering adds, why the Gustafson–Kessel and Gath–Geva algorithms detect cluster shapes that fuzzy c-means cannot, and how a cluster becomes a rule of a Takagi–Sugeno model or a segment of a process record. The mathematics requires no more than linear algebra and elementary calculus; every algorithm is stated in a box, summarized in a few bullets, and demonstrated on data whose structure is known. Readers who only want to use the toolbox can skip the theory sections and read the *Using it in fuzzyclust* and *Examples* sections of each chapter; readers interested only in the methods can skip the code.

The toolbox and its predecessor. `fuzzyclust` is the Python successor of the *Fuzzy Clustering and Data Analysis Toolbox for MATLAB*, written in 2005 by Balázs Balaskó, Balázs Feil, and the author. The MATLAB toolbox accompanied the book *Cluster Analysis for Data Mining and System Identification* and served the author's research group at the University of Pannonia as the engine of a long line of work on clustering-based modeling, visualization, and time-series analysis. Its core—the partitioning algorithms, the validity indices, and the projection methods—maps function by function onto the Python package (appendix B). The research threads that grew around it are collected in the later chapters and in the sub-packages `models`, `timeseries`, `viz`, and `graph`; further sub-packages add methods from the wider literature.

How to run the examples. The package is installed with `pip install fuzzyclust`. The scripts that produce the figures are in the `examples/` directory of the repository, grouped into folders by topic, and are listed in appendix C. The sphinx-gallery extension renders them as the online gallery of the documentation and converts them into Jupyter notebooks that run in the browser on Binder or Google Colab, so every experiment in the book can be rerun and modified without a local installation. Apart from a few short listings, the code snippets printed in the chapters are extracted from the same scripts and therefore run with the version of the toolbox that the book describes.

Acknowledgments. Much of the design of the MATLAB toolbox, written with Balázs Balaskó and Balázs Feil, survives in the Python package. Many methods of the later chapters were developed with co-authors to whom the author is deeply indebted: Robert Babuška, Ferenc Szeifert, Sándor Németh, Péter Árva, Annamária Kovács, Sándor Migály, Daniela Girimonte, Ágnes Vathy-Fogarassy, and Tamás Kenesei; their papers are cited in the *Further reading* sections of the chapters. The author acknowledges the support of the OTKA 143482 project of the National Research, Development and Innovation Office (NKFIH), Hungary.

Veszprém, November 2026 — János Abonyi

Contents

Preface iii

1 Introduction 1

1.1 What clustering is for 1
1.2 Hard and fuzzy partitions in one picture 2
1.3 From the MATLAB toolbox to fuzzyclust 3
1.4 How the book is organized 5
1.5 How to read this book 6
1.6 Installation and a first session 7
1.7 Further reading 8

2 Foundations 9

2.1 Problem and idea 9
2.2 Theory 10
2.3 Using it in fuzzyclust 20
2.4 Examples 22
2.5 Further reading 24

3 Partitioning algorithms 27

3.1 Problem and idea 27
3.2 Theory 28
3.3 Using it in fuzzyclust 44
3.4 Examples 50
3.5 Further reading 61

4 Cluster validity 63

4.1 Problem and idea 63
4.2 Theory 64
4.3 Using it in fuzzyclust 77
4.4 Examples 80

4.5 Further reading . . . 85

5 Visualization 87

5.1 Problem and idea . . . 87
5.2 Theory . . . 88
5.3 Using it in fuzzyclust . . . 95
5.4 Examples . . . 99
5.5 Further reading . . . 105

6 Robust and shape-oriented variants 107

6.1 Problem and idea . . . 107
6.2 Theory . . . 110
6.3 Using it in fuzzyclust . . . 124
6.4 Examples . . . 130
6.5 Further reading . . . 143

7 Graph-based fuzzy clustering 145

7.1 Problem and idea . . . 145
7.2 Theory . . . 147
7.3 Using it in fuzzyclust . . . 156
7.4 Examples . . . 159
7.5 Further reading . . . 167

8 Clustering for model identification 169

8.1 Problem and idea . . . 169
8.2 Theory . . . 171
8.3 Using it in fuzzyclust . . . 182
8.4 Examples . . . 188
8.5 Further reading . . . 199

9 Clustering of time series 203

9.1 Problem and idea . . . 203
9.2 Theory . . . 204
9.3 Using it in fuzzyclust . . . 218
9.4 Examples . . . 221
9.5 Further reading . . . 230

10 Relational and mixed data 233

10.1 Problem and idea . . . 233
10.2 Theory . . . 236
10.3 Using it in fuzzyclust . . . 246

10.4 Examples . . . 249
10.5 Further reading . . . 259

11 Incomplete, very large, and image data 261
11.1 Problem and idea . . . 261
11.2 Theory . . . 264
11.3 Using it in fuzzyclust . . . 274
11.4 Examples . . . 277
11.5 Further reading . . . 284

12 Beyond a single unsupervised run 287
12.1 Problem and idea . . . 287
12.2 Theory . . . 291
12.3 Using it in fuzzyclust . . . 301
12.4 Examples . . . 304
12.5 Further reading . . . 313

A API reference 315
A.1 `fuzzyclust.base` . . . 315
A.2 `fuzzyclust.cluster` . . . 315
A.3 `fuzzyclust.validity` . . . 316
A.4 `fuzzyclust.extensions` . . . 317
A.5 `fuzzyclust.graph` . . . 318
A.6 `fuzzyclust.viz` . . . 319
A.7 `fuzzyclust.models` . . . 320
A.8 `fuzzyclust.timeseries` . . . 321
A.9 `fuzzyclust.relational` . . . 323
A.10 `fuzzyclust.incomplete` . . . 324
A.11 `fuzzyclust.large` . . . 324
A.12 `fuzzyclust.image` . . . 325
A.13 `fuzzyclust.semisupervised` . . . 325
A.14 `fuzzyclust.coclustering` . . . 326
A.15 `fuzzyclust.collaborative` . . . 326
A.16 `fuzzyclust.ensemble` . . . 327
A.17 `fuzzyclust.datasets` . . . 327
A.18 `fuzzyclust.preprocessing` . . . 328
A.19 `fuzzyclust.plotting` . . . 328

B Migrating from the MATLAB toolbox 329
B.1 Functions . . . 329

B.2 Parameters . . . 329
B.3 Results . . . 329
B.4 Behavioral differences . . . 330

C The example gallery 335

C.1 Core partitioning algorithms . . . 335
C.2 Visualization of fuzzy clusters . . . 336
C.3 Model identification . . . 336
C.4 Time-series analysis . . . 337
C.5 Graph-based clustering . . . 337
C.6 Possibilistic . . . 338
C.7 Relational, categorical, and mixed data . . . 338
C.8 Incomplete, very large, and image data . . . 339
C.9 Partial supervision . . . 339

Bibliography 341

List of Figures

1.1 Hard and fuzzy partitions of two overlapping groups. 3
1.2 Lineage of the toolbox. 5
1.3 The first session: fuzzy c-means on three Gaussian clusters. 8

2.1 The prototypes of this book and the cluster shapes they detect. 10
2.2 Unit spheres $\{\mathbf{x} : \mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} = 1\}$ of the inner-product norms. 12
2.3 One partition matrix of three clusters and five samples in each of the three partitioning spaces. 14
2.4 Alternating optimization as implemented once in `BaseFuzzyClusterer`. 15
2.5 Fuzzy c-means with four norms on three clusters of identical elongated shape. 23
2.6 Gustafson–Kessel clustering of the motorcycle data without scaling (left), after range scaling (middle), and after variance scaling (right), with the prototypes shown in the original units. 23
2.7 Final objective of Gustafson–Kessel on three line segments for twenty outer seeds and four initialization strategies. 24
2.8 Xie–Beni index of FCM on held-out folds of the standardized iris data for several c and m, computed inside `GridSearchCV` with `predict_proba`. 25

3.1 Level curves of the maximal membership for one set of prototypes under four algorithms. 28
3.2 Membership functions of two clusters with prototypes $v_1 = 0$ and $v_2 = 1$ under four fuzzification mechanisms. 41
3.3 Subtractive clustering in one dimension. 45
3.4 Fuzzy c-means on three separated clusters for $m = 1.2$, 2, and 4. 51
3.5 Membership in the cluster of $v_1 = 0$ along the line to $v_2 = 1$ for five exponents. 51
3.6 FCM on three overlapping clusters for $m = 1.2$, 2, and 4: as m grows, the partition becomes less sharp and the prototypes move slightly toward the grand mean. 52
3.7 The information a fuzzy partition carries beyond the labels. 52
3.8 Three elongated clusters. 53

3.9 The volume parameter of GK. 53
3.10 Covariance regularization on a perfectly collinear cluster. 54
3.11 Twenty random starts of GK on three line segments. 54
3.12 Clusters of very different size and density. 55
3.13 The GG result read as a Gaussian mixture: the density $\sum_i \alpha_i \mathcal{N}(\mathbf{x}; \mathbf{v}_i, \mathbf{F}_i)$ of the three fitted components over the data. 55
3.14 The membership function learned from the same training data by four algorithms. 56
3.15 Gustafson–Kessel contours on the standardized motorcycle data. 57
3.16 Memberships of three clusters with prototypes at 0, 1, and 4 on a line under four fuzzification mechanisms. 58
3.17 Maximal membership on four groups for FCM, entropy and quadratic regularization, and the polynomial fuzzifier: the sparse mechanisms are crisp inside the groups and fuzzy only between neighbors. 58
3.18 Deterministic annealing on four groups. 59
3.19 Parameter study: partition coefficient of the entropy regularization against λ (left; the drop to $1/2$ is the merging of the prototype pairs above the critical temperature) and partition coefficient and fraction of exact zeros of the polynomial fuzzifier against β (right). 59
3.20 Subtractive clustering and the mountain method on eight elongated groups. 60
3.21 Number of centers against the radius for three squash factors and for the mountain method. 60
3.22 Test error of Takagi–Sugeno models of $\sin(x)/x$ against the number of rules, with the rule numbers estimated by subtractive clustering in the input–output space for five radii (dashed lines). 61

4.1 Two routes to the number of clusters. 65
4.2 The two ingredients of the geometric validity indices. 65
4.3 Validity scan of FCM on five Gaussian clusters, indices min–max scaled to $[0, 1]$. 81
4.4 Compatible cluster merging with the similarity criterion on five groups. 82
4.5 Validity scan (left) and stability selection (right) on the same five groups. 82
4.6 Fuzzy Rand, fuzzy adjusted Rand, and Campello's fuzzy adjusted Rand index between six partitions of the same data. 83
4.7 Compatible cluster merging with the geometric criterion on three line segments: GK with ten clusters (left) and the three clusters left after merging pieces with parallel normals on a common line (right). 83
4.8 Parameter study: the final number of clusters of the similarity merging against its threshold (left) and the mean stability profile for three sub-sample fractions (right). 84
4.9 Gustafson–Kessel fitted on 60 % of the data and evaluated on the rest: the test samples (open markers, colored by predicted label) fall into the membership regions learned from the training data. 85

5.1 The idea of the fuzzy Sammon mapping. 89
5.2 The regularization of the fuzzy self-organizing map. 96
5.3 PCA keeps the directions of largest spread and lets clusters that differ in the discarded directions overlap; the Sammon mapping started from the same configuration bends the picture until the inter-point distances match, which separates the five groups and halves the stress. 99
5.4 Convergence of Sammon's stress. 100
5.5 Three elongated clusters in four dimensions. 100
5.6 The FCM partition of the wine data projected by PCA, Sammon mapping, and FUZZSAM; colors are the cultivars, contours the recomputed membership levels, circles the wines whose cluster disagrees with their cultivar. 101
5.7 FUZZSAM pictures of the GK clustering of the iris data with $m = 2$ (left) and $m = 1.4$ (right), drawn with the same membership levels. 101
5.8 A chain of seven prototypes on a noisy helix. 102
5.9 A 6×6 fuzzy SOM of points on a hemisphere, lattice drawn in the (x_1, x_2) plane. 103
5.10 Component planes of the ordered map ($\vartheta = 0.5$): the first two coordinates change monotonically across the grid, each in its own direction; the third (height on the hemisphere) peaks in the middle of the grid, so the map has folded onto the surface. 103
5.11 Largest extent of the prototypes as a function of ϑ. 103
5.12 A 6×6 fuzzy SOM of the wine data: U-matrix (dark ridges separate groups of similar prototypes), fuzzy hit histogram, and the component planes of two attributes. 104
5.13 Hit histograms of the three cultivars on the same map: each occupies its own region, separated by the ridges of the U-matrix. 104
5.14 FUZZSAM pictures of FCM partitions of the breast cancer data with two, three, and four clusters (colors: diagnosis). 104
5.15 Convergence of the fuzzy Sammon stress for different step sizes; a larger α speeds up the decrease, and the safeguarded step keeps every run monotone. 105

6.1 Three prototypes and the distance each of them induces; the dashed curves are level sets of the distance. 108
6.2 Relative versus absolute membership along the line through two prototypes at 0 and 4 (both with $m = 2$ and, for PCM, $\eta_i = 1$). 109
6.3 Noise clustering adds a fictitious prototype $\mathbf{v}_*$ that has no position but is at the same distance δ from every sample. 109
6.4 FCM memberships (left) and PCM typicalities (middle) on three groups with 60 outliers: the FCM contours tile the plane and the centers are pulled outward, the PCM contours are bumps around the prototypes. 131
6.5 Along a line from one center through another and beyond, the FCM memberships cross at the midpoint and stay high beyond the prototypes, whereas the PCM typicalities peak at the prototypes and vanish on both sides. 131

6.6 Center error (left) and mean maximal typicality of the outliers (right) against the scale factor K for three exponents: $m = 1.5$ with $K \leq 1$ is robust, too large a K lets the outliers back in, and too large an m makes the typicality decay too slowly. . . . 131
6.7 Coincident clusters: a large group next to a small, close one. . . . 132
6.8 PFCM on the noisy blobs: the memberships (left) sum to one and partition the plane, the typicalities (right) are bumps around the prototypes and flag the outliers. . . . 133
6.9 Left: center error on the noisy blobs against the typicality weight b (with $a = 1$); $b = 0$ is FCM. . . . 133
6.10 Partitions of the coincident-cluster data at $K = 4$: PCM merges the prototypes, PFCM with $a = b = 1$ nearly merges them, and PFCM with $a = 20$ keeps them on their groups. . . . 133
6.11 FCM (left) and noise clustering with $\lambda = 0.3$ (middle) on the noisy blobs; the dashed circles have radius δ and the gray samples are labeled noise. . . . 134
6.12 Memberships along a probe line through two centers: in noise clustering the real memberships decay and the noise membership (dashed) takes over beyond δ. . . . 134
6.13 Number of true outliers and of inliers labeled noise (left) and center error (right) against the multiplier λ. . . . 134
6.14 Partitions for $\lambda = 0.05$, 0.3, and 2: a small λ peels the tails off the groups, a large one lets every outlier back in. . . . 135
6.15 FCM (left) and kernel FCM with feature-space prototypes (right) on two rings (top) and two moons (bottom); contours of the maximal membership evaluated through the kernelized distance. . . . 136
6.16 Agreement with the true groups against the kernel parameter γ on the rings ($m = 1.2$) and the moons ($m = 1.1$); the dashed line is FCM. . . . 136
6.17 The rings with $\gamma = 0.05$ (FCM-like halves), $\gamma = 0.3$ (the rings), and $\gamma = 5$ (orthogonal feature space, arbitrary partition). . . . 136
6.18 Kernel FCM with input-space prototypes on the noisy blobs: the saturating distance makes it a robust FCM (left), for a wide range of γ (right). . . . 137
6.19 Three lines through one point: FCM tiles the plane (left), GK cuts the lines into ellipsoidal pieces (middle), and fuzzy c-lines finds the lines (right). . . . 137
6.20 Fuzzy c-lines: the maximal membership is a band around each line (left); the eigenvalues of the fuzzy scatter matrices separate the spread along the lines from the scatter around them (right). . . . 138
6.21 Two collinear segments and an oblique one, clustered with $\alpha = 0$ (left; the collinear segments form one line) and with $\alpha = 0.3$ (middle), and the agreement against α for these data and for the crossing lines (right). . . . 138
6.22 FCM on two concentric rings (left) and FCSS on the same rings (middle) and on three circles, two of which intersect (right); the fitted circles and their centers are drawn. . . . 139
6.23 Maximal membership around the two fitted rings (left) and the signed radial residuals of the samples from their own shell on the intersecting circles (right). . . . 139

6.24 Concentric rings from the FCM-hardened start with a single run (left; two circles through both rings) and with five restarts (middle), and the agreement over ten seeds against the number of restarts for three initializations (right). 139
6.25 Three concentric rings of radii 1, 1.6, and 3 with FCSS (left) and FCS (right), both from the best of ten restarts. 140
6.26 FCM (left) and fuzzy clustering with volume prototypes (right) on a large diffuse group and two small dense ones; the dashed circles are the cores, inside which the distance is zero. 140
6.27 Memberships along the line from the center of the large group to the center of the smallest one: with volume prototypes (solid) the memberships are one inside the shaded cores and the border moves toward the small cluster; FCM (dotted) for comparison. 141
6.28 Adaptive cluster merging: eight volume prototypes before merging (left) and the three clusters with their cores after five merges (right). 141
6.29 Parameter study: agreement with the true groups against the radius scale κ (left), final number of clusters against the similarity threshold (middle) and against the starting number of clusters (right). 142
6.30 Two informative and six noise features. 142
6.31 SCAD-2 on clusters in different subspaces: level sets of the weighted distance of every cluster (left) and the weights (right); each band weights the feature along which it is tight. 143
6.32 Parameter study: agreement with the true groups against the number of noise features for FCM, the weighted FCM, and SCAD-1 (left), and agreement and total weight of the informative features against the weight exponent t of the weighted FCM with four noise features (right). 143

7.1 Clustering with the minimal spanning tree. 147
7.2 Geodesic distance on a neighborhood graph. 148
7.3 The minimal spanning tree of two groups joined by a chain (left) and the single edge that Zahn's local criterion removes (right, dashed): the chain stays attached to one group. 159
7.4 The volume criterion cuts the chain (orange dashed edges) and separates it from the groups; the length-criterion cut is drawn in vermilion. 160
7.5 Gath–Geva clustering initialized with the tree partition (left) polishes it into a fuzzy partition with covariance ellipses; the same GG from a random start (right) spends one Gaussian on the chain and collapses another onto a few points of the left group. 160
7.6 Left: the number of tree clusters against the required relative volume gain for several minimum cluster sizes. 160
7.7 Length criteria on a diffuse group and two tight groups. 161
7.8 Edge lengths of the tree with the two global thresholds (dashed) and the two edges removed by the local criterion (vermilion): the edge between the tight groups lies among the long edges of the diffuse group. 161

7.9 Number of clusters found by the ratio form (left) and the deviation form (right) of Criterion 2 as a function of the factor and the neighborhood depth; the dotted line is the true number. . . . 162
7.10 Volume-based validity indices of the MST–GG partitions of four groups against the number of clusters: fuzzy hypervolume, partition density, average partition density and the relative decrease of the hypervolume with the default minimum gain (dashed); the dotted line is the number the automatic run picks. . . . 163
7.11 Two rings: the tree partition with two clusters (left), the failed Gath–Geva refinement of two clusters (middle) and the twelve Gaussian arcs the volume criterion produces (right). . . . 163
7.12 Fuzzy similarity of the twelve arcs (left) and its single-linkage dendrogram (right): the two branches are the two rings. . . . 163
7.13 Fuzzy c-means (top) and geodesic c-medoids with $k = 8$ (bottom, with the neighborhood graph and the bridge of the two-component cases) on two spiral arms, two moons and two rings: only the geodesic clusters follow the manifolds. . . . 164
7.14 Memberships of the spiral samples in the two geodesic clusters: they fade along the arm, not across it. . . . 165
7.15 Agreement with the true groups (left) and number of graph components (right) against the number of neighbors k for the four data sets: good results need a k between fragmentation and short-circuits. . . . 165
7.16 Noisier moons with $k = 4$ and $k = 15$: the gap is bridged by the graph itself, and the tips of the moons follow the other medoid. . . . 165
7.17 Shortest paths in the neighborhood graph of the spiral: along one arm from its inner to its outer end (left) and between the closest outer pair of samples of different arms (right); the dotted lines are the Euclidean chords. . . . 166
7.18 Eight clusters on the S-curve: FCM centers float between the layers (left), the re-projected centers of the Isomap route (middle) and the geodesic medoids (right) lie on the sheet. . . . 166
7.19 The Isomap projection of the S-curve with the FCM centers (left) and the geodesic c-medoid clusters in the same projection (right): both tile the unrolled sheet regularly. . . . 167
7.20 Isomap projection of the spiral with $k = 4$, where the arms are fragmented into seven components: penalized bridges keep the fragments apart (left), plain bridges re-join the fragments of each arm (right); colors are the true arms, crosses the FCM centers. . . . 167

8.1 Product-space clustering for Takagi–Sugeno model identification. . . . 171
8.2 Projected Gath–Geva cluster and modified Gath–Geva prototype. . . . 173
8.3 Supervised fuzzy partition of one feature. . . . 180
8.4 Hinging-hyperplane regression tree. . . . 184
8.5 Four rules identified by modified Gath–Geva clustering on the sinc function. 189

8.6 The same data with four rules from the modified GG algorithm, from Gath–Geva clustering with projected antecedents and from Gustafson–Kessel clustering with projected antecedents (shaded bands: $\pm\sigma_i$ of the sets). . . 189
8.7 The product space (x, y): Gustafson–Kessel clusters are oblique ellipses whose width along x is not the width of their operating region (left); modified Gath–Geva clusters are Gaussians in x carrying a line (right). 190
8.8 RMSE on the noisy training data and on noise-free test data as a function of the number of rules; apart from the poorer five-rule solution, both errors fall, and the test error keeps falling up to twelve rules. 190
8.9 The second-order plant is identified on data with a random input (left part) and validated on data with a sinusoidal input (right part). 191
8.10 Validation on the sinusoidal input: one-step-ahead prediction and free-run simulation of the four-rule model follow the plant closely although the model was identified under random excitation. 191
8.11 The antecedent membership functions of the four rules tile the ranges of $y(k-1)$ and $y(k-2)$. 191
8.12 True nonlinearity $f(y_1, y_2)$ (left), TS model (middle) and operating regimes with the rule centers (right); dots are the identification samples. 192
8.13 Training and free-run error as a function of the number of rules for the two restart criteria. 192
8.14 Free-run simulation of the three-rule MIMO TS model on validation data with random step inputs; both outputs are simulated jointly. 193
8.15 The antecedent sets on the two scheduling variables $y_1(k-1)$ and $y_2(k-1)$ divide the operating range into three regimes. 193
8.16 Free-run RMSE versus the number of rules for scheduling on the outputs, on the inputs and on all regressors: scheduling on the inputs brings little. 194
8.17 Two crossing lines: FCM cuts the points into two halves, GK follows the lines with two elongated ellipsoids, and FCRM clusters the pairs with regression prototypes and recovers the lines themselves. 194
8.18 Left: membership in the first model along x, fuzzy only where the lines cross. 195
8.19 Three lines, two of them parallel: the worst of twelve random starts averages the parallel lines (left); the best restart recovers all three (right). 195
8.20 Effect of the exponent m: the partition becomes fuzzier, the agreement with the generating lines stays. 195
8.21 Decision regions of the fuzzy classifier with two rules (left) and four rules (right) on two classes made of two blobs each; dashed ellipses are the $\pm\sigma$ extents of the rules. 196
8.22 Posterior probability of class 1 and the 0.5 boundary of the four-rule classifier: the posterior changes smoothly between the rules. 196
8.23 The antecedent sets of the four rules come in pairs on each of the two features. 197
8.24 Cross-validated accuracy versus the number of rules without and with variance regularization: four rules are needed, and the regularized classifier keeps its accuracy for many rules. 197

8.25 Backward elimination of the wine attributes by interclass separability: the training accuracy of the three-rule classifier drops only when fewer than four attributes remain. 197
8.26 Supervised fuzzy partition (top) and Ruspini partition (bottom) of three features, with the class-conditional samples as rugs. 198
8.27 Supervised and Ruspini sets of the petal length of the iris data: the supervised sets switch between the species. 199
8.28 Cross-validated accuracy of the tree and total number of sets as functions of the initial number of sets, for three merging thresholds: three to five initial sets suffice. 199
8.29 One hinge from FCRM with two models on one input: the two lines (dashed), the hinge function (black) and the memberships of the two models, which are the fuzzy operating regions. 200
8.30 True piecewise-linear surface with the samples (left) and the prediction of a depth-two hinge tree with its hinges (right): the hinges of the internal nodes in black, those of the leaves dashed white. 200
8.31 Test RMSE versus the number of leaves for balanced and greedy growth. 201

9.1 Fuzzy segmentation as clustering with a time-aware prototype. 205
9.2 The false-nearest-neighbor idea and its clustering-based threshold. 212
9.3 State-space reconstruction by delay embedding and clustering. 214
9.4 The six observed variables of the synthetic series change regime at the dashed lines; the first change alters only the correlation and is visible in no single variable. 223
9.5 The scree plot (left) and the cumulative fraction of variance (right) of the ten initial segments show two dominant components above a noise floor; the dashed line marks an accuracy of 95 %, which two components reach only in the segment dominated by a mean shift. 223
9.6 Two variables colored by the hard segment, the Gaussian segments in time $\beta_i(t_k)$ and the posterior memberships show the fuzzy segmentation with four segments; dashed lines mark the crisp borders. 224
9.7 The borders found for different numbers of segments (left, $q = 2$) and of components (right, $c = 4$) are compared with the true change points (dashed lines). 224
9.8 The compatibility matrix of the ten initial segments is shown on the left, the fuzzy segments after merging to four with $\gamma = 0.75$ on the right. 225
9.9 The borders found by the bottom-up algorithm with the Q cost, by the bottom-up algorithm with the T^2 cost and by the fuzzy segmentation, all with four segments, are compared with the true change points (gray dashed lines). 225
9.10 The cost of every merge of the bottom-up algorithm with the Q cost (left) and the T^2 cost (right) is plotted against the number of segments left; the jump marks the merge of two different regimes. 226
9.11 The polymerization reactor benchmark is driven by a random initiator flow (top) and observed through the molecular weight (bottom). 226

9.12 The table gives the percentage of false nearest neighbors over the orders with the cluster-based threshold; the frame marks the selected order $(n_a, n_b) = (1, 2)$. 227
9.13 The point-wise thresholds R_k of the order $(1, 2)$ are plotted along the lagged input and colored by cluster (left), and their distribution is shown on the right. 227
9.14 The percentage of false nearest neighbors with the cluster-based threshold, with three constant thresholds and with the classical test of Kennel et al. shows that the corner moves with the constant. 227
9.15 The normalized one-step error of linear ARX models over the orders confirms the selected order with a model-based check. 228
9.16 The Rössler system is observed through x_1 (left); its true attractor is shown in the (x_1, x_2) plane (right). 229
9.17 The embedding scan plots the one-step training and test errors (top) and the prior-weighted smallest eigenvalue J^{d_e} (bottom) against the embedding dimension and marks the selected $d_e = 3$ and $d_l = 2$. 229
9.18 The reconstructed state space is shown with the twelve regimes and their centers (left) and with the free-run simulation of the test part (right). 229
9.19 The free-run prediction of the test part is compared with the measured series in time (left), and its error is plotted against the horizon (right). 230
9.20 The delay vectors and the regime centers are projected to the plane by PCA (left) and by the fuzzy Sammon mapping (right). 230
9.21 The variance–complexity trade-off is plotted for the 30 best structures of the mixed-input problem; the framed dip is the recommended structure $\{x_1, x_2, x_3, x_4\}$. 231
9.22 Fuzzy c-means gives three fuzzy sets of the continuous input x_1 (left), and the binary input x_2 has two crisp sets (right). 231
9.23 The variance–complexity trade-off is plotted for the 70 best structures of the Friedman function with five dummy inputs; the dip at structure six is the true set of inputs. 231

10.1 Object–feature and object–object data. 234
10.2 The β-spread of NERF c-means. 235
10.3 Mean, medoid, and mode. 236
10.4 FCM on the coordinates (left) and RFCM on the matrix of squared distances with its dual centers (middle) give the same partition; the objective trajectories coincide (right). 250
10.5 The relational prototypes are weight vectors over the objects (left); the squared-distance matrix ordered by cluster shows three blocks (right). 251
10.6 NERF on powers of the Euclidean distances. 251
10.7 Three shape classes with random time warps and shifts: a bell, a step, and a double bump. 252
10.8 Euclidean (left) and DTW (middle) dissimilarities ordered by the true class, and the DTW alignment of two bells (right): the class blocks are clear only under DTW. 252

10.9 Fuzzy c-medoid clusters on the DTW matrix (top) and on the Euclidean matrix (bottom), with the medoid series in black. 252
10.10 Memberships of the DTW c-medoid clustering, with the series ordered by the true class. 253
10.11 Linear fuzzy c-medoids: final objective (left) and time of thirty restarts (right) against the number of medoid candidates. 253
10.12 Fitted modes with the share of the membership mass on the mode (left) and the matching dissimilarity of the respondents to their group's mode (right). 253
10.13 Membership mass of the answers to question q1 in each cluster; the mode is the answer with the largest mass. 254
10.14 Agreement with the groups and mean maximal membership of fuzzy k-modes against the exponent m. 254
10.15 The objective of fuzzy k-modes against the number of clusters has its knee at the three planted groups. 254
10.16 Mixed data: the numeric part colored by the true group, with the planted centers as crosses (left), and the planted modes of the categorical part with their frequencies (right). 255
10.17 Fuzzy k-prototypes with the default weight: the numeric part of the partition with the prototypes (left) and the modes of the categorical part (right). 255
10.18 The weight γ interpolates between fuzzy c-means and fuzzy k-modes. 256
10.19 Gower's dissimilarity (left) and the Euclidean distance of the one-hot encoded records (right), ordered by the true group. 257
10.20 Gower with fuzzy c-medoids (left, medoids as crosses), one-hot encoding with FCM (middle), and fuzzy k-prototypes (right), shown in the numeric plane. 257
10.21 Agreement with the groups against the weight of the numeric attributes in Gower's coefficient, for fuzzy c-medoids and NERF c-means. 257
10.22 Two series of each process (top) and the estimated autocorrelation functions of all series of a class with the theoretical one (bottom). 258
10.23 A-FCM in the plane of the first two autocorrelations, with centers, A-FCMd medoids, and the theoretical values (left), and the memberships ordered by the true class (right). 258
10.24 Mean memberships of new AR(1) series in the fitted clusters against their coefficient ϕ. 259
10.25 Agreement with the processes (left) and mean maximal membership (right) against the number of lags L for four series lengths T. 259

11.1 Partial distances and the completion of a missing value. 262
11.2 The chunk pipelines of single-pass and online FCM. 263
11.3 Spatial constraints for image segmentation. 264
11.4 The four strategies on data with 25 % missing values. 278
11.5 The PDS partition at the true coordinates (left) and the maximal membership of the incomplete samples (right): an ambiguous observed coordinate gives a low membership. 278

11.6 Prototype error against complete-data FCM and agreement with the true groups for missing rates up to 60 % (four-dimensional blobs), and the share of complete samples. 278
11.7 Imputed against true values with 20 % of the values missing, for PDS (post hoc), NPS, and OCS from a random and from the PDS start. 279
11.8 Memberships of the incomplete samples under OCS and PDS (left), and the largest imputation error of a sample against its PDS membership (right). 280
11.9 Imputation error against the missing rate (left) and against the exponent m at 30 % missing (right), with the column mean and the oracle as references. 280
11.10 Literal FCM and the three schemes on 200 000 samples. 281
11.11 The spFCM prototypes after every chunk, marker area proportional to the carried weight (left), and the oFCM chunk prototypes relative to the merged ones (right). 281
11.12 Speedup over literal FCM (left) and agreement with its partition (right) against the sample or chunk size. 281
11.13 The first and the last chunk of a drifting stream with the true center paths (dotted), the prototype paths of spFCM with $\lambda = 1$ and $\lambda = 0.5$, and the final prototypes of the three models. 282
11.14 Distance of the prototypes from the current true centers along the stream. 282
11.15 Mean tracking error over the second half of the stream against the forgetting factor for four drift speeds; circles mark the best factor. 282
11.16 Seven segmentations of an image with Gaussian noise. 284
11.17 The same image with 20 % salt-and-pepper noise: averaging spreads the impulses, whereas the median and the adaptive weights remove them. 285
11.18 Segmentation accuracy against the level of Gaussian (left) and salt-and-pepper noise (right). 285
11.19 EnFCM with increasing weight and window size. 286
11.20 Interior (solid) and boundary (dashed) accuracy against α for FCM_S, EnFCM, and FCM_S2 under Gaussian (left) and salt-and-pepper noise (right). 286
11.21 Interior and boundary accuracy against the window size with $\alpha = 4$ under Gaussian noise. 286

12.1 Two kinds of partial supervision. 288
12.2 Co-clustering of an object–item matrix. 289
12.3 Collaborative clustering of three data sites. 290
12.4 From a library of soft partitions to a consensus. 290
12.5 FCM splits the two bars across (left); with 10 % labeled samples (large markers) and $\alpha = 1$ the partition turns toward the bars (right). 305
12.6 Membership in the upper-bar cluster along the bars: clear in the middle, ambiguous at the ends; labeled samples (black) are pulled toward their label. 305
12.7 Agreement with the true bars against the share of labeled samples for several weights α: more labels help, and moderate weights help most. 306

12.8 The 24 constraints that FCM violates (top) and the partition of PCCA with $\alpha = 5$, which satisfies all 100 constraints (bottom); green: must-link, vermilion dashed: cannot-link. 306
12.9 Agreement with the classes (left) and share of satisfied constraints (right) against the penalty α for 10 to 100 pairs of each kind. 307
12.10 Competitive agglomeration from eight clusters: final number of clusters against η_0 for three decays (left) and the three clusters found with $\eta_0 = 5$ (right). 307
12.11 A shuffled document–term matrix (left) and the same matrix reordered by the four FCCM co-clusters (right). 308
12.12 Scaled term memberships of the four co-clusters: every topic owns its vocabulary, and the general terms (right of the dotted line) belong to none. 308
12.13 FCCM on a grid of the two entropy weights: agreement with the topics (left) and crispness of the document memberships (right). 308
12.14 Fuzzy CoDoK: the 40 terms of largest scaled membership, grouped by co-cluster. 309
12.15 The three sites clustered independently (top) and with collaboration $\alpha = 3$ (bottom); colors are the clusters matched across the sites. 309
12.16 Against the collaboration weight: fuzzy Rand index between the sites (left), agreement of every site with the true groups (middle), and mean membership change of every site (right). 310
12.17 Twenty weak members and their consensus: distribution of the member ARIs with the consensus and the member mean (left), co-association matrix ordered by the consensus (right). 310
12.18 ARI of the consensus against the library size for the five consensus functions. 311
12.19 Fuzzy adjusted Rand index between repeated consensus partitions against the library size: the ensemble makes an unstable clusterer reproducible. 311
12.20 Evidence accumulation on five groups: the consensus with the number of clusters chosen by the lifetime (left), the co-association matrix (middle), and the lifetimes of k clusters (right). 312
12.21 Top of the average-link dendrogram of the co-association distances; the band is the range of thresholds that yields five clusters. 312
12.22 Number of clusters chosen by the lifetime against the range of the members' numbers of clusters, for average and single link. 312

List of Tables

2.1 Notation of the book and the corresponding names in `fuzzyclust`. . . . 19

3.1 The four partitioning algorithms at a glance. . . . 38

4.1 Validity indices of `fuzzyclust.validity`. . . . 71

4.2 Validity scan of FCM ($m = 2$) on five Gaussian clusters, $c = 2, \ldots, 9$ (script `plot_05_validity_scan.py`). . . . 81

5.1 Quality of three projections of the FCM partition ($c = 3$, $m = 2$) of the range-scaled wine data: mean membership error P, partition coefficient of the fitted (F) and of the recomputed (F^*) partition, Sammon stress E. . . . 101

10.1 Which algorithm for which data. . . . 246

B.1 MATLAB toolbox functions and their `fuzzyclust` counterparts. . . . 330

B.2 Fields of the MATLAB `param` structure and the corresponding constructor arguments. . . . 331

B.3 Fields of the MATLAB `result` structure and the fitted attributes. . . . 332

CHAPTER

1



Introduction

This chapter explains what clustering is good for, why fuzzy partitions are often the more honest description of data, and where the `fuzzyclust` toolbox comes from. The toolbox is the Python successor of the *Fuzzy Clustering and Data Analysis Toolbox for MATLAB* and puts two decades of clustering-based research of the author's group (model identification, time-series analysis, visualization, and graph-based methods) and the classical methods of the literature behind one scikit-learn compatible interface. The chapter then describes how the book is organized and how to read it, and closes with the installation and a short first session. The mathematics starts in chapter 2.

1.1 What clustering is for

Cluster analysis groups objects by their mutual similarity: objects in the same group should resemble each other more than objects in different groups. It is *unsupervised*—no class labels are used—and this is its main asset. Where a classifier reproduces categories that somebody has already defined, clustering reveals a structure nobody has defined yet: the operating regimes of a chemical reactor, the types of customers behind a sales table, the segments of a long measured signal, the local linear behaviors of a nonlinear system. Ruspini [1], Dunn [2], and Bezdek [3] formulated fuzzy clustering as the minimization of an objective function, and the monographs by Bezdek [3], by Jain and Dubes [4], by Kaufman and Rousseeuw [5], and by Höppner et al. [6] remain the standard references.

In this book the objects are the N rows $\mathbf{x}_k \in \mathbb{R}^n$ of a data matrix $\mathbf{X} \in \mathbb{R}^{N \times n}$; the columns are measured variables (*features*). What the rows mean depends on the application, and so does the meaning of a cluster:

- In *exploratory data analysis* the rows are independent observations (patients, products, samples), and a cluster is a group of similar observations. The analyst wants to know how many groups there are, how they differ, and which observations lie between them.
- In *pattern recognition and data reduction* the prototypes of the clusters stand in for the data: a few hundred prototypes summarize a million samples, and the memberships of a new sample in the clusters serve as features for a subsequent classifier or regression model.
- In *system identification* the rows $[\mathbf{x}_k^\mathsf{T}\ y_k]^\mathsf{T}$ join a regression vector, built from lagged inputs and outputs of a dynamic system, with the corresponding output. A cluster in this product space is a region where one local linear model fits, and the fuzzy clustering of the product space yields a Takagi–Sugeno model with the memberships as rule firing strengths [7, 8]. Here clustering is not the end result but the structure-identification step of a model.
- In *time-series analysis* the rows are consecutive samples or delay vectors of a signal; a cluster is a segment with homogeneous behavior, and the memberships along time show when one regime gives way to the next.

The toolbox and the book cover all four uses. Chapters 2–4 develop the partitioning algorithms and their validation for the first two uses, chapter 8 the identification use, and chapter 9 the time-series use; the visualization and graph methods of chapters 5 and 7 serve all of them.

1.2 Hard and fuzzy partitions in one picture

A *hard* (crisp) partition assigns every sample to exactly one cluster; the clusters are disjoint subsets of the data. A *fuzzy* partition [1, 9] assigns every sample a membership degree $\mu_{ik} \in [0, 1]$ in every cluster, with $\sum_i \mu_{ik} = 1$. Figure 1.1 shows why the second description is often the more faithful one. Two groups of points overlap. The hard partition (top left) draws a boundary and puts every sample on one side of it, although the samples near the boundary are no more similar to their own prototype than to the other one. The fuzzy partition (top right) preserves the degree of ambiguity of every sample. Along the line through the prototypes (bottom), the hard partition is a step function at the midpoint, whereas the fuzzy partition is a smooth membership function whose steepness is set by the fuzziness exponent m of chapter 3.

Three practical consequences follow, and they are the reasons this book is about *fuzzy* clustering. First, the memberships carry information that a label cannot: the sample with memberships $(0.49, 0.51)$ and the sample with $(0.02, 0.98)$ receive the same hard label but mean different things, and a downstream model (a classifier, a local linear model, a monitoring rule) can use the difference. Second, the fuzzy objective function is differentiable in the memberships, which makes closed-form update equations and convergence proofs possible [10, 11]; hard partitions have

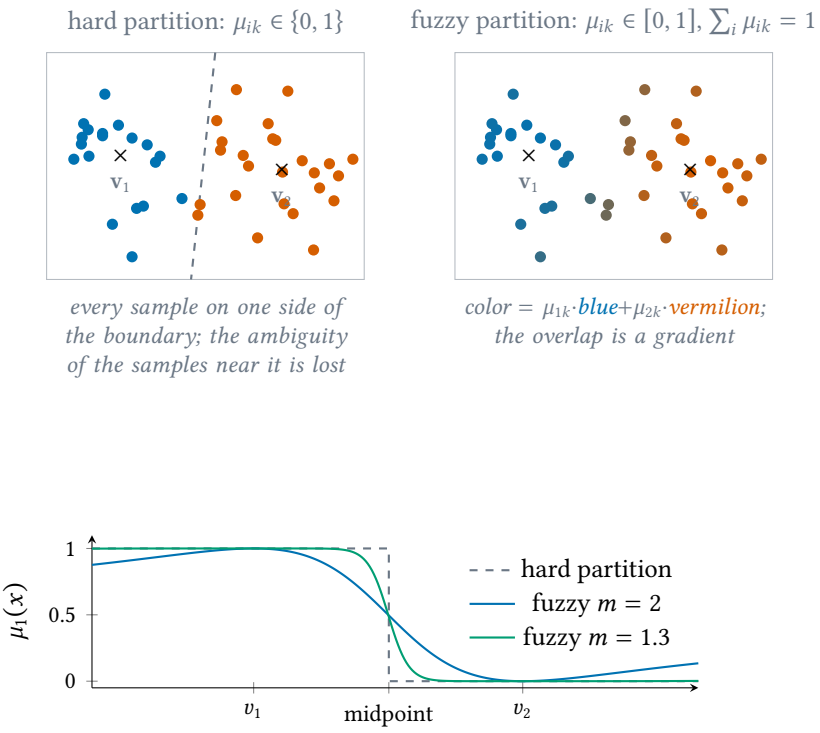


Figure 1.1. Hard and fuzzy partitions of two overlapping groups. Top left: a hard partition assigns every sample to the nearer prototype; the dashed boundary is the perpendicular bisector of $\mathbf{v}_1\mathbf{v}_2$. Top right: the fuzzy c-means partition of the same data, with every sample colored by the blend of its two memberships; the cores are pure, and the overlap region is a gradient. Bottom: along the line through the prototypes, a hard partition is a step and a fuzzy partition is a smooth membership function whose steepness grows as the fuzziness exponent m approaches one.

to be searched combinatorially. Third, fuzzy prototypes are weighted means over *all* samples, so they move smoothly when the data change, which matters for identification and for tracking slowly drifting processes.

1.3 From the MATLAB toolbox to fuzzyclust

`fuzzyclust` descends from the *Fuzzy Clustering and Data Analysis Toolbox for MATLAB* of Balaskó, Abonyi, and Feil [12]. That toolbox implemented, in the notation of Babuška's monograph [7], the hard k-means and k-medoids algorithms; the fuzzy c-means (FCM), Gustafson–Kessel (GK), and Gath–Geva (GG) algorithms; the evaluation of new data with fitted prototypes (`clusteval`); seven validity indices; data normalization; and three projection methods for the visualization of clustering results: principal component analysis, Sammon mapping, and the fuzzy

Sammon mapping (FUZZSAM) of Kovács and Abonyi. It became the companion of the book *Cluster Analysis for Data Mining and System Identification* [8] and the basis of a line of research in which clustering is the engine of other tasks.

Figure 1.2 shows how the Python toolbox grew out of it. The core of the MATLAB toolbox maps one-to-one onto the sub-packages `cluster`, `validity`, `preprocessing`, `plotting`, and `viz` (appendix B gives the mapping function by function). Around this core, four research threads of the group are collected:

Identification (`fuzzyclust.models`).
Takagi–Sugeno fuzzy models identified by clustering the product space of inputs and outputs with a modified Gath–Geva algorithm, fuzzy c-regression models, supervised fuzzy clustering for classification, fuzzy partitions for decision trees, and hinging-hyperplane trees.

Time series (`fuzzyclust.timeseries`).
Fuzzy segmentation of multivariate time series by clustering with time as an extra dimension, model-order selection with false nearest neighbors and clustering, state-space reconstruction of chaotic systems, and structure selection for mixed-input models.

Visualization (`fuzzyclust.viz`).
FUZZSAM and its quality measures, the fuzzy self-organizing map as a regularized FCM, and projection diagnostics for the number of clusters.

Graph methods (`fuzzyclust.graph`).
Minimal-spanning-tree clustering with fuzzy hypervolume splitting and Gath–Geva refinement, and geodesic-distance fuzzy c-medoids on neighborhood graphs.

A fifth sub-package, `fuzzyclust.extensions`, adds classical variants that the MATLAB toolbox did not have: possibilistic c-means, noise clustering, kernel FCM, fuzzy c-varieties and c-shells, volume prototypes, and feature weighting. Eight further sub-packages take the methods beyond a complete numeric data matrix and a single unsupervised run: `relational`, `incomplete`, `large`, `image`, `semisupervised`, `coclustering`, `collaborative`, and `ensemble` (chapters 10 to 12).

Two design decisions shape the package. First, almost every clustering algorithm is a class derived from one base class that implements the alternating optimization once; an algorithm only defines how its prototypes are updated and how its distances are measured (chapter 2). Second, the estimators follow the scikit-learn conventions [13]: `fit`, `predict`, `predict_proba`, and `transform` behave as in scikit-learn; pipelines, cloning, grid search, and pickling work without extra code; and a `pandas` data frame is accepted as input and, on request, returned by `transform`. The MATLAB function `clusteval`, which evaluated new data with fitted prototypes, has become `predict_proba`.

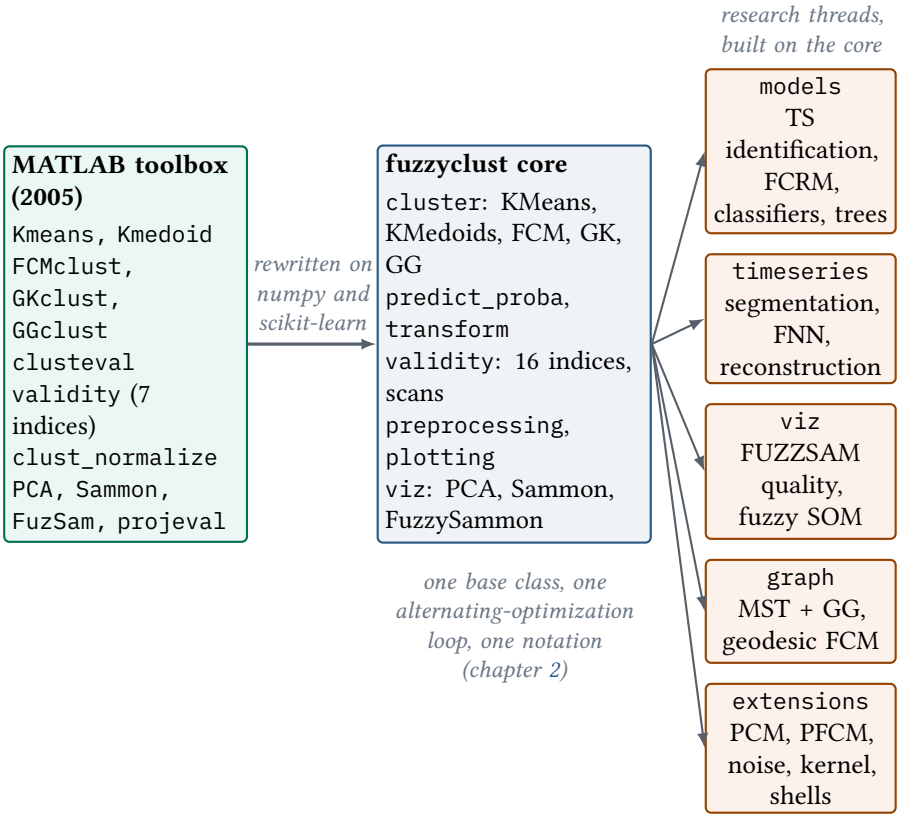


Figure 1.2. Lineage of the toolbox. The functions of the MATLAB toolbox became the core sub-packages of `fuzzyclust`; the clustering-based research of the group (identification, time series, visualization, and graph methods) and the classical extensions are built on the same base class and the same notation.

1.4 How the book is organized

Chapter 2 lays the common ground: the data matrix; the distance norms; the hard, fuzzy, and possibilistic partition spaces; the objective-function view of clustering, with alternating optimization as the template of all algorithms; initialization, termination, and scaling; and the notation used throughout. It ends with the scikit-learn interface that the estimators of the toolbox share.

Chapter 3 presents the partitioning algorithms that carry the toolbox: k-means and k-medoids as hard reference methods; fuzzy c-means, with the derivation of its update equations and the role of the fuzziness exponent; the Gustafson–Kessel algorithm, with its adaptive norm, its volume constraint, and the covariance regularization of Babuška, van der Veen, and Kaymak; and the Gath–Geva algorithm, with its exponential distance, its priors, and its need for a good initialization. It closes with alternative fuzzifiers and with mountain and subtractive clustering, which estimate the number and location of the clusters.

Chapter 4 asks how many clusters there are. It describes the sixteen validity indices of the toolbox (partition-based, compactness and separation, crisp, and volume-based) with their optimum directions and weaknesses, the scan over the number of clusters, compatible cluster merging and stability-based selection as alternatives to the scan, and the evaluation of new data as a form of validation.

Chapter 5 puts a clustering on paper: principal component projection, Sammon mapping, the fuzzy Sammon mapping that preserves the sample–prototype distances of the algorithm, measures of projection quality, and the fuzzy self-organizing map.

Chapter 6 extends the prototype and the partition: possibilistic and noise clustering for data with outliers, kernel FCM, fuzzy c-varieties and c-shells for linear and circular clusters, volume prototypes, and feature weighting.

Chapter 7 abandons the prototype as the starting point: minimal-spanning-tree clustering with fuzzy hypervolume splitting and Gath–Geva refinement, and fuzzy c-medoids with geodesic distances on neighborhood graphs for clusters that follow a manifold.

Chapter 8 turns clustering into model identification: Takagi–Sugeno models from Gath–Geva clustering of the product space, fuzzy c-regression models, supervised fuzzy classifiers, fuzzy partitions for decision trees, and hinging-hyperplane trees.

Chapter 9 applies clustering to signals: fuzzy segmentation of multivariate time series, model-order selection by false nearest neighbors, state-space reconstruction of chaotic systems, and structure selection for models with mixed inputs.

Chapter 10 gives up the numeric feature vector: relational and NERF c-means work from a dissimilarity matrix alone, fuzzy c-medoids accepts any dissimilarity, and fuzzy k-modes and k-prototypes cluster categorical and mixed records.

Chapter 11 adapts fuzzy c-means to incomplete data, to very large data sets and streams, and to image segmentation, where the neighborhood of a pixel enters the objective function.

Chapter 12 goes beyond a single unsupervised run: partial supervision by class labels or pairwise constraints, co-clustering of objects and items, collaborative clustering across data sites, and fuzzy cluster ensembles.

Three appendices complete the book: a compact reference of every public class and function (appendix A), the migration table from the MATLAB toolbox (appendix B), and the catalog of example scripts (appendix C).

1.5 How to read this book

Every chapter has the same structure. A *roadmap* says what the chapter covers and how it connects to the previous one. *Problem and idea* motivates the methods in words and with one concept figure. *Theory* develops the methods in the unified

notation, states every algorithm in an algorithm box, and closes with a summary box. *Using it in fuzzyclust* lists the parameters and attributes of the classes in API boxes, with code snippets taken verbatim from the example scripts. *Examples* discusses the figures that the example scripts produce, with emphasis on what happens when the main parameters change. *Further reading* points to the original papers. Theory and experiments are kept apart on purpose: the theory sections can be read without running anything, and the example sections can be run without reading the theory.

Notation at a glance

$\mathbf{X} \in \mathbb{R}^{N \times n}$ is the data matrix with N samples $\mathbf{x}_k$ as rows and n features; c is the number of clusters; $\mathbf{v}_i$ is the prototype of cluster i and $\mathbf{V}$ the matrix of prototypes; μ_{ik} is the membership of sample k in cluster i and $\mathbf{U} = [\mu_{ik}] \in \mathbb{R}^{c \times N}$ the partition matrix; $m > 1$ is the fuzziness exponent; D_{ik}^2 is the squared distance of $\mathbf{x}_k$ to $\mathbf{v}_i$ in the norm of the algorithm; and J is the objective. Index i (and j) always runs over clusters, k over samples. In code the partition is stored transposed as `membership_` of shape `(n_samples, n_clusters)`, in line with `predict_proba`. The full table is in table 2.1.

The figures that show computed results are produced by the example scripts in the `examples/` directory of the repository (see the preface). The text names the script behind each such figure, and appendix C lists all scripts.

1.6 Installation and a first session

The package is installed from the Python Package Index with

```
pip install fuzzyclust
```

and depends only on `numpy`, `scipy`, `scikit-learn`, `pandas`, and `matplotlib`. A first session generates three Gaussian clusters, fits fuzzy c-means, and inspects the result:

```
import matplotlib.pyplot as plt
from fuzzyclust import FCM
from fuzzyclust.datasets import make_fuzzy_blobs
from fuzzyclust.plotting import plot_clusters, plot_membership_contours

X, y_true = make_fuzzy_blobs(n_clusters=3, n_samples_per_cluster=100,
    random_state=0)
fcm = FCM(n_clusters=3, m=2.0, random_state=0).fit(X)

print(fcm.cluster_centers_)          # prototypes v_i, shape (3, 2)
print(fcm.membership_[:5].round(3))  # memberships mu_ik of the first
    samples
```

```
print(fcm.labels_[:5])               # hard labels, argmax of the
    memberships

fig, axes = plt.subplots(1, 2, figsize=(11, 4.5))
plot_clusters(X, fcm.labels_, fcm.cluster_centers_, ax=axes[0])
plot_membership_contours(fcm, X, ax=axes[1])
plt.show()
```

The two central lines, which generate the data and fit the model, appear verbatim in the first example script, `plot_01_fcm_basics.py`:

```
X, y_true = make_fuzzy_blobs(n_clusters=3, n_samples_per_cluster=100,
    random_state=0)

fcm = FCM(n_clusters=3, m=2.0, random_state=0).fit(X)
```

Figure 1.3 shows the result. The level curves of the maximal membership $\max_i \mu_i(\mathbf{x})$ are nearly circular around the prototypes because fuzzy c-means measures every distance in the Euclidean norm. The MATLAB toolbox drew the same kind of map with `clusteval`, and in chapter 3 such maps show how differently the algorithms shape their clusters.

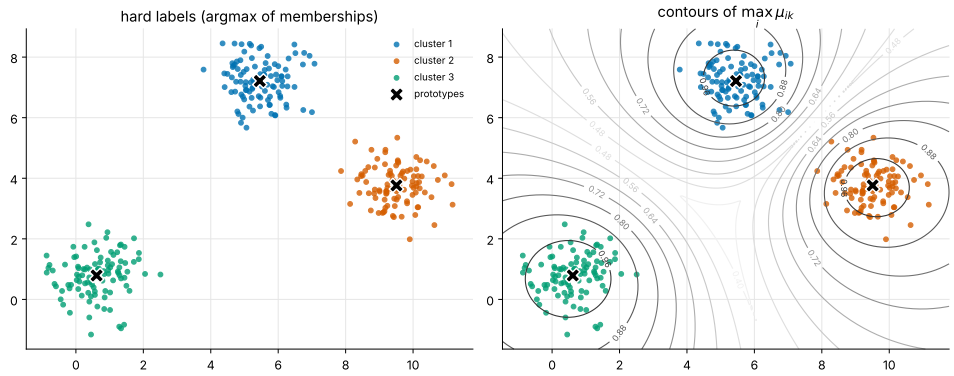


Figure 1.3. The first session: fuzzy c-means on three Gaussian clusters. Left: hard labels and prototypes. Right: level curves of the maximal membership, nearly circles around the prototypes.

1.7 Further reading

The classical monograph on objective-function clustering is Bezdek's [3]. Höppner, Klawonn, Kruse, and Runkler [6] cover the fuzzy algorithms, including shells and varieties; Jain and Dubes [4] and Kaufman and Rousseeuw [5] cover the hard and hierarchical methods. Babuška's *Fuzzy Modeling for Control* [7] is the source of the notation of this book and a standard reference on clustering-based identification; Abonyi and Feil [8] collect the research threads that became the sub-packages of the toolbox. The MATLAB toolbox manual [12] documents the predecessor, and scikit-learn is described in [13]. Zadeh introduced fuzzy sets [9], Ruspini fuzzy partitions [1], and Dunn the fuzzy c-means functional [2].

CHAPTER

2



Foundations of prototype-based clustering

The previous chapter motivated fuzzy partitions; this one lays the common ground on which every algorithm of the toolbox stands. We fix the data matrix and the notation, define the distance norms that decide what a cluster can look like, introduce the hard, fuzzy, and possibilistic partition spaces, and describe the kinds of prototypes that the later chapters fit. We then present the objective-function view of clustering, with alternating optimization as the template that all algorithms share, discuss initialization, termination, and scaling, and collect the notation in a table. The chapter closes with the scikit-learn interface shared by the clustering estimators of `fuzzyclust`; the individual algorithms follow in chapter 3.

2.1 Problem and idea

Prototype-based clustering fits a small number of geometric objects, the *prototypes*, to a large number of samples and describes every sample by its distances to the prototypes. Three ingredients are involved: the *data*, a set of N samples in $\mathbb{R}^n$; the *prototypes*, c objects of a chosen kind; and the *partition*, which says to what degree each sample belongs to each prototype. An objective function scores how well the prototypes fit the data given the partition, and an iterative algorithm improves the partition and the prototypes in turn. The algorithms of this book (hard or fuzzy, with points, ellipsoids, lines, or shells as prototypes) are instances of this one scheme, and almost all of their differences come down to two choices: *which prototype* and *which distance*.

Figure 2.1 shows the four kinds of prototypes that appear in the book. A point prototype with a fixed norm detects spherical clusters (or, with a diagonal or

Mahalanobis norm, ellipsoidal clusters that all have the same shape); this is fuzzy c-means. A point prototype with a norm adapted to each cluster detects ellipsoids of different shape and orientation (and, in the Gath–Geva algorithm, of different size); these are the Gustafson–Kessel and Gath–Geva algorithms, whose prototype is in fact the pair (center, covariance). A linear-variety prototype measures the distance of a sample to a line, plane, or hyperplane and detects flat clusters, which is what local linear models in identification need. A shell prototype measures the distance to a circle or ellipse and detects hollow clusters. The rest of this chapter is about what all of them have in common.

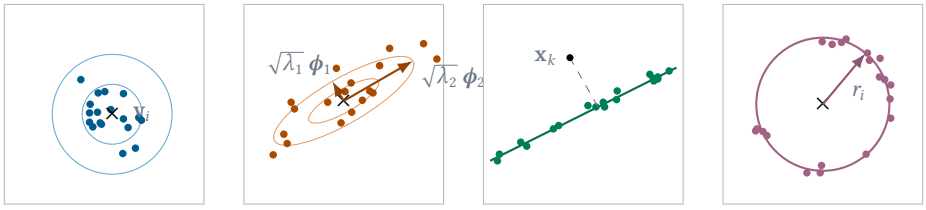


Figure 2.1. The prototypes of this book and the cluster shapes they detect. A point prototype with one fixed norm detects spheres (chapter 3, FCM); a point with its own covariance matrix detects ellipsoids whose axes are the eigenvectors $\boldsymbol{\phi}_j$ of $\mathbf{F}_i$ scaled by $\sqrt{\lambda_j}$ (GK, GG); a linear variety measures the orthogonal distance of a sample to a line or hyperplane and detects flat clusters; a shell prototype measures the distance to a circle of radius r_i and detects hollow clusters (both in chapter 6). Whatever the prototype, the algorithm needs only the distances D_{ik} of every sample to every prototype.

2.2 Theory

This section follows the order of the ingredients: data (section 2.2.1), distances (section 2.2.2), partitions (section 2.2.3), and prototypes (section 2.2.4); then the objective function and the alternating optimization that ties them together (section 2.2.5); the practical matters of initialization, termination, and scaling (sections 2.2.6 to 2.2.8); and finally the notation table (section 2.2.9). The notation is that of Babuška [7], which the MATLAB toolbox [12] also used.

2.2.1 The data

Clustering is applied here to quantitative data. Each observation consists of n measured variables, grouped into a row vector $\mathbf{x}_k = [x_{k1}, x_{k2}, \ldots, x_{kn}] \in \mathbb{R}^n$, and

the N observations form the data matrix

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Nn} \end{bmatrix} \in \mathbb{R}^{N\times n}, \qquad \mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\mathsf{T}. \tag{2.1}$$

In pattern recognition the rows are called patterns or objects, and the columns features or attributes. The meaning of the rows depends on the application (section 1.1): independent observations in data analysis, consecutive samples or delay vectors of signals in time-series analysis, and regression vectors built from lagged inputs and outputs in system identification, where $\mathbf{X}$ spans the product space of regressors and regressands and the clusters are regions in which one local model is valid. In the last two cases, past values of the variables usually enter $\mathbf{X}$ as extra columns; `make_regressors` and `lagged_matrix` of `fuzzyclust.preprocessing` build them. Throughout the book $k = 1, \ldots, N$ indexes samples and $i, j = 1, \ldots, c$ index clusters; in code the data are stored row-wise, exactly as in (2.1).

2.2.2 Distance norms

A cluster is a group of objects that are more similar to one another than to the members of other clusters, and in a metric space similarity is measured by a *distance norm*. Except for Gath–Geva, the algorithms of chapter 3 measure the distance of a sample to a prototype with the squared inner-product norm

$$D^2_{ik\mathbf{A}} = \|\mathbf{x}_k - \mathbf{v}_i\|^2_\mathbf{A} = (\mathbf{x}_k - \mathbf{v}_i)^\mathsf{T}\mathbf{A}\,(\mathbf{x}_k - \mathbf{v}_i), \tag{2.2}$$

where the symmetric positive definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the *norm-inducing matrix*. Its unit sphere $\{\mathbf{x} : \mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} = 1\}$, the boundary of the unit ball, holds the points at unit distance from the origin; centered on a prototype, it gives the shape of the iso-membership contours near that prototype (fig. 2.2). Three global choices are classical:

- $\mathbf{A} = \mathbf{I}$, the Euclidean norm: the unit ball is a sphere, and all features are weighted equally in their given units.
- $\mathbf{A} = \mathbf{A}_D = \operatorname{diag}(1/\sigma_1^2, \ldots, 1/\sigma_n^2)$ with the standard deviations σ_j of the features: the unit ball is an axis-parallel ellipsoid, and every feature is measured in units of its own spread. Clustering with $\mathbf{A}_D$ is identical to Euclidean clustering of the standardized data.
- $\mathbf{A} = \mathbf{F}^{-1}$ with the covariance matrix of the whole data set,

$$\mathbf{F} = \frac{1}{N}\sum_{k=1}^{N}(\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^\mathsf{T}, \qquad \bar{\mathbf{x}} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k, \tag{2.3}$$

  the Mahalanobis norm: the unit ball is an ellipsoid with the orientation of the data cloud, so a correlation that all clusters share is removed.

Any positive definite $\mathbf{A}$ can be factored as $\mathbf{A} = \mathbf{L}^\mathsf{T}\mathbf{L}$, and then $D^2_{ik\mathbf{A}} = \|\mathbf{L}\mathbf{x}_k - \mathbf{L}\mathbf{v}_i\|^2$: clustering with the norm $\mathbf{A}$ is Euclidean clustering of the linearly transformed data $\mathbf{L}\mathbf{x}_k$. A global norm therefore cannot give clusters different shapes; it fixes one shape for all of them. The Gustafson–Kessel algorithm (section 3.2.4) gives every cluster its own matrix $\mathbf{A}_i$, derived from the covariance of its members, and the Gath–Geva algorithm (section 3.2.5) replaces the inner-product norm with an exponential distance. Both are treated in the next chapter; fig. 2.2 already shows what the adaptive norm adds.

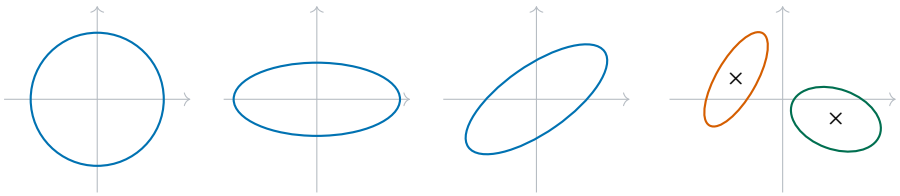


Figure 2.2. Unit spheres $\{\mathbf{x} : \mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} = 1\}$ of the inner-product norms. The Euclidean norm gives spheres, the diagonal norm axis-parallel ellipsoids (equivalent to standardizing the features), and the Mahalanobis norm ellipsoids oriented like the whole data cloud. All three are global, so every cluster has the same shape; the Gustafson–Kessel algorithm (right) gives each cluster its own norm-inducing matrix and thus its own ellipsoid.

2.2.3 Partition spaces

The clusters are subsets of the data, and the partition says which sample belongs to which subset. A *hard partition* of $\mathbf{X}$ into c clusters is a family of subsets $A_1, \ldots, A_c$ with

$$\bigcup_{i=1}^{c} A_i = \mathbf{X}, \qquad A_i \cap A_j = \emptyset \ (i \neq j), \qquad \emptyset \subset A_i \subset \mathbf{X}, \tag{2.4}$$

that is, the subsets cover the data, they are disjoint, and none is empty or equal to the whole data set. In terms of the characteristic functions $\mu_i(\mathbf{x}_k) = \mu_{ik} \in \{0, 1\}$ of the subsets, these conditions read

$$\mu_{ik} \in \{0, 1\}, \qquad \sum_{i=1}^{c} \mu_{ik} = 1 \ \ \forall k, \qquad 0 < \sum_{k=1}^{N} \mu_{ik} < N \ \ \forall i, \tag{2.5}$$

and the memberships are collected in the partition matrix $\mathbf{U} = [\mu_{ik}] \in \mathbb{R}^{c\times N}$, whose i-th row is the membership function of cluster i and whose k-th column holds the memberships of sample k.

Definition 2.1 (Hard partitioning space). Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a finite set and $2 \leq c < N$ an integer. The hard partitioning space of $\mathbf{X}$ is

$$M_{hc} = \left\{ \mathbf{U} \in \mathbb{R}^{c\times N} \;\middle|\; \mu_{ik} \in \{0, 1\} \ \forall i, k; \ \sum_{i=1}^{c} \mu_{ik} = 1 \ \forall k; \ 0 < \sum_{k=1}^{N} \mu_{ik} < N \ \forall i \right\}. \tag{2.6}$$

A *fuzzy partition* [1] relaxes the first condition and lets the memberships take any value in $[0, 1]$. The sum of each column is still one, so the total membership of each sample is distributed over the clusters; how it is distributed is not constrained.

Definition 2.2 (Fuzzy partitioning space)**.** With $\mathbf{X}$ and c as above, the fuzzy partitioning space is

$$M_{fc} = \left\{ \mathbf{U} \in \mathbb{R}^{c \times N} \;\middle|\; \mu_{ik} \in [0,1]\ \forall i,k;\ \sum_{i=1}^{c} \mu_{ik} = 1\ \forall k;\ 0 < \sum_{k=1}^{N} \mu_{ik} < N\ \forall i \right\}. \quad (2.7)$$

The column constraint $\sum_i \mu_{ik} = 1$ makes the memberships *relative*: a sample far from all prototypes still has memberships that sum to one, and a sample equidistant from all prototypes has $\mu_{ik} = 1/c$ in every cluster, however far away it is (fig. 1.1, bottom). This is sometimes called the probabilistic constraint. The *possibilistic partition* [14] drops it and only requires every sample to belong to at least one cluster to some degree:

Definition 2.3 (Possibilistic partitioning space)**.**

$$M_{pc} = \left\{ \mathbf{U} \in \mathbb{R}^{c \times N} \;\middle|\; \mu_{ik} \in [0,1]\ \forall i,k;\ \forall k\ \exists\, i\ :\ \mu_{ik} > 0;\ 0 < \sum_{k=1}^{N} \mu_{ik} < N\ \forall i \right\}. \quad (2.8)$$

Possibilistic memberships are *absolute* degrees of typicality: a sample far from everything has small memberships in all clusters, so an outlier is recognized as such. The price is that the clusters are no longer coupled through the column sum, and the clustering problem splits into c independent problems that may all converge to the same cluster unless additional terms hold them apart; this is the subject of chapter 6. Figure 2.3 shows a partition matrix in each of the three spaces. The nesting $M_{hc} \subset M_{fc} \subset M_{pc}$ is immediate, and so is the fact that M_{fc} is a convex set, whereas M_{hc} is a finite one; the former is why fuzzy clustering can be treated with calculus, and the latter why hard clustering is a combinatorial problem.

2.2.4 Prototypes

The prototype of cluster i is the object the distances are measured to. In the simplest case it is a point $\mathbf{v}_i \in \mathbb{R}^n$ of the data space, the *cluster center*, and the matrix $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_c]^\mathsf{T} \in \mathbb{R}^{c \times n}$ collects the centers. The algorithms of chapter 3 all have point prototypes. The Gustafson–Kessel and Gath–Geva algorithms add to each center a covariance matrix $\mathbf{F}_i$ that shapes the norm around it, so their prototype is in effect the ellipsoid $(\mathbf{v}_i, \mathbf{F}_i)$; Gath–Geva also adds a prior probability α_i that scales the size of the cluster. A *medoid* (section 3.2.2) is a point prototype constrained to be one of the samples, which is useful when the mean of a group has no meaning. Prototypes of higher order are linear varieties (points, lines, and planes) and shells (circles and ellipses), treated in chapter 6. In the identification setting of chapter 8 the prototype of a cluster is a local linear model, whose

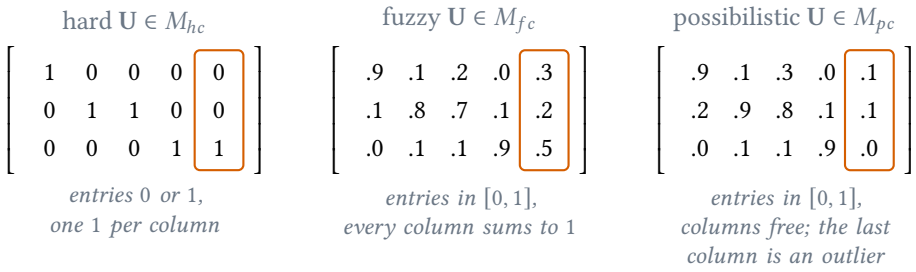


Figure 2.3. One partition matrix of three clusters and five samples in each of the three partitioning spaces. In the hard space every column is a unit vector; in the fuzzy space the columns are points of the probability simplex, so memberships are relative; in the possibilistic space the columns are free, so memberships are absolute degrees of typicality and an outlier (highlighted column) has small memberships everywhere.

parameters are read off the eigenstructure of $\mathbf{F}_i$ or fitted by weighted least squares. Whatever the prototype, the clustering algorithm needs only the distances D_{ik} of every sample to every prototype, and this is the interface that the base class of the toolbox asks an algorithm to provide.

2.2.5 Objective functions and alternating optimization

Most of the algorithms in this book minimize a sum of weighted squared distances, the *c-means functional* of Dunn [2] and Bezdek [3],

$$J(\mathbf{X}; \mathbf{U}, \mathbf{V}) = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^{m} D_{ik}^{2}, \qquad \mathbf{U} \in M_{fc},\ m > 1, \tag{2.9}$$

where D_{ik}^2 is the squared distance of $\mathbf{x}_k$ to the prototype of cluster i in the norm of the algorithm and m is the *fuzziness* or weighting exponent. Statistically, J measures the total variance of the samples about the prototypes, with each sample counted in cluster i with weight μ_{ik}^m. For $m \to 1$ and $\mathbf{U} \in M_{hc}$ it is the within-cluster sum of squares of k-means. Most algorithms of chapter 3 and of the later chapters are defined by (2.9) together with a choice of distance and prototype; the Gath–Geva algorithm, the possibilistic variants, and the fuzzy self-organizing map modify the functional but keep its structure.

Minimizing J over $\mathbf{U}$ and $\mathbf{V}$ jointly is a nonconvex problem. It is, however, convex in $\mathbf{U}$ for fixed prototypes and easy in $\mathbf{V}$ for fixed $\mathbf{U}$. With the column constraint adjoined by Lagrange multipliers, setting the gradient with respect to $\mathbf{U}$ to zero yields a closed-form expression for the memberships in terms of the distances (derived in section 3.2.3), and setting the gradient with respect to the

prototypes to zero yields the prototypes as membership-weighted means (or an analogous update for ellipsoids, lines, and shells). The classical way to exploit this is *alternating optimization*, also called Picard iteration or grouped coordinate descent: fix $\mathbf{U}$ and update the prototypes, fix the prototypes and update $\mathbf{U}$, and repeat. Neither half-step can increase J, so the sequence $J^{(1)} \geq J^{(2)} \geq \dots$ is monotone and bounded below. Bezdek's original theorem [10] claimed convergence to a local minimum; the corrected theorem of Bezdek, Hathaway, Sabin, and Tucker [11] states that every convergent subsequence of the iterates tends to a local minimum or a saddle point of J, and saddle points are rarely met in practice. Figure 2.4 shows the loop as it is implemented once in `BaseFuzzyClusterer`. An algorithm provides the prototype update and the distance computation; the base class provides everything else.

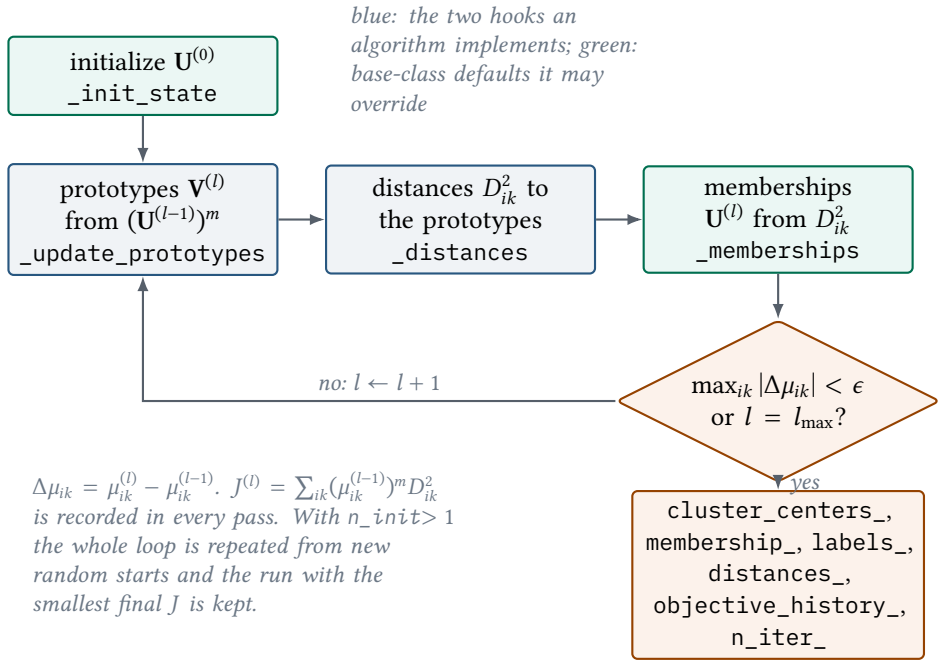


Figure 2.4. Alternating optimization as implemented once in `BaseFuzzyClusterer`. An algorithm defines only how prototypes are computed from the weighted partition and how distances to its prototypes are measured (blue blocks); the base class provides the initialization and the membership update (green blocks, which an algorithm may override), the termination test, the objective history, and the restarts.

2.2.6 Initialization and local optima

The alternating optimization is a local method, so the initial partition decides which stationary point is reached. The toolbox offers three ways to start. The default, `init="random"`, reproduces the MATLAB toolbox: c prototypes are drawn uniformly from a box that is centered on the data mean and encloses all samples,

Algorithm 2.1 Alternating optimization of a prototype-based clustering functional

Input: data $\mathbf{X}$, number of clusters $1 < c < N$, fuzziness exponent m, tolerance $\epsilon > 0$, maximum number of iterations $l_{\max}$, number of restarts r

for *run* $= 1, \ldots, r$ **do**
  initialize $\mathbf{U}^{(0)} \in M_{fc}$ (random prototypes, k-means++ seeds, or a user-supplied matrix);
  for $l = 1, 2, \ldots, l_{\max}$ **do**
    prototypes: compute $\mathbf{V}^{(l)}$ (and any shape parameters $\mathbf{F}_i, \mathbf{A}_i, \alpha_i$) from $\mathbf{X}$ and $(\mathbf{U}^{(l-1)})^m$;
    distances: compute D_{ik}^2 of every sample to every prototype;
    objective: $J^{(l)} \leftarrow \sum_{i,k} (\mu_{ik}^{(l-1)})^m D_{ik}^2$;
    memberships: compute $\mathbf{U}^{(l)}$ from the distances (fuzzy or crisp rule);
    if $\max_{i,k} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$ **then break**;
  keep the run if $J^{(l)}$ is the smallest so far;

Output: $\mathbf{V}$, $\mathbf{U}$, hard labels $\arg\max_i \mu_{ik}$, distances D_{ik}, history $J^{(1)}, \ldots, J^{(l)}$

In a nutshell The common template

- Every algorithm minimizes a sum of membership-weighted squared distances $J = \sum_{ik} \mu_{ik}^m D_{ik}^2$ over a partition $\mathbf{U}$ and prototypes $\mathbf{V}$.
- Two choices define an algorithm: the prototype (point, ellipsoid, medoid, line, shell) and the distance (fixed norm, adaptive norm, exponential).
- Alternating optimization updates prototypes and memberships in turn; J decreases monotonically, and the iteration converges to a local minimum (in rare cases to a saddle point).
- Which optimum is reached depends on the start: restarts (`n_init`) and good seeds are the remedy; a converged run with a poor J is a poor optimum, not an unfinished one.
- Parameters shared by the estimators: `n_clusters`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, and, in the fuzzy ones, `m`.

$\mathbf{v}_i^{(0)} = \bar{\mathbf{x}} + (2\mathbf{u}_i - \mathbf{1}) \odot \mathbf{h}$ with $\mathbf{u}_i \sim \mathcal{U}[0,1]^n$ and $h_j = \max_k |x_{kj} - \bar{x}_j|$, and the initial memberships follow from the Euclidean distances to these prototypes through the fuzzy membership formula. Starting from random prototypes rather than from a random membership matrix avoids the near-uniform partitions that a random $\mathbf{U}^{(0)}$ produces for large N. The second option, `init="kmeans++"`, is the seeding of Arthur and Vassilvitskii [15]: the first seed is a random sample, and every further seed is a sample drawn with probability proportional to its squared distance from the seeds chosen so far, so that the seeds spread over the data. The toolbox calls the scikit-learn implementation, a greedy variant that draws $2 + \lfloor \ln c \rfloor$ candidates in this way for every further seed and keeps the one that most reduces the sum of squared distances to the nearest seed. The seeding is designed for compact groups; on elongated or crossing clusters its preference for far-apart seeds can be a disadvantage (section 2.4.3). The third option is an explicit array, either initial centers of shape (c, n) or an initial partition of shape (N, c). This is how the Gath–Geva algorithm is started from a converged FCM or GK result, and how a clustering can be continued or reproduced.

Restarts complement the seeding. With `n_init=` $r > 1$ the loop of algorithm 2.1 is run r times from fresh starts, and the run with the smallest final J is kept; the final objective is the only criterion, because it is the only one the algorithm itself can evaluate. On well-separated compact clusters nearly every start reaches the same optimum, and $r = 1$ suffices. On data with elongated, crossing, or differently sized clusters, the spread of the final objectives over seeds (fig. 2.7) is the quickest diagnostic of a multimodal objective, and r between 5 and 20 is cheap insurance. When `init` is an array, `n_init` is ignored.

2.2.7 Termination

The iteration stops when the partition no longer changes,

$$\max_{i,k} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon, \tag{2.10}$$

with the tolerance `tol=` ϵ (default 10^{-4}, the value in the code of the MATLAB toolbox, although its manual states 10^{-3}), or after `max_iter` iterations (default 300). The maximum change of a membership is a scale-free criterion: it depends neither on the units of the data nor on the size of J. It is the termination test of Bezdek's FCM algorithm [3, 16]. For hard partitions it reduces to "no sample changed cluster." The number of iterations actually run is stored in `n_iter_`, and the objective after every iteration in `objective_history_`; a history that is still falling steeply at `max_iter` indicates that the limit, not convergence, stopped the run. A smaller ϵ locates the prototypes more precisely but costs iterations; because the memberships change slowly when m is large (section 3.2.3), partitions with m above about two need more iterations for the same ϵ. Two numerical safeguards protect the membership update: squared distances are clipped at zero, and a constant 10^{-10} is added to them before negative powers are taken (as in the

MATLAB toolbox), so that a sample that coincides with a prototype receives a membership of almost one instead of causing a division by zero.

2.2.8 Scaling the data

Distances compare features, and features have units. If one feature ranges over hundreds and another over fractions, the Euclidean distance is in effect the first feature alone, and so is the clustering (section 2.4.1). Scaling the data is therefore part of the clustering method, not a preliminary detail. Two linear scalings are standard; `fuzzyclust.preprocessing.normalize` reproduces the MATLAB function `clust_normalize`, and the scaler it returns reproduces `clust_denormalize`:

$$\text{range:}\quad \tilde{x}_{kj} = \frac{x_{kj} - \min_k x_{kj}}{\max_k x_{kj} - \min_k x_{kj}} \in [0,1], \qquad \text{variance:}\quad \tilde{x}_{kj} = \frac{x_{kj} - \bar{x}_j}{\sigma_j}. \tag{2.11}$$

Range scaling is bounded and easy to interpret but sensitive to outliers; variance scaling (standardization) depends less on single extreme values but is still affected by heavy tails. Both correspond to a diagonal norm in the original units ($\mathbf{A}_D$ for variance scaling), and both must be inverted to report prototypes in physical units, which the returned scaler does with `inverse_transform`. The recommended pattern is a scikit-learn `Pipeline`: the scaler is fitted on the training data and applied automatically to every new sample passed to `predict_proba`.

For the adaptive norm of Gustafson–Kessel the situation is subtler. In exact arithmetic the GK distance $D^2_{ik\mathbf{A}_i}$ with $\mathbf{A}_i = [\rho_i \det \mathbf{F}_i]^{1/n}\mathbf{F}_i^{-1}$ is invariant up to a common factor under any invertible linear map $\mathbf{x}_k \mapsto \mathbf{S}\mathbf{x}_k$: the covariance becomes $\mathbf{S}\mathbf{F}_i\mathbf{S}^\mathsf{T}$, its determinant is multiplied by $(\det \mathbf{S})^2$, and all squared distances are multiplied by the same constant $(\det \mathbf{S})^{2/n}$, which the membership formula cancels. GK is therefore affine-invariant in principle, and so is Gath–Geva. In practice the invariance is only partial. The random initialization in an axis-parallel box is not invariant. The regularization of the covariance matrices (section 3.2.4) shrinks every $\mathbf{F}_i$ toward a multiple of the *identity* and bounds its condition number, and neither operation is invariant under anisotropic scaling, so with $\gamma > 0$, or when the bound β is active, the result depends on the units. Finally, the validity indices of chapter 4 are computed with Euclidean distances. Scaling the data is therefore good practice for every algorithm of the toolbox, and indispensable for k-means and for FCM with the Euclidean norm.

2.2.9 Notation

Table 2.1 collects the symbols of the book together with their names in code. The convention that index i (and j) runs over clusters and k over samples is used without exception; the partition matrix is $c \times N$ in formulas, as in Bezdek and Babuška, and is stored transposed in code so that it aligns with the (N, c) layout of `predict_proba`.

Table 2.1. Notation of the book and the corresponding names in `fuzzyclust`.

Symbol	Meaning	Code
N	number of samples	`n_samples`
n	number of features	`n_features`
c	number of clusters	`n_clusters`
$\mathbf{X} \in \mathbb{R}^{N \times n}$	data matrix $[\mathbf{x}_1, \dots, \mathbf{x}_N]^\mathsf{T}$, rows are samples	`X`
$\mathbf{x}_k$, $k = 1, \dots, N$	k-th sample (row)	`X[k]`
$\mathbf{v}_i$, $i = 1, \dots, c$	prototype (center) of cluster i	`cluster_centers_[i]`
$\mathbf{V} \in \mathbb{R}^{c \times n}$	matrix of prototypes $[\mathbf{v}_1, \dots, \mathbf{v}_c]^\mathsf{T}$	`cluster_centers_`
$\mu_{ik} \in [0, 1]$	membership of sample k in cluster i	`membership_[k, i]`
$\mathbf{U} = [\mu_{ik}] \in \mathbb{R}^{c \times N}$	fuzzy partition matrix	`membership_` is $\mathbf{U}^\mathsf{T}$, shape `(N, c)`
$m > 1$	fuzziness exponent	`m`
$D^2_{ik\mathbf{A}_i}$, eq. (2.2)	squared inner-product distance of $\mathbf{x}_k$ to $\mathbf{v}_i$	`transform(X)` returns D_{ik}, shape `(N, c)`
$\mathbf{A}_i$	norm-inducing matrix of cluster i	`norm_matrices_[i]` (`norm_matrix_` if shared)
$\mathbf{F}_i$	fuzzy covariance matrix $\sum_k \mu_{ik}^m (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^\mathsf{T} / \sum_k \mu_{ik}^m$	`covariances_[i]`
$\alpha_i = \frac{1}{N} \sum_k \mu_{ik}$	prior probability of cluster i (GG)	`priors_[i]`
$J(\mathbf{X}; \mathbf{U}, \mathbf{V})$	objective, $\sum_i \sum_k \mu_{ik}^m D_{ik}^2$	`objective_`, `objective_history_`
ϵ	termination tolerance on $\max_{ik} \left\lvert \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right\rvert$	`tol`
l, $l_{\max}$	iteration counter, maximum	`n_iter_`, `max_iter`
ρ_i	cluster volume (GK)	`rho`
γ	covariance regularization weight (GK, GG)	`gamma`
β	condition-number bound (GK, GG)	`beta`
$\mathbf{y}_k \in \mathbb{R}^q$	low-dimensional projection of $\mathbf{x}_k$	`embedding_`
$\mathbf{z}_i$	projected cluster center	`embedded_centers_`
M_{hc}, M_{fc}, M_{pc}	hard, fuzzy, possibilistic partitioning spaces	–

2.3 Using it in fuzzyclust

The clustering estimators derive from the base class `BaseFuzzyClusterer` of `fuzzyclust.base`, which mixes in the scikit-learn classes `BaseEstimator`, `ClusterMixin`, and `TransformerMixin` and implements the alternating optimization of algorithm 2.1. The contract below therefore holds for FCM, GK, GG, the hard methods, and most estimators of the later chapters.

`fuzzyclust.base.BaseFuzzyClusterer` – constructor parameters shared by every estimator

`n_clusters=3` (c); `m=2.0` (m; not a parameter of the hard methods, which fix $m = 1$); `tol=1e-4` (ϵ); `max_iter=300` ($l_{\max}$); `init="random"` (`"random"`: random prototypes in a box around the data, as in MATLAB; `"kmeans++"`; or an array of initial centers (c, n) or an initial partition (N, c)); `n_init=1` (restarts; the run with the lowest final J is kept; ignored for an array `init`); `random_state=None`; `verbose=0` (a positive value prints J at every iteration). Algorithm-specific parameters, e.g. `norm` for FCM or `rho`, `gamma`, and `beta` for GK, follow `n_clusters` (and `m`, where present) in the signatures, so pass them by keyword. Nothing is validated or computed in the constructor (a scikit-learn rule); all work happens in `fit`.

fitted attributes (trailing underscore)

`cluster_centers_` (c, n): $\mathbf{V}$. `membership_` (N, c): $\mathbf{U}^{\mathsf{T}}$, rows sum to one. `labels_` $(N,)$: $\arg\max_i \mu_{ik}$. `distances_` (N, c): D_{ik} of the training data at convergence, in the norm of the algorithm. `objective_`: final J; `objective_history_` $(l,)$: J per iteration; `n_iter_`. Where defined: `covariances_` (c, n, n) for $\mathbf{F}_i$, `norm_matrices_` (c, n, n) for $\mathbf{A}_i$, `norm_matrix_` (n, n) for the shared $\mathbf{A}$ of FCM, `priors_` $(c,)$ for α_i, `eigenvalues_`, and `eigenvectors_`. Set by scikit-learn's validation: `n_features_in_`, and `feature_names_in_` when a `DataFrame` was fitted.

methods

`fit(X, y=None)` runs the alternating optimization of algorithm 2.1 and returns `self`. `predict_proba(X)` returns the memberships (N', c) of new samples, computed from their distances to the fitted prototypes with the algorithm's own membership rule (the MATLAB `clusteval`); `predict(X)` returns their arg-max; `transform(X)` returns the distances D_{ik} (N', c), so a clusterer is a transformer that maps n features to c distance features (`get_feature_names_out` gives `dist_cluster_0`, …, and `set_output(transform="pandas")` is honored). `fit_predict` and `fit_transform` are available, and `score(X)` returns $-J(X)$ (higher is better, so

grid search maximizes it). Inputs may be `numpy` arrays, `pandas` data frames, or anything scikit-learn accepts; they are never modified and never scaled inside the estimator.

implementing a new algorithm

A subclass defines `_update_prototypes(X, Um)`, which sets `cluster_centers_` (and shape attributes) from $\mathbf{X}$ and the matrix $\mathbf{U}^m$ of shape (N, c), and `_distances(X)`, which returns the squared distances (N, c) to the current prototypes. Optionally it overrides `_memberships(d2)` (the hard methods return one-hot rows, Gath–Geva a log-domain softmax), `_check_params(X)` for parameter validation, `_init_state(X, rng)` for a special initialization, and `_finalize(X, Um)` for attributes computed once after convergence. Everything else (restarts, history, the scikit-learn API, and pandas support) is inherited.

`fuzzyclust.preprocessing.normalize(X, method="var")`

Returns `(X_scaled, scaler)`: the data standardized to zero mean and unit variance (`"var"`) or scaled to the range $[0, 1]$ (`"range"`), and the fitted `StandardScaler` or `MinMaxScaler`, whose `inverse_transform` is the MATLAB `clust_denormalize`. A `DataFrame` comes back as a `DataFrame`. For new data, put the scaler into a `Pipeline`.

The three built-in norms of FCM are selected by name, and an explicit matrix is accepted as well:

```
norms = {
    "euclidean": "euclidean",
    "diagonal": "diagonal",
    "mahalanobis": "mahalanobis",
    r"explicit $\mathbf{A}=\mathbf{\Sigma}^{-1}$": np.linalg.inv(S @ S.T),
}
fig, axes = plt.subplots(1, 4, figsize=(19, 4.5), sharex=True, sharey=True)
for ax, (name, norm) in zip(axes, norms.items(), strict=True):
    fcm = FCM(n_clusters=3, norm=norm, random_state=0, n_init=5).fit(X)
    plot_membership_contours(fcm, X, ax=ax, labels=False)
    plot_covariance_ellipses(fcm.cluster_centers_, fcm.covariances_, ax=ax)
    ax.set_title(f"{name}: ARI = {adjusted_rand_score(y_true,
    fcm.labels_):.2f}")
```

Restarts take one keyword, and the spread of the final objective over seeds diagnoses local optima:

```
seeds = range(20)
runs = [GK(n_clusters=3, random_state=s).fit(X) for s in seeds]
J = np.array([r.objective_ for r in runs])
ari = np.array([adjusted_rand_score(y_true, r.labels_) for r in runs])
```

Scaling composes with any estimator through a pipeline, which then scales new data automatically:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pipe = make_pipeline(StandardScaler(), GK(n_clusters=4,
    random_state=0)).fit(X)
pipe.predict_proba(new_points)          # scaled, then evaluated with the
    GK norms
pipe[-1].cluster_centers_               # prototypes in the scaled space
pipe[0].inverse_transform(pipe[-1].cluster_centers_)   # ... and in
    physical units
```

2.4 Examples

2.4.1 Distance norms

The script `plot_11_distance_norms.py` generates three clusters with the same elongated, rotated covariance and clusters them with the three built-in norms and with the explicit inverse of the generating covariance (fig. 2.5). The Euclidean norm cuts the elongated groups with circles and mixes neighboring clusters (adjusted Rand index, ARI, 0.47); the diagonal norm helps only insofar as the variances differ along the axes (0.61). The Mahalanobis norm, whose unit ball has the orientation of the pooled data, does better (0.78) but not perfectly, because the covariance of the pooled data also contains the spread of the cluster centers and is rounder than the clusters themselves (its eigenvalue ratio is 5.4, against 39.5 for the true within-cluster covariance). The explicit within-cluster norm recovers the groups exactly. The second figure of the script (not shown) stretches one feature by a factor of twenty. The Euclidean norm then produces horizontal bands, whereas the diagonal norm and standardization followed by the Euclidean norm give identical partitions, the same as without the stretch, because the two are the same operation.

2.4.2 Scaling: the motorcycle data

The motorcycle data set (time after impact in milliseconds against head acceleration in g) is the real data set that the MATLAB toolbox used throughout. The two variables have ranges of about 55 and 210, so the raw Euclidean distance is dominated by the acceleration, and a Euclidean method such as k-means or FCM sees mainly that variable. Figure 2.6 (script `plot_07_motorcycle_scaling.py`)

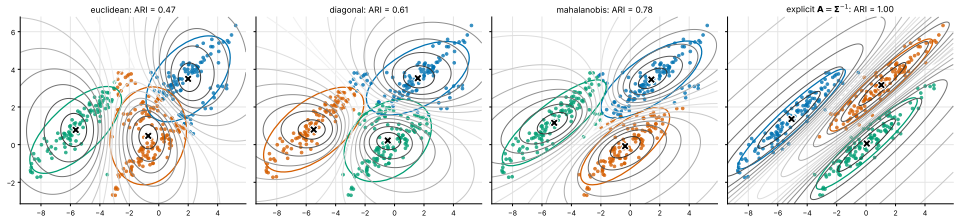


Figure 2.5. Fuzzy c-means with four norms on three clusters of identical elongated shape. The level curves of the maximal membership follow the shape of the unit ball of the norm; the colored ellipses are the 2σ ellipses of the fuzzy covariance matrices of the clusters found. From left to right the norm fits the clusters better, and the ARI with the generating labels rises from 0.47 to 1.00.

clusters the raw, the range-scaled, and the variance-scaled data with Gustafson–Kessel and maps the prototypes back to physical units. All three partitions are identical, with the same prototypes in physical units: the clusters follow the course of the signal in time, which is the physically meaningful segmentation. This is the affine invariance of GK (section 2.2.8) at work: on these data the units do not change the optimum that the iteration reaches. With regularization switched on, or on data where the box initialization leads to different local optima, the scaled and raw results can differ, so scaling remains the safe default.

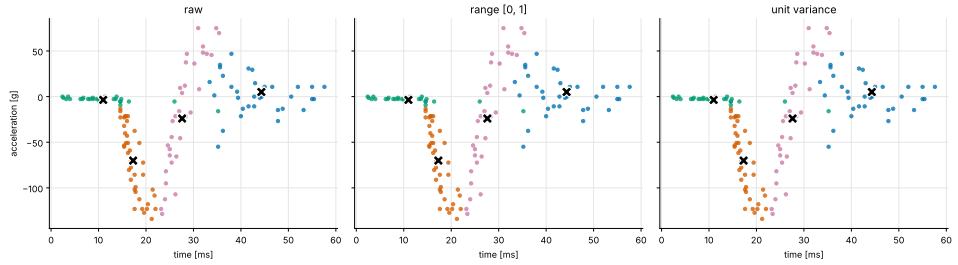


Figure 2.6. Gustafson–Kessel clustering of the motorcycle data without scaling (left), after range scaling (middle), and after variance scaling (right), with the prototypes shown in the original units. The three partitions coincide: the adaptive norm of GK makes the result independent of the units of the features, and every cluster is a segment of the signal in time.

2.4.3 Initialization and restarts

The script `plot_10_initialisation_restarts.py` fits GK with three clusters to three noisy line segments from twenty random starts. The final objectives take two distinct values, $J = 142.1$ in six runs and $J = 229.4$ in fourteen, with adjusted Rand indices of 0.94 and 0.47; the objective histories of all runs decrease monotonically, so the poor runs are converged runs. Figure 2.7 compares four strategies over twenty outer seeds: a single random start reaches the good optimum in 30 % of the cases, five restarts in 70 %, and ten restarts in 80 %. Seeding with

k-means++, which places the seeds far apart, reaches it in none: on line-shaped clusters the far-apart seeds are the *ends* of different lines, the configuration from which GK slides into the poor optimum. Seeding heuristics are tuned to compact groups and are no substitute for restarts.

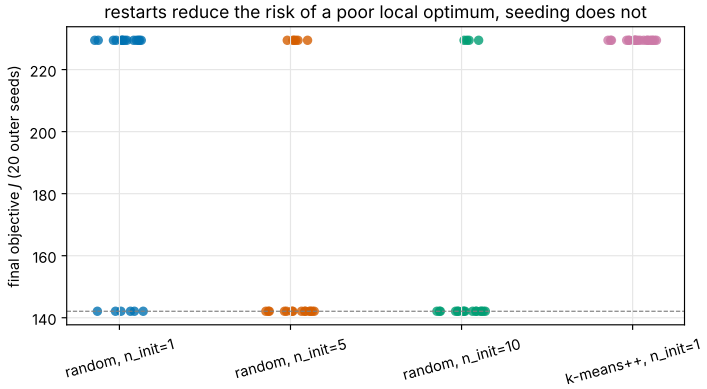


Figure 2.7. Final objective of Gustafson–Kessel on three line segments for twenty outer seeds and four initialization strategies. Restarts (`n_init`) move most runs to the good optimum (dashed line); k-means++ seeding, designed for compact groups, fails on these data in every run.

2.4.4 The scikit-learn interface

The script `plot_08_sklearn_pandas_workflow.py` exercises the interface on the iris data, given as a `pandas` data frame. A pipeline of a `StandardScaler` and GK with `set_output(transform="pandas")` returns the distance matrix as a data frame with columns `dist_cluster_0`, …, and keeps the feature names in `feature_names_in_`. Aggregated per species, the memberships recover setosa with a mean membership of 0.95, whereas versicolor and virginica overlap (0.72 and 0.70). `GridSearchCV` selects c and m of FCM with the Xie–Beni index on held-out folds as the scoring function (fig. 2.8), and the fitted pipeline survives a pickle round trip. The grid search prefers two clusters, a well-known property of the iris data: validity indices measure geometric separation, not agreement with external labels, and two of the three species are not geometrically separated.

2.5 Further reading

The data–clusters–partition exposition follows Babuška [7] and the MATLAB toolbox manual [12]. The partitioning spaces are Bezdek's [3]; Ruspini [1] introduced fuzzy partitions, and Krishnapuram and Keller [14] the possibilistic ones. The c-means functional goes back to Dunn [2], its alternating optimization and the convergence theorem to Bezdek [10, 16], with the corrected theorem by Bezdek,

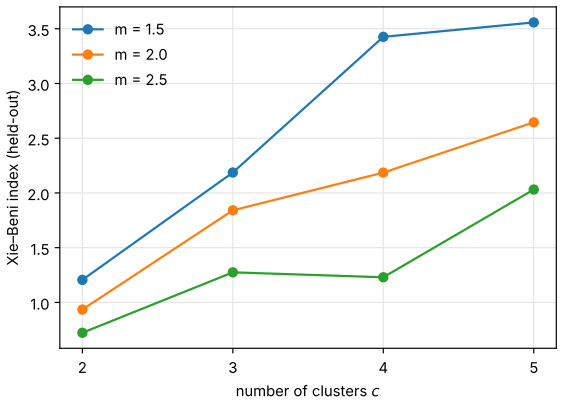


Figure 2.8. Xie–Beni index of FCM on held-out folds of the standardized iris data for several c and m, computed inside `GridSearchCV` with `predict_proba`. The index is lowest for two clusters because versicolor and virginica overlap.

Hathaway, Sabin, and Tucker [11]; Höppner et al. [6] give a unified treatment of the objective-function family, including linear varieties and shells. The k-means++ seeding is due to Arthur and Vassilvitskii [15]. The estimator interface follows scikit-learn [13].

CHAPTER

3



Hard and fuzzy partitioning algorithms

The previous chapter set up the template: a prototype, a distance, the c-means functional, and its alternating optimization. This chapter fills it with the core algorithms of the toolbox, all of which the MATLAB predecessor already offered: the hard reference methods k-means and k-medoids; fuzzy c-means (FCM), derived in full together with the role of the fuzziness exponent; the Gustafson–Kessel algorithm (GK), which gives every cluster its own norm and is presented with the eigenstructure of its covariance matrices and the regularization of Babuška, van der Veen, and Kaymak; and the Gath–Geva algorithm (GG), which replaces the norm by an exponential distance that adapts volume and density and must be started from a good partition. A comparison of the four is followed by two extensions: the mechanisms that make a partition fuzzy (the exponent m, entropy regularization and its interpretation as deterministic annealing, quadratic regularization, and the polynomial fuzzifier of Klawonn and Höppner, which allows exact zero memberships), and the density-based estimation of the number and location of clusters by the mountain method and subtractive clustering, which also initializes the iterative algorithms. Everything that follows builds on this chapter: validity indices judge the partitions these algorithms produce, and the identification, time-series, and graph methods of later chapters use GK and GG as building blocks.

3.1 Problem and idea

Every algorithm of this chapter minimizes the c-means functional $J = \sum_i \sum_k \mu_{ik}^m D_{ik}^2$ of section 2.2.5 by alternating optimization, and all of them have point prototypes. They differ in the distance and in the partition space. Figure 3.1 shows what this means for one set of three prototypes: the level curves of the maximal membership

$\max_i \mu_i(\mathbf{x})$ over the plane, the picture that the MATLAB function `clusteval` drew and `plot_membership_contours` draws, change shape from algorithm to algorithm. k-means tiles the plane into Voronoi cells with crisp boundaries. FCM, with a fixed Euclidean norm, draws nearly circular contours around every prototype; it can only see spherical clusters of about the same size. GK derives a norm-inducing matrix from the fuzzy covariance of each cluster, so its contours are ellipses with the orientation and elongation of the cluster, but the determinant constraint fixes the volume of every cluster. GG measures distance by the reciprocal of a Gaussian density weighted by a prior probability; its contours are ellipses of different size and density, and the memberships fall off exponentially, so the partition is almost crisp between the clusters. Each step up this ladder buys flexibility at a price: more parameters per cluster, more computation, and greater sensitivity to the initial partition, which is why GG is usually started from FCM or GK.

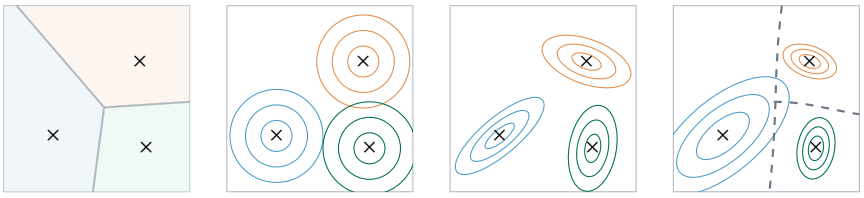


Figure 3.1. Level curves of the maximal membership $\max_i \mu_i(\mathbf{x})$ for one set of prototypes under four algorithms. They are the Voronoi cells of k-means (crisp), circles for fuzzy c-means (one global norm), ellipses of equal volume but individual orientation for Gustafson–Kessel (adaptive norm with determinant constraint), and ellipses of individual size and density with nearly crisp borders for Gath–Geva (exponential distance with priors).

3.2 Theory

We treat the algorithms in the order of fig. 3.1: the hard methods (sections 3.2.1 and 3.2.2), fuzzy c-means with its derivation and the role of m (section 3.2.3), Gustafson–Kessel (section 3.2.4), and Gath–Geva (section 3.2.5);section 3.2.6 compares them. Section 3.2.7 then examines the mechanism that makes these partitions fuzzy and its alternatives, and section 3.2.8 the estimation of the number and location of clusters before any iteration. The notation is that of table 2.1; in formulas $\mathbf{U}$ is $c \times N$, in code it is stored transposed.

Algorithm 3.1 Hard c-means (k-means)

Input: data $\mathbf{X}$, number of clusters $1 < c < N$, tolerance ϵ

initialize the prototypes $\mathbf{v}_i^{(0)}$ (random, k-means++, or given); $l \leftarrow 0$;

repeat

  $l \leftarrow l + 1$;

  distances: $D_{ik}^2 = \left\| \mathbf{x}_k - \mathbf{v}_i^{(l-1)} \right\|^2$ for all i, k;

  partition: $\mu_{ik}^{(l)} = 1$ if $i = \arg\min_j D_{jk}^2$, else 0;

  prototypes: $\mathbf{v}_i^{(l)} = \sum_k \mu_{ik}^{(l)} \mathbf{x}_k / \sum_k \mu_{ik}^{(l)}$ (previous prototype if the cluster is empty);

until *no sample changed cluster, i.e.* $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: $\mathbf{V}$, crisp $\mathbf{U} \in M_{hc}$, within-cluster sum of squares J

3.2.1 Hard c-means (k-means)

The hard c-means or k-means algorithm [17] partitions the data into c disjoint subsets $A_1, \ldots, A_c$ and minimizes the within-cluster sum of squares,

$$J(\mathbf{X}; \mathbf{U}, \mathbf{V}) = \sum_{i=1}^{c} \sum_{\mathbf{x}_k \in A_i} \|\mathbf{x}_k - \mathbf{v}_i\|^2 = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik} \|\mathbf{x}_k - \mathbf{v}_i\|^2, \qquad \mathbf{U} \in M_{hc}, \tag{3.1}$$

which is the c-means functional (2.9) for $m = 1$ and a hard partition. For fixed subsets the minimizing prototype is the mean of the members,

$$\mathbf{v}_i = \frac{1}{N_i} \sum_{\mathbf{x}_k \in A_i} \mathbf{x}_k, \qquad N_i = |A_i| = \sum_{k=1}^{N} \mu_{ik}, \tag{3.2}$$

and for fixed prototypes the minimizing partition assigns every sample to its nearest prototype, $\mu_{ik} = 1$ if $i = \arg\min_j \|\mathbf{x}_k - \mathbf{v}_j\|^2$ and 0 otherwise. Alternating the two steps decreases J and terminates after finitely many iterations, because there are finitely many partitions and none is visited twice. The result depends more strongly on the start than that of FCM, because a prototype that loses all its members cannot recover. `fuzzyclust` keeps the previous prototype of an emptied cluster instead of letting it collapse, and restarts (`n_init`) are the standard remedy. The MATLAB toolbox manual illustrates the problem with two runs on the same data, one of which splits a cluster and merges two others.

3.2.2 k-medoids

The mean of a cluster need not be a meaningful object: if the rows of $\mathbf{X}$ are discrete states of a system, categorical codes mapped to numbers, or positions on a network, the average of two states is no state. The k-medoids algorithm [5]

In a nutshell k-means

- Minimizes the within-cluster sum of squares; prototypes are cluster means, the partition is crisp (Voronoi cells).
- Assumes compact, roughly spherical clusters of similar size in the Euclidean norm; needs scaled data.
- Fast ($\mathcal{O}(Ncn)$ per iteration) and converges in finitely many steps, but depends strongly on the start: use `n_init` or k-means++ seeding.
- Parameters: `n_clusters`, `init`, `n_init`; it is the $m \to 1$ limit of FCM and the hard reference for all fuzzy results.

Algorithm 3.2 k-medoids

Input: data $\mathbf{X}$, number of clusters c, tolerance ϵ, medoid rule (exact or nearest to the mean)

initialize the prototypes $\mathbf{v}_i^{(0)}$; $l \leftarrow 0$;
repeat
  $l \leftarrow l + 1$;
  distances and crisp partition as in algorithm 3.1;
  for $i = 1, \dots, c$ **do**
    exact: $\mathbf{v}_i \leftarrow \arg\min_{\mathbf{x} \in A_i} \sum_{\mathbf{y} \in A_i} \|\mathbf{x} - \mathbf{y}\|$; nearest mean: $\mathbf{v}_i \leftarrow \arg\min_{\mathbf{x} \in A_i} \|\mathbf{x} - \bar{\mathbf{x}}_{A_i}\|$;
until *no sample changed cluster*;

Output: medoids $\mathbf{V} \subset \mathbf{X}$ with their indices, crisp $\mathbf{U}$, J

keeps the nearest-prototype assignment of k-means but requires every prototype to be one of the samples, $\mathbf{v}_i \in \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$. Two rules choose the prototype of a cluster among its members. The exact rule takes the *medoid*, the member with the smallest total distance to the other members; it minimizes the sum of distances rather than of squared distances, which is the criterion of the PAM algorithm of Kaufman and Rousseeuw. The MATLAB toolbox took the member closest to the cluster mean, which is cheaper and is the exact minimizer of the within-cluster sum of squares (3.1) among the members, because the total squared distance of a member to all members equals N_i times its squared distance to the mean plus a constant. `fuzzyclust` offers both (`medoid="exact"` or `"nearest_mean"`). Because prototypes are data points, k-medoids is less sensitive to outliers than k-means (an outlier can pull a mean but cannot become a medoid unless it forms a cluster of its own), and the medoids can be reported as representative samples. The row indices of the medoids are kept in `medoid_indices_`; the geodesic fuzzy c-medoids of chapter 7 build on the same idea with a different distance.

In a nutshell k-medoids

- k-means with prototypes restricted to data points; the exact medoid minimizes the total distance to the other members (the PAM criterion), the cheaper nearest-to-mean rule the total squared distance.
- Use when the mean is not a valid object (discrete states, codes) or when representative samples are wanted; more robust to outliers than k-means.
- Exact medoids cost $\mathcal{O}(\sum_i N_i^2 n)$ per iteration; same start dependence as k-means.
- Parameters: `n_clusters`, `medoid`, `init`, `n_init`; attribute `medoid_indices_`.

3.2.3 Fuzzy c-means

Fuzzy c-means [2, 3] minimizes the c-means functional over the fuzzy partitioning space,

$$J(\mathbf{X};\mathbf{U},\mathbf{V}) = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^{m} D_{ik\mathbf{A}}^2, \qquad D_{ik\mathbf{A}}^2 = (\mathbf{x}_k - \mathbf{v}_i)^{\mathsf{T}}\mathbf{A}(\mathbf{x}_k - \mathbf{v}_i), \tag{3.3}$$

where $\mathbf{U} \in M_{fc}$, $m > 1$, and $\mathbf{A}$ is a fixed norm-inducing matrix (identity, diagonal, or Mahalanobis, section 2.2.2). The stationary points are found by adjoining the column constraint $\sum_i \mu_{ik} = 1$ with Lagrange multipliers λ_k,

$$\bar{J}(\mathbf{X};\mathbf{U},\mathbf{V},\boldsymbol{\lambda}) = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^{m} D_{ik\mathbf{A}}^2 + \sum_{k=1}^{N} \lambda_k \Big(\sum_{i=1}^{c} \mu_{ik} - 1 \Big), \tag{3.4}$$

and setting the partial derivatives to zero. The derivative with respect to a single membership gives

$$m\,\mu_{ik}^{m-1} D_{ik\mathbf{A}}^2 + \lambda_k = 0 \quad \Longrightarrow \quad \mu_{ik} = \left(\frac{-\lambda_k}{m}\right)^{\frac{1}{m-1}} \left(D_{ik\mathbf{A}}^2\right)^{-\frac{1}{m-1}}, \tag{3.5}$$

and summing (3.5) over i under the column constraint gives

$$\sum_{j=1}^{c} \left(\frac{-\lambda_k}{m}\right)^{\frac{1}{m-1}} \left(D_{jk\mathbf{A}}^2\right)^{-\frac{1}{m-1}} = 1,$$

so that dividing (3.5) by this sum eliminates the multiplier:

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{c} \left(D_{ik\mathbf{A}} / D_{jk\mathbf{A}}\right)^{2/(m-1)}}, \qquad 1 \le i \le c,\ 1 \le k \le N. \tag{3.6}$$

The derivative with respect to $\mathbf{v}_i$ gives $-2\mathbf{A}\sum_k \mu_{ik}^m(\mathbf{x}_k - \mathbf{v}_i) = 0$, hence

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} \mu_{ik}^m \mathbf{x}_k}{\sum_{k=1}^{N} \mu_{ik}^m}, \qquad 1 \le i \le c, \tag{3.7}$$

independently of $\mathbf{A}$. If $D^2_{ik\mathbf{A}} > 0$ for all i, k and $m > 1$, then $(\mathbf{U}, \mathbf{V})$ can minimize (3.3) only if (3.6) and (3.7) hold [3]; the solution automatically satisfies the remaining constraints of M_{fc}. Equation (3.7) is the mean of the data weighted by the memberships raised to m, which is why the algorithm is called c-*means*; (3.6) makes the membership of a sample in a cluster a decreasing function of its distance to that prototype *relative* to its distances to all prototypes. The FCM algorithm is the Picard iteration through the two equations, that is, the template of algorithm 2.1 with the Euclidean (or $\mathbf{A}$-) norm. If some $D_{ik\mathbf{A}} = 0$, the membership formula is undefined; `fuzzyclust`, like the MATLAB toolbox, adds 10^{-10} to the squared distances, which gives the sample a membership of practically one in the coinciding prototype.

The role of the fuzziness exponent

The exponent m determines how fast the membership decays with the relative distance. Writing (3.6) for two prototypes along the line through them (fig. 3.5) makes this visible. As $m \to 1^+$ the exponent $2/(m-1)$ grows without bound, each ratio $(D_{ik}/D_{jk})^{2/(m-1)}$ tends to 0 or ∞, and the partition becomes crisp: FCM turns into k-means. As $m \to \infty$ the exponent tends to zero, every ratio tends to one, and $\mu_{ik} \to 1/c$ for every sample; the prototypes (3.7) then all coincide with the grand mean of the data. In between, m trades the sharpness of the partition against its robustness: with a small m every sample pulls mainly its own prototype, with a large m every prototype averages over all samples. Two further properties of (3.6) matter in practice. First, the membership is one only *at* the prototype, and a sample far from all prototypes receives nearly uniform memberships $1/c$; this is the price of the column constraint, which the possibilistic variants of chapter 6 remove. Second, the memberships change more slowly for larger m, so above $m \approx 2$ the number of iterations needed to reach a given tolerance grows with m; close to $m = 1$ it rises again, because the nearly crisp partition takes several iterations to settle (section 3.4.2). The usual default is $m = 2$, for which (3.6) reduces to $\mu_{ik} = D_{ik}^{-2} / \sum_j D_{jk}^{-2}$, and values in $[1.5, 2.5]$ are the range to explore; the validity indices of chapter 4 can be scanned over m as well as over c.

3.2.4 Gustafson–Kessel

Gustafson and Kessel [18] extended FCM with an *adaptive* norm: each cluster has its own norm-inducing matrix $\mathbf{A}_i$, which becomes an optimization variable,

$$J(\mathbf{X}; \mathbf{U}, \mathbf{V}, \{\mathbf{A}_i\}) = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \, D^2_{ik\mathbf{A}_i}, \qquad D^2_{ik\mathbf{A}_i} = (\mathbf{x}_k - \mathbf{v}_i)^\mathsf{T} \mathbf{A}_i (\mathbf{x}_k - \mathbf{v}_i). \tag{3.8}$$

For fixed $\mathbf{A}_i$ the memberships and prototypes are those of FCM, (3.6) and (3.7). The functional cannot be minimized over $\mathbf{A}_i$ directly, because it is linear in $\mathbf{A}_i$ and can

Algorithm 3.3 Fuzzy c-means (FCM)

Input: data $\mathbf{X}$, number of clusters $1 < c < N$, exponent $m > 1$, tolerance $\epsilon > 0$, norm-inducing matrix $\mathbf{A}$

initialize $\mathbf{U}^{(0)} \in M_{fc}$; $l \leftarrow 0$;
repeat
  $l \leftarrow l + 1$;
  prototypes: $\mathbf{v}_i^{(l)} = \sum_k (\mu_{ik}^{(l-1)})^m \mathbf{x}_k \big/ \sum_k (\mu_{ik}^{(l-1)})^m$, $1 \le i \le c$;
  distances: $D_{ik\mathbf{A}}^2 = (\mathbf{x}_k - \mathbf{v}_i^{(l)})^\mathsf{T} \mathbf{A} (\mathbf{x}_k - \mathbf{v}_i^{(l)})$, $1 \le i \le c$, $1 \le k \le N$;
  partition: $\mu_{ik}^{(l)} = 1 \big/ \sum_j (D_{ik\mathbf{A}} / D_{jk\mathbf{A}})^{2/(m-1)}$;
until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: $\mathbf{V}$, $\mathbf{U} \in M_{fc}$, J

In a nutshell Fuzzy c-means

- Minimizes $\sum_{ik} \mu_{ik}^m D_{ik\mathbf{A}}^2$ over fuzzy partitions; prototypes are μ_{ik}^m-weighted means, memberships are inverse relative distances raised to $2/(m-1)$.
- One global norm $\mathbf{A}$: spherical clusters (Euclidean), axis-parallel or commonly oriented ellipsoids (diagonal, Mahalanobis); clusters of similar size.
- $m \to 1$: k-means; $m \to \infty$: all memberships $1/c$, prototypes collapse; default $m = 2$, explore $[1.5, 2.5]$.
- $\mathcal{O}(Ncn)$ per iteration, insensitive to the initialization on compact data, the standard initializer for GK and GG.
- Parameters: `n_clusters`, `m`, `norm`; attributes `norm_matrix_`, `covariances_` (for inspection).

be made arbitrarily small by making $\mathbf{A}_i$ less positive definite. The matrices must therefore be constrained, and the natural constraint fixes their determinants,

$$\det \mathbf{A}_i = \rho_i, \qquad \rho_i > 0, \tag{3.9}$$

which lets every cluster adapt its *shape* while its *volume* stays constant. Adjoining (3.9) with multipliers λ_i and differentiating with respect to $\mathbf{A}_i$ gives $\sum_k \mu_{ik}^m (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^\mathsf{T} = \lambda_i \det(\mathbf{A}_i) \mathbf{A}_i^{-1}$, so $\mathbf{A}_i$ is proportional to the inverse of the *fuzzy covariance matrix* of the cluster, and the constraint fixes the factor:

$$\mathbf{A}_i = \left[\rho_i \det(\mathbf{F}_i) \right]^{1/n} \mathbf{F}_i^{-1}, \qquad \mathbf{F}_i = \frac{\sum_{k=1}^N \mu_{ik}^m (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^\mathsf{T}}{\sum_{k=1}^N \mu_{ik}^m}. \tag{3.10}$$

Substituting (3.10) into (3.8) shows that the GK distance is a Mahalanobis distance with the cluster's own membership-weighted covariance, multiplied by the volume

normalizer $[\rho_i \det \mathbf{F}_i]^{1/n}$. Without prior knowledge all ρ_i are set to one, so GK can only find clusters of approximately equal volume; a cluster known to be larger can be given a larger ρ_i (section 3.4.4). In exact arithmetic, any invertible affine transformation of the data multiplies the GK distances of all samples by a common factor, so the partition is affine-invariant (section 2.2.8).

Eigenstructure. The covariance matrix $\mathbf{F}_i = \mathbf{\Phi}_i \mathbf{\Lambda}_i \mathbf{\Phi}_i^{\mathsf{T}}$ describes the shape and orientation of the cluster: the eigenvectors $\boldsymbol{\phi}_{ij}$ are the axes of the ellipsoid and the square roots of the eigenvalues λ_{ij} its semi-axis lengths (fig. 2.1). The ratio of the largest to the smallest eigenvalue measures the elongation, and the eigenvector of the smallest eigenvalue is the *normal* of the cluster: if the data of cluster i lie close to a hyperplane, that hyperplane is $\boldsymbol{\phi}_{i1}^{\mathsf{T}}(\mathbf{x} - \mathbf{v}_i) = 0$. This is the bridge from GK to local linear models: in the product space of regressors and output, the normal of a GK cluster gives the parameter vector of a local linear model, which chapter 8 exploits. `fuzzyclust` stores the eigenvalues in ascending order in `eigenvalues_` and the eigenvectors as columns of `eigenvectors_`.

Regularization. The adaptive norm has a numerical weak spot [19]. When a cluster has few members or its members lie almost on a linear subspace, $\mathbf{F}_i$ is nearly singular: its smallest eigenvalue approaches zero, the condition number explodes, $\mathbf{F}_i^{-1}$ is unreliable, and the volume normalization fails as $\det \mathbf{F}_i \to 0$. GK can even converge to a degenerate solution in which one cluster collapses onto a line of points, whose zero volume makes its distance to those points zero. Babuška, van der Veen, and Kaymak proposed two independent safeguards, both applied to $\mathbf{F}_i$ before inversion:

$$\mathbf{F}_i \leftarrow (1-\gamma)\mathbf{F}_i + \gamma \, \det(\mathbf{F}_0)^{1/n} \, \mathbf{I}, \qquad 0 \le \gamma \le 1, \tag{3.11}$$

$$\lambda_{ij} \leftarrow \lambda_{i,\max}/\beta \quad \text{for all } j \text{ with } \lambda_{i,\max}/\lambda_{ij} > \beta,$$

$$\mathbf{F}_i \leftarrow \mathbf{\Phi}_i \, \operatorname{diag}(\lambda_{i1}, \ldots, \lambda_{in}) \, \mathbf{\Phi}_i^{\mathsf{T}}. \tag{3.12}$$

The first shrinks every cluster covariance toward a scaled identity; $\mathbf{F}_0$ is the covariance of the whole data set, so the scale is that of the data. It is recommended for small or nearly collinear clusters and for identification, where it keeps the local models from overfitting. The second bounds the condition number of $\mathbf{F}_i$ by β by raising the small eigenvalues; the MATLAB default $\beta = 10^{15}$ is effectively off, and a bound between 10^2 and 10^4 is a practical choice when degenerate clusters appear; it acts only on clusters whose condition number exceeds it (section 3.4.4). `fuzzyclust` implements both as published, with the defaults $\gamma = 0$ and $\beta = 10^{15}$, and stores the regularized $\mathbf{F}_i$ in `covariances_`. Both devices break the affine invariance of GK, because the identity matrix and the eigenvalues of $\mathbf{F}_i$ depend on the units of the data.

Algorithm 3.4 Gustafson–Kessel (GK) with covariance regularization

Input: data $\mathbf{X}$, number of clusters $1 < c < N$, exponent $m > 1$, tolerance ϵ, volumes ρ_i (default 1), regularization $\gamma \in [0, 1]$, condition bound β

compute the data covariance $\mathbf{F}_0$; initialize $\mathbf{U}^{(0)} \in M_{fc}$; $l \leftarrow 0$;

repeat

- $l \leftarrow l + 1$;
- prototypes: $\mathbf{v}_i^{(l)} = \sum_k (\mu_{ik}^{(l-1)})^m \mathbf{x}_k \big/ \sum_k (\mu_{ik}^{(l-1)})^m$;
- covariances: $\mathbf{F}_i = \sum_k (\mu_{ik}^{(l-1)})^m (\mathbf{x}_k - \mathbf{v}_i^{(l)})(\mathbf{x}_k - \mathbf{v}_i^{(l)})^\top \big/ \sum_k (\mu_{ik}^{(l-1)})^m$;
- regularize: $\mathbf{F}_i \leftarrow (1-\gamma)\mathbf{F}_i + \gamma \det(\mathbf{F}_0)^{1/n}\mathbf{I}$; eigen-decompose $\mathbf{F}_i$ and raise eigenvalues below $\lambda_{i,\max}/\beta$ to that value (3.12);
- norms: $\mathbf{A}_i = [\rho_i \det \mathbf{F}_i]^{1/n} \mathbf{F}_i^{-1}$;
- distances: $D^2_{ik\mathbf{A}_i} = (\mathbf{x}_k - \mathbf{v}_i^{(l)})^\top \mathbf{A}_i (\mathbf{x}_k - \mathbf{v}_i^{(l)})$;
- partition: $\mu_{ik}^{(l)} = 1 \big/ \sum_j (D_{ik\mathbf{A}_i} / D_{jk\mathbf{A}_j})^{2/(m-1)}$;

until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: $\mathbf{V}$, $\mathbf{U}$, $\{\mathbf{F}_i\}$, $\{\mathbf{A}_i\}$, eigenvalues and eigenvectors of $\mathbf{F}_i$, J

In a nutshell Gustafson–Kessel

- FCM with a norm-inducing matrix per cluster, $\mathbf{A}_i = [\rho_i \det \mathbf{F}_i]^{1/n} \mathbf{F}_i^{-1}$; the determinant constraint $\det \mathbf{A}_i = \rho_i$ fixes the volume and lets the shape adapt.
- Detects ellipsoidal clusters of different orientation and elongation but (for $\rho_i = 1$) of similar volume; affine-invariant in exact arithmetic.
- The eigenstructure of $\mathbf{F}_i$ gives axes, elongation, and the normal direction of a cluster—the basis of local linear models.
- Nearly singular $\mathbf{F}_i$ (few points, collinear data) is handled by shrinkage γ toward $\det(\mathbf{F}_0)^{1/n}\mathbf{I}$ and by the condition bound β.
- $\mathcal{O}(Ncn^2 + cn^3)$ per iteration; more start-dependent than FCM on crossing or elongated clusters, so use `n_init`.
- Parameters: `n_clusters`, `m`, `rho`, `gamma`, `beta`; attributes `covariances_`, `norm_matrices_`, `eigenvalues_`, `eigenvectors_`.

3.2.5 Gath–Geva

The fuzzy maximum likelihood estimation (FMLE) algorithm of Bezdek and Dunn [20], developed into a clustering method by Gath and Geva [21], interprets cluster i as a Gaussian component with mean $\mathbf{v}_i$, covariance $\mathbf{F}_i$, and prior probability α_i, and uses the reciprocal of the prior-weighted density as distance,

$$D_{ik}^2 = \frac{\sqrt{\det \mathbf{F}_{wi}}}{\alpha_i} \exp\!\left(\tfrac{1}{2}(\mathbf{x}_k - \mathbf{v}_i)^{\mathsf{T}} \mathbf{F}_{wi}^{-1} (\mathbf{x}_k - \mathbf{v}_i)\right), \tag{3.13}$$

with

$$\mathbf{F}_{wi} = \frac{\sum_{k=1}^{N} \mu_{ik}^{w} (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^{\mathsf{T}}}{\sum_{k=1}^{N} \mu_{ik}^{w}}, \qquad \alpha_i = \frac{1}{N} \sum_{k=1}^{N} \mu_{ik}. \tag{3.14}$$

The covariance weighting exponent is $w = 1$ in the original FMLE algorithm; the MATLAB toolbox and `fuzzyclust` use $w = m$ (so $w = 2$ by default), which makes the partition fuzzier to compensate for the exponential term. The two weightings generalize the classical covariance from different starting points, and the exponent w covers both. Memberships and prototypes are updated with the FCM formulas (3.6) and (3.7). Three consequences follow from (3.13). First, $1/D_{ik}^2$ is proportional to $\alpha_i \, \mathcal{N}(\mathbf{x}_k; \mathbf{v}_i, \mathbf{F}_{wi})$, so for $m = 2$ the membership $\mu_{ik} = D_{ik}^{-2} / \sum_j D_{jk}^{-2}$ is exactly Bayes' rule for the posterior probability of component i given $\mathbf{x}_k$: GG is a fuzzy relative of the EM algorithm for Gaussian mixtures and differs from it in the weights μ_{ik}^{m} of the center and covariance updates. Second, volume *and* density adapt: neither $\det \mathbf{F}_i$ nor α_i is constrained, so a large diffuse cluster next to a small dense one is described correctly, whereas GK's volume constraint forces the dense cluster to steal points (section 3.4.6). Third, the distance grows exponentially with the Mahalanobis distance instead of quadratically, so the memberships fall off fast, the partition is nearly crisp between the clusters, and the algorithm converges quickly, but to the *nearest* local optimum. GG is therefore more sensitive to the start than FCM or GK and needs a good initialization; the recommended practice, in the MATLAB toolbox and here, is to pass the partition of a converged FCM or GK run as `init`.

Evaluation in the log domain. The exponential in (3.13) overflows a double for squared Mahalanobis distances above about 1400, which happens routinely for far samples and small clusters; the MATLAB manual warns that GG "can overflow at large c values." `fuzzyclust` therefore works with the logarithm $\log D_{ik}^2 = \tfrac{1}{2} \log\det \mathbf{F}_{wi} - \log \alpha_i + \tfrac{1}{2}(\mathbf{x}_k - \mathbf{v}_i)^{\mathsf{T}} \mathbf{F}_{wi}^{-1} (\mathbf{x}_k - \mathbf{v}_i)$, computes it with a stable log-determinant, caps it at 700 (so that D_{ik}^2 stays below the largest finite double), and evaluates the membership formula as a softmax over clusters,

$$\mu_{ik} = \frac{\exp\!\left(-\frac{1}{m-1} \log D_{ik}^2\right)}{\sum_j \exp\!\left(-\frac{1}{m-1} \log D_{jk}^2\right)}, \tag{3.15}$$

after subtracting the largest logit per sample. Below the cap this is algebraically identical to (3.6), and it cannot overflow. The distances reported by `transform` and

Algorithm 3.5 Gath–Geva (GG, fuzzy maximum likelihood estimation)

Input: data $\mathbf{X}$, number of clusters c, exponent m (1: original FMLE, 2: recommended), tolerance ϵ, regularization γ, β

initialize $\mathbf{U}^{(0)}$ with a less start-dependent method (the partition of FCM or GK); $l \leftarrow 0$;

repeat

  $l \leftarrow l + 1$;

  prototypes: $\mathbf{v}_i^{(l)} = \sum_k (\mu_{ik}^{(l-1)})^m \mathbf{x}_k \big/ \sum_k (\mu_{ik}^{(l-1)})^m$;

  covariances $\mathbf{F}_{wi}$ with $w = m$ by (3.14), regularized as in algorithm 3.4; priors $\alpha_i = \frac{1}{N} \sum_k \mu_{ik}^{(l-1)}$;

  log-distances: $\log D_{ik}^2 = \frac{1}{2} \log \det \mathbf{F}_{wi} - \log \alpha_i + \frac{1}{2} (\mathbf{x}_k - \mathbf{v}_i^{(l)})^\top \mathbf{F}_{wi}^{-1} (\mathbf{x}_k - \mathbf{v}_i^{(l)})$;

  partition: $\mu_{ik}^{(l)}$ by the softmax (3.15);

until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: $\mathbf{V}$, $\mathbf{U}$, $\{\mathbf{F}_{wi}\}$, priors $\{\alpha_i\}$, J

In a nutshell Gath–Geva

- Distance = reciprocal of a prior-weighted Gaussian density; memberships are posterior probabilities (for $m = 2$), the clusters a Gaussian mixture with fuzzy weights.
- Volume, orientation *and* density of every cluster adapt; handles clusters of very different size and spread, which FCM and GK cannot.
- Exponential distance: nearly crisp borders, fast convergence to the nearest optimum, hence sensitive to the start—initialize from FCM or GK (`init=fcm.membership_`).
- Computed in the log domain (softmax) to avoid the overflow of the exponential; covariances regularized like GK if needed.
- Parameters: `n_clusters`, `m`, `gamma`, `beta`, `init`; attributes `covariances_`, `priors_`, `eigenvalues_`, `eigenvectors_`.

`distances_` are the square roots of the capped values, $D_{ik} \leq e^{350}$. The covariances can be regularized exactly as in GK (`gamma`, `beta`), which is advisable when the clusters have few samples.

3.2.6 Comparison

Table 3.1 summarizes the four algorithms. Read from left to right, every column adds freedom to the cluster description and removes an assumption about the data, at the cost of more parameters per cluster (from n for a center to $n + n(n+1)/2 + 1$

for center, covariance, and prior), more computation (the $n \times n$ inversions of GK and GG), and greater sensitivity to the initial partition. In practice the data decide: FCM when the clusters are compact and the features scaled, GK when they are elongated or differently oriented, GG when their sizes or densities differ, and FCM first in any case, because it initializes the other two. The validity indices of the next chapter compare the resulting partitions on a common footing, and the examples below show what happens when each assumption is violated.

Table 3.1. The four partitioning algorithms at a glance. N samples, n features, c clusters.

	k-means / k-medoids	FCM	GK	GG
prototype	mean / medoid $\mathbf{v}_i$	center $\mathbf{v}_i$	center $\mathbf{v}_i$, covariance $\mathbf{F}_i$	center $\mathbf{v}_i$, covariance $\mathbf{F}_{wi}$, prior α_i
distance	Euclidean	fixed norm $\mathbf{A}$, eq. (3.3)	volume-scaled Mahalanobis, eq. (3.10)	inverse prior-weighted Gaussian density, eq. (3.13)
partition	crisp, M_{hc}	fuzzy, M_{fc}	fuzzy, M_{fc}	fuzzy, nearly crisp borders
cluster shape	spherical, equal size	spherical (or one common ellipsoid), similar size	ellipsoidal, any orientation, equal volume	ellipsoidal, any orientation, size, and density
parameters	c	c, m, $\mathbf{A}$	c, m, ρ_i, γ, β	c, m, γ, β
cost per iteration	$\mathcal{O}(Ncn)$ (medoids: $+\sum_i N_i^2 n$)	$\mathcal{O}(Ncn)$	$\mathcal{O}(Ncn^2 + cn^3)$	$\mathcal{O}(Ncn^2 + cn^3)$
start dependence	high	low on compact data	medium (crossing or elongated clusters)	high: initialize from FCM/GK
typical use	fast reference, representative samples	compact clusters, initializer	oriented clusters, local linear models	unequal sizes and densities, mixture interpretation

3.2.7 What is fuzzy about fuzzy clustering? Fuzzifiers and regularization

The fuzzy algorithms above owe their fuzziness to one device: the memberships enter the functional as μ_{ik}^m with $m > 1$. Without the exponent ($m = 1$) the c-means functional is *linear* in every column of $\mathbf{U}$, and a linear function on the simplex $\{\boldsymbol{\mu}_k : \sum_i \mu_{ik} = 1,\ \mu_{ik} \geq 0\}$ attains its minimum at a vertex: every sample goes

entirely to its nearest prototype and the partition is crisp. Fuzziness therefore requires that the cost of a column be strictly convex in the memberships, and there are three ways to obtain this: raise the memberships to the power m as FCM does, transform them with another *fuzzifier* function [22], or keep them linear and add a strictly convex *regularization term* [23]. The choice decides the shape of the membership functions (fig. 3.2). This section presents the two regularizations, the deterministic-annealing view of the entropy regularization, and the polynomial fuzzifier.

Entropy regularization. Li and Mukaidono [24] and Miyamoto and Mukaidono [23] add the negative entropy of the partition to the hard functional,

$$J_E(\mathbf{X};\mathbf{U},\mathbf{V}) = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik} D_{ik}^2 + \lambda \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik} \log \mu_{ik}, \qquad \lambda > 0. \tag{3.16}$$

Adjoining the column constraint as in (3.4) and differentiating gives $D_{ik}^2 + \lambda(\log \mu_{ik} + 1) + \lambda_k = 0$, so $\mu_{ik} \propto \exp(-D_{ik}^2/\lambda)$, and normalization yields the Gibbs (softmax) memberships

$$\mu_{ik} = \frac{\exp(-D_{ik}^2/\lambda)}{\sum_{j=1}^{c} \exp(-D_{jk}^2/\lambda)}, \qquad \mathbf{v}_i = \frac{\sum_k \mu_{ik}\mathbf{x}_k}{\sum_k \mu_{ik}}. \tag{3.17}$$

The prototypes are weighted means with the memberships themselves, not their powers, as weights. Two properties distinguish (3.17) from (3.6). First, the membership depends on the *differences* $D_{jk}^2 - D_{ik}^2$ of the squared distances, not on their ratios: far from all prototypes a sample belongs entirely to the nearest one instead of tending to $1/c$, and the membership function of a cluster is a Gaussian bump modulated by its neighbors. Second, λ has the units of a squared distance. It acts as a *temperature*, with $\lambda \to 0$ giving k-means and $\lambda \to \infty$ uniform memberships, and it must be chosen relative to the squared radius of the clusters (standardize the data first).

Deterministic annealing. Eliminating the memberships from (3.16) with (3.17) turns the functional into the free energy of statistical physics,

$$F(\mathbf{V};\lambda) = -\lambda \sum_{k=1}^{N} \log \sum_{i=1}^{c} \exp\!\left(-D_{ik}^2/\lambda\right), \tag{3.18}$$

and Rose's deterministic annealing [25] minimizes it while the temperature is lowered. At a high temperature F has a single minimum, at which every prototype sits at the grand mean of the data. Lowering λ geometrically and re-converging from the previous solution tracks this minimum until it becomes unstable: a cluster with covariance $\mathbf{F}_i$ (weights μ_{ik}) *splits* when λ falls below the critical temperature

$$\lambda_{c,i} = 2\,\lambda_{\max}(\mathbf{F}_i), \tag{3.19}$$

and the first split occurs at twice the largest eigenvalue of the data covariance. The number of distinct prototypes thus grows in *phase transitions* as the temperature falls, and the annealing history reveals the cluster structure scale by scale: the temperatures at which the number of distinct prototypes jumps are the squared scales of the structure (section 3.4.9). Because the start is the unique high-temperature minimum, annealing removes the dependence on the initial partition, at the price of one convergence per temperature. Coinciding prototypes must be perturbed slightly after every cooling step; otherwise they stay together by symmetry.

Quadratic regularization. Miyamoto and Umayahara [26] use the regularizer $\lambda \sum_{ik} \mu_{ik}^2$ instead,

$$J_Q(\mathbf{X}; \mathbf{U}, \mathbf{V}) = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik} D_{ik}^2 + \lambda \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^2. \tag{3.20}$$

The stationarity condition $D_{ik}^2 + 2\lambda\mu_{ik} + \lambda_k = 0$ is linear in μ_{ik}, and it can hold only for the clusters of an *active set* A_k; for the others the nonnegativity constraint binds and $\mu_{ik} = 0$. With A_k the $\hat{c}_k$ nearest prototypes,

$$\mu_{ik} = \frac{1}{\hat{c}_k}\left(1 + \frac{1}{2\lambda} \sum_{j \in A_k} \left(D_{jk}^2 - D_{ik}^2\right)\right) \ (i \in A_k), \qquad \mu_{ik} = 0 \ \ (i \notin A_k), \tag{3.21}$$

where $\hat{c}_k$ is the largest number of nearest prototypes for which all these values are positive. Equivalently, the column $\boldsymbol{\mu}_k$ is the Euclidean projection of $-\mathbf{D}_{\cdot k}^2/(2\lambda)$ onto the simplex, which is computed exactly by sorting. For two clusters, $\mu_{1k} = \frac{1}{2} + (D_{2k}^2 - D_{1k}^2)/(4\lambda)$ clipped to $[0, 1]$: a linear ramp between the prototypes and *exact* zeros and ones outside it.

The polynomial fuzzifier. Klawonn and Höppner [22] asked why FCM never assigns a zero membership and found the answer in the derivative of the fuzzifier: $\frac{d}{d\mu}\mu^m = 0$ at $\mu = 0$, so the marginal cost of giving a sample a tiny membership in a far-away cluster is zero, and the optimum always does so. Every FCM prototype is therefore pulled, if weakly, by every sample of the data set. Any fuzzifier g with $g'(0) > 0$ removes this effect; they proposed the quadratic polynomial

$$g(\mu) = \frac{1-\beta}{1+\beta}\,\mu^2 + \frac{2\beta}{1+\beta}\,\mu, \qquad 0 \le \beta < 1, \tag{3.22}$$

which interpolates between μ^2 ($\beta = 0$, FCM with $m = 2$) and μ ($\beta \to 1$, k-means), and minimized $J = \sum_{ik} g(\mu_{ik}) D_{ik}^2$. The Kuhn–Tucker conditions give, for the active set A_k of the $\hat{c}_k$ nearest prototypes,

$$\mu_{ik} = \frac{1}{1-\beta}\left(\frac{1 + (\hat{c}_k - 1)\beta}{\sum_{j \in A_k} D_{ik}^2 / D_{jk}^2} - \beta\right) \ (i \in A_k), \qquad \mu_{ik} = 0 \ \ (i \notin A_k), \tag{3.23}$$

with $\hat{c}_k$ the largest size for which all values are positive, and prototypes $\mathbf{v}_i = \sum_k g(\mu_{ik})\mathbf{x}_k / \sum_k g(\mu_{ik})$. For two clusters the second membership vanishes as soon

as $D_{1k}^2/D_{2k}^2 \le \beta$: a sample whose nearest prototype is at distance D_1 ignores every cluster farther than $D_1/\sqrt{\beta}$. The memberships are thus fuzzy between neighboring clusters and crisp around each prototype; only a sample far from all prototypes, whose distance ratios approach one, again shares its membership among them. The distances still enter as ratios, so the result is scale-invariant, unlike that of the regularizations. The authors recommend $\beta = 0.5$. The parameter keeps the name β of the paper and of the code parameter `beta`; it is unrelated to the condition-number bound β of GK.

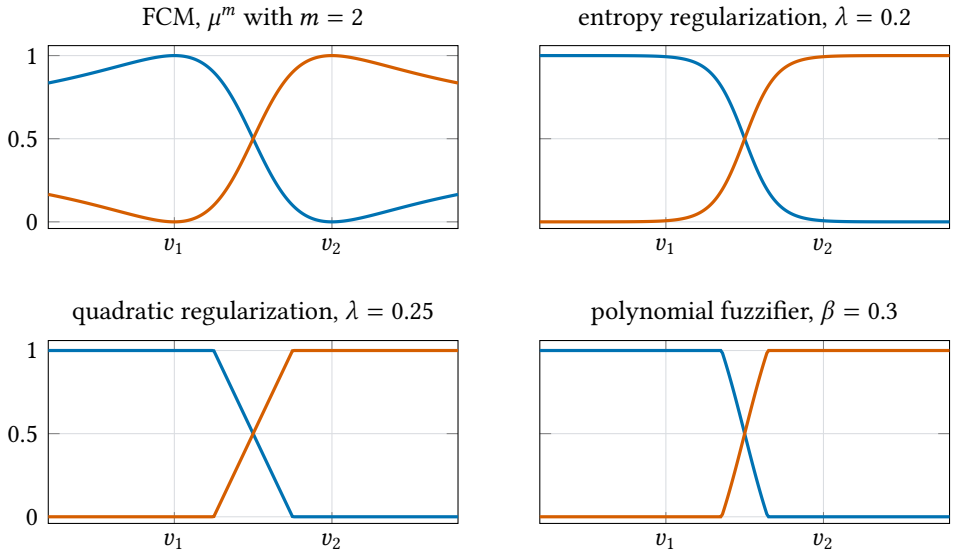


Figure 3.2. Membership functions of two clusters with prototypes $v_1 = 0$ (blue) and $v_2 = 1$ (vermilion) under four fuzzification mechanisms. FCM: the membership is one only at the prototypes and returns toward $1/2$ far away. Entropy regularization: a softmax of the squared-distance differences, crisp far from the border. Quadratic regularization: a linear ramp with exact zeros and ones outside the band $|D_2^2 - D_1^2| < 2\lambda$. Polynomial fuzzifier: exact zeros wherever one squared distance is less than β times the other; fuzzy only between the prototypes.

3.2.8 Estimating the number and location of clusters from density

The iterative algorithms of this chapter need the number of clusters and a start. Before the validity indices of chapter 4 judge finished partitions, a cheaper question can be put to the data directly: where the density peaks are and how many there are. The mountain method of Yager and Filev [27] and its data-point version, the subtractive clustering of Chiu [28], answer it in one pass without iteration and without random initialization (fig. 3.3).

Algorithm 3.6 Regularized fuzzy c-means and deterministic annealing

Input: data $\mathbf{X}$, number of clusters c, regularization (entropy or quadratic) with weight $\lambda > 0$, tolerance ϵ; for annealing a cooling factor $\alpha \in (0, 1)$ and a start temperature $\lambda_0 > 2\lambda_{\max}(\mathbf{F}_0)$

if *annealing* **then**
  all prototypes at the grand mean plus a small perturbation; $\lambda \leftarrow \lambda_0$;
else
  initialize $\mathbf{U}^{(0)}$ (random, k-means++, or given); λ fixed;
repeat
  repeat
    prototypes: $\mathbf{v}_i = \sum_k \mu_{ik}\mathbf{x}_k / \sum_k \mu_{ik}$ (linear weights);
    distances $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$;
    memberships: softmax (3.17) (entropy) or simplex projection (3.21) (quadratic);
  until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;
  record the number of distinct prototypes at this λ;
  if *annealing* **then** $\lambda \leftarrow \max(\alpha\lambda, \lambda_{\text{final}})$; perturb the prototypes slightly;
until λ *has reached its final value (immediately without annealing)*;

Output: $\mathbf{V}$, $\mathbf{U}$ (exact zeros for the quadratic term), J_E or J_Q; with annealing the history of distinct prototypes over λ and the critical temperatures $2\lambda_{\max}(\mathbf{F}_i)$

In a nutshell Regularized fuzzy c-means and deterministic annealing

- Hard c-means functional plus a convex regularizer: entropy (Gibbs/softmax memberships) or quadratic (piecewise linear memberships with exact zeros); prototypes are means with linear weights.
- λ is a squared length (temperature): $\lambda \to 0$ gives k-means, $\lambda \to \infty$ uniform memberships; scale it with the squared cluster radius, standardize the data.
- Entropy memberships depend on differences of squared distances: far samples belong fully to the nearest cluster.
- Deterministic annealing lowers λ from above $2\lambda_{\max}(\mathbf{F}_0)$; prototypes split at the critical temperatures $2\lambda_{\max}(\mathbf{F}_i)$, which reveals the cluster structure scale by scale and removes the start dependence.
- Parameters: `regularization`, `lam`, `annealing`, `lam_start`, `anneal_rate`, `perturbation`; attributes `cluster_history_`, `n_effective_clusters_`, `critical_temperatures_`.

Algorithm 3.7 Fuzzy c-means with the polynomial fuzzifier

Input: data $\mathbf{X}$, number of clusters c, fuzzifier parameter $0 \le \beta < 1$, tolerance ϵ

initialize $\mathbf{U}^{(0)}$; $l \leftarrow 0$;
repeat
  $l \leftarrow l + 1$;
  prototypes: $\mathbf{v}_i^{(l)} = \sum_k g(\mu_{ik}^{(l-1)})\mathbf{x}_k \big/ \sum_k g(\mu_{ik}^{(l-1)})$ with g from (3.22);
  distances $D_{ik}^2 = \left\|\mathbf{x}_k - \mathbf{v}_i^{(l)}\right\|^2$; sort them for every sample;
  for $k = 1, \dots, N$ **do**
    $\hat{c}_k \leftarrow$ largest number of nearest prototypes for which (3.23) is positive for all of them;
    $\mu_{ik}^{(l)}$ by (3.23) for the $\hat{c}_k$ nearest prototypes, 0 for the others;
until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: $\mathbf{V}$, $\mathbf{U}$ with exact zeros, number of active clusters per sample, $J = \sum_{ik} g(\mu_{ik}) D_{ik}^2$

In a nutshell Polynomial fuzzifier

- Replaces μ^m by $g(\mu) = \frac{1-\beta}{1+\beta}\mu^2 + \frac{2\beta}{1+\beta}\mu$, whose positive slope at zero makes zero memberships optimal for far clusters.
- Fuzzy between neighboring clusters, crisp around each prototype: a sample ignores every cluster farther than $D_1/\sqrt{\beta}$, so the members of one cluster no longer pull the distant prototypes.
- Distances enter as ratios, so the partition is scale-invariant like FCM; $\beta = 0$ is FCM with $m = 2$, $\beta \to 1$ k-means, $\beta = 0.5$ recommended.
- Same cost per iteration as FCM plus a sort of the c distances per sample; sparse membership matrix.
- Parameters: `n_clusters`, `beta`; attribute `n_active_`.

The mountain method. The data are scaled to the unit hypercube and a grid of nodes $\mathbf{g}$ is laid over it. Every node receives the value of the *mountain function*, the sum of Gaussian influences of all samples,

$$M(\mathbf{g}) = \sum_{k=1}^{N} \exp\left(-\alpha \left\|\mathbf{g} - \mathbf{x}_k\right\|^2\right), \qquad \alpha = 4/r_a^2, \tag{3.24}$$

where r_a is the neighborhood radius: samples farther than r_a contribute less than $e^{-4} \approx 0.02$. The highest node $\mathbf{g}_1^*$ with height M_1^* is the first center. Its mountain is then *destroyed*: $M(\mathbf{g}) \leftarrow M(\mathbf{g}) - M_1^* \exp(-4 \left\|\mathbf{g} - \mathbf{g}_1^*\right\|^2 / r_b^2)$ with a second radius

r_b, so that nodes near the center lose most of their height and distant nodes keep theirs. The next highest node is the next center. The procedure stops when the highest remaining node falls below a fraction δ of M_1^*. The grid is the method's weakness: with G nodes per axis there are G^n nodes, which is affordable for two or three features only.

Subtractive clustering. Chiu [28] took the data points themselves as the candidate centers, which makes the cost $\mathcal{O}(N^2 n)$, linear instead of exponential in the dimension. The *potential* of sample k is

$$P_k = \sum_{l=1}^{N} \exp\bigl(-\alpha \,\|\mathbf{x}_k - \mathbf{x}_l\|^2\bigr), \qquad \alpha = 4/r_a^2, \tag{3.25}$$

a kernel density estimate with bandwidth $r_a/\sqrt{8}$. After the selection of the center $\mathbf{x}_i^*$ with potential P_i^* all potentials are revised,

$$P_k \leftarrow P_k - P_i^* \exp\bigl(-4 \,\|\mathbf{x}_k - \mathbf{x}_i^*\|^2 / r_b^2\bigr), \qquad r_b = \eta\, r_a, \tag{3.26}$$

with the *squash factor* $\eta > 1$ (Chiu suggests $r_b = 1.5 r_a$; a common default is $\eta = 1.25$), so that no second center is placed close to an existing one. The stopping rule has a gray zone. A candidate with $P^* > \bar{\varepsilon} P_1^*$ is accepted, one with $P^* < \underline{\varepsilon} P_1^*$ ends the search ($\bar{\varepsilon} = 0.5$, $\underline{\varepsilon} = 0.15$ in Chiu's paper), and in between the candidate is accepted only if it is far enough from the existing centers relative to its potential, $d_{\min}/r_a + P^*/P_1^* \geq 1$; otherwise its potential is set to zero and the next candidate is examined. The radius r_a, a fraction of the data range, is the parameter that matters; it sets the scale at which the data are viewed, small radii giving many centers and large radii few. Counting the centers over a range of radii and looking for a *plateau* gives a quick, model-free estimate of the number of clusters (section 3.4.10).

Uses. The centers are good initial prototypes for FCM, GK, and GG, and their number is an estimate of c. Chiu's own application was fuzzy modeling: run on the product space $\mathbf{z}_k = [\mathbf{x}_k^\mathsf{T}\ y_k]^\mathsf{T}$ of inputs and output, every center becomes the center of a rule of a Takagi–Sugeno model, so the number of centers is a *rule-number estimate* and the radius controls the granularity of the model (chapter 8).

3.3 Using it in fuzzyclust

The five core estimators are in `fuzzyclust.cluster` and are re-exported at the package top level; the estimators of sections 3.2.7 and 3.2.8 are in `fuzzyclust.cluster` only. The iterative estimators share the constructor parameters, attributes, and methods of section 2.3, so only their algorithm-specific additions are listed here; the two density-based estimators have no `n_clusters`, `tol`, or `init` and are listed in full.

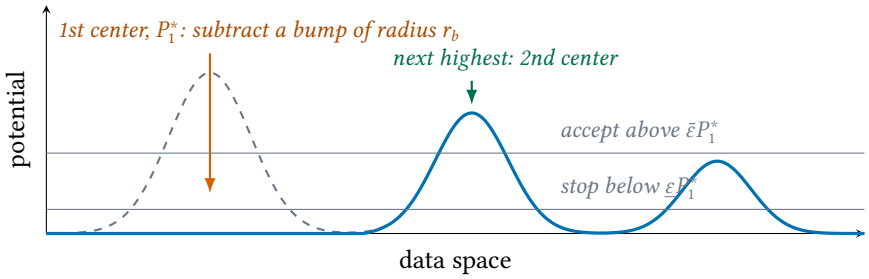


Figure 3.3. Subtractive clustering in one dimension. The dashed curve is the initial potential, a kernel density estimate of the data. The highest point becomes the first center and a Gaussian bump of radius $r_b = \eta r_a$ scaled to its potential is subtracted (solid curve); the highest remaining potential gives the next center. Candidates above $\bar{\varepsilon}P_1^*$ are accepted, the search stops below $\underline{\varepsilon}P_1^*$, and in between a candidate must also be far enough from the existing centers.

Algorithm 3.8 Subtractive clustering

Input: data $\mathbf{X}$, radius r_a (fraction of the data range, scalar or per feature), squash factor η, acceptance and rejection ratios $\bar{\varepsilon} > \underline{\varepsilon}$

scale the data to $[0,1]^n$; potentials P_k by (3.25); $\mathcal{C} \leftarrow \emptyset$;
while *true* **do**
  $k^* \leftarrow \arg\max_k P_k$, $P^* \leftarrow P_{k^*}$; for the first candidate $P_1^* \leftarrow P^*$;
  if $P^* > \bar{\varepsilon}P_1^*$ *or* $\mathcal{C} = \emptyset$ **then**
    accept
  else if $P^* < \underline{\varepsilon}P_1^*$ **then**
    stop
  else
    $d_{\min} \leftarrow$ distance of $\mathbf{x}_{k^*}$ to the nearest center in $\mathcal{C}$;
    if $d_{\min}/r_a + P^*/P_1^* \geq 1$ **then** accept;
    else $P_{k^*} \leftarrow 0$ and examine the next candidate;
  on acceptance: $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{x}_{k^*}\}$ and revise all potentials by (3.26);

Output: centers $\mathcal{C}$ (data points, original units), their number c, and relative potentials P_i^*/P_1^*

In a nutshell **Subtractive clustering**

- Selects data points of high potential (kernel density) as centers, subtracting the influence of every selected center; no number of clusters, no initialization, no iteration.
- One essential parameter: the radius r_a as a fraction of the data range; a plateau of the number of centers over r_a indicates a stable structure.
- $\mathcal{O}(N^2 n)$ time and $\mathcal{O}(N^2)$ memory: practical for thousands of samples in any dimension.
- Uses: initial centers and c for FCM/GK/GG; on product-space data a rule-number estimate for Takagi–Sugeno models.
- Parameters: `radii`, `squash`, `accept_ratio`, `reject_ratio`, `max_clusters`; attributes `n_clusters_`, `center_indices_`, `potentials_`, `initial_potentials_`.

Algorithm 3.9 **Mountain method**

Input: data $\mathbf{X}$, grid size G per feature, radius r_a, squash factor η, stopping ratio δ

1 scale the data to $[0,1]^n$; build the grid of G^n nodes; mountain function $M(\mathbf{g})$ by (3.24);
2 $\mathbf{g}_1^* \leftarrow \arg\max M$, $M_1^* \leftarrow M(\mathbf{g}_1^*)$; $i \leftarrow 1$;
3 **repeat**
4 destroy the mountain of the last center:
$M(\mathbf{g}) \leftarrow M(\mathbf{g}) - M_i^* \exp(-4\,\|\mathbf{g} - \mathbf{g}_i^*\|^2 / r_b^2)$, $r_b = \eta r_a$;
5 $i \leftarrow i + 1$; $\mathbf{g}_i^* \leftarrow \arg\max M$, $M_i^* \leftarrow M(\mathbf{g}_i^*)$;
6 **until** $M_i^* < \delta M_1^*$;

Output: the grid nodes $\mathbf{g}_1^*, \ldots, \mathbf{g}_{i-1}^*$ as centers, their number, the mountain function

In a nutshell **Mountain method**

- The grid-based predecessor of subtractive clustering: a density (mountain) function on grid nodes, peaks selected and destroyed one by one.
- Centers are grid nodes, so their precision is the grid spacing; the stop rule is a single ratio δ.
- G^n nodes: exponential in the number of features, practical for $n \leq 3$; `max_nodes` bounds the grid explicitly.
- Parameters: `grid_size`, `max_nodes`, `radii`, `squash`, `stop_ratio`; attributes `grid_`, `mountain_`.

fuzzyclust.cluster.KMeans

`n_clusters=3`, `tol=1e-4`, `max_iter=300`, `init="random"`, `n_init=1`, `random_state=None`, `verbose=0`. The exponent is fixed at $m = 1$ and the partition is crisp; `membership_` holds one-hot rows, `objective_` the within-cluster sum of squares. An emptied cluster keeps its previous prototype. For large data sets `sklearn.cluster.KMeans` is faster; this class exists so that hard and fuzzy methods share one interface and one loop.

fuzzyclust.cluster.KMedoids

`n_clusters=3`, `medoid="exact"`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`. `medoid="exact"` selects the member with minimal total distance to the other members, `"nearest_mean"` the member closest to the cluster mean (MATLAB behavior). `cluster_centers_` are rows of `X`, and `medoid_indices_` holds their row indices.

fuzzyclust.cluster.FCM

`n_clusters=3`, `m=2.0`, `norm="euclidean"`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`. `norm` is `"euclidean"` ($\mathbf{A} = \mathbf{I}$), `"diagonal"` (inverse feature variances), `"mahalanobis"` (pseudo-inverse of the data covariance), or an explicit (n, n) matrix; the matrix used is stored in `norm_matrix_`. Requires $m > 1$. `covariances_` holds the fuzzy covariance matrices of the final partition for inspection and plotting (they are not used by the algorithm).

fuzzyclust.cluster.GK

`n_clusters=3`, `m=2.0`, `rho=None`, `gamma=0.0`, `beta=1e15`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`. `rho` is a scalar or a vector of c volumes (`None`: all one); `gamma`$\in [0, 1]$ the shrinkage weight (3.11); `beta` the condition-number bound (3.12). Attributes: `covariances_` (c, n, n) after regularization, `norm_matrices_` (c, n, n), `eigenvalues_` (c, n) ascending, `eigenvectors_` (c, n, n) with the normal direction in the first column, `rho_`.

fuzzyclust.cluster.GG

`n_clusters=3`, `m=2.0`, `gamma=0.0`, `beta=1e15`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`. `m=1` gives the original FMLE algorithm, `m=2` is recommended; the covariance exponent is $w = m$. Pass `init=fcm.membership_` (or `gk.membership_`) for the recommended initialization. Attributes: `covariances_`, `priors_` $(c,)$, `eigenvalues_`, `eigenvectors_`; `distances_` and `transform` return $\sqrt{D_{ik}^2}$ capped at e^{350}.

fuzzyclust.cluster.RegularizedFCM

`n_clusters=3`, `regularization="entropy"`, `lam=1.0`, `annealing=False`, `lam_start=None`, `anneal_rate=0.8`, `perturbation=1e-3`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`. `regularization` is `"entropy"` (3.17) or `"quadratic"` (3.21); `lam` is λ in squared data units (the final temperature when annealing). `annealing=True` (entropy only) starts at `lam_start` (`None`: $2.2\lambda_{\max}$ of the data covariance) with all prototypes at the grand mean and cools by `anneal_rate`; `max_iter` then applies per temperature. Attributes: `lam_` and `covariances_`; with annealing `cluster_history_` (a `DataFrame` with columns `lam`, `n_effective`, and `objective`), `n_effective_clusters_`, and `critical_temperatures_`. The functions `entropy_memberships(d2, lam)` and `quadratic_memberships(d2, lam)` evaluate the membership formulas for given squared distances.

fuzzyclust.cluster.PolynomialFuzzifierFCM

`n_clusters=3`, `beta=0.5`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`. `beta`$\in [0, 1)$ is the fuzzifier parameter of (3.22). Attributes: `n_active_` (number of positive memberships per training sample), `covariances_` (weights $g(\mu_{ik})$). The functions `polynomial_fuzzifier(U, beta)` and `polynomial_memberships(d2, beta)` implement (3.22) and (3.23).

fuzzyclust.cluster.SubtractiveClustering

`radii=0.5`, `squash=1.25`, `accept_ratio=0.5`, `reject_ratio=0.15`, `max_clusters=None`. `radii` is r_a as a fraction of the range of every feature (a scalar or one value per feature). There is no `n_clusters` parameter; the number of clusters is a result, `n_clusters_`. Attributes: `cluster_centers_` (data points in original units), `center_indices_`, `potentials_` (relative potentials at selection), `initial_potentials_` (the density estimate), `membership_`, `labels_`. `predict_proba` returns normalized Gaussian memberships $\exp(-4D^2/r_a^2)$, `transform` the Euclidean distances to the centers in the original units.

fuzzyclust.cluster.MountainClustering

`grid_size=None`, `max_nodes=10000`, `radii=0.5`, `squash=1.25`, `stop_ratio=0.15`, `max_clusters=None`. `grid_size=None` chooses the largest grid with at most `max_nodes` nodes; an explicit grid that exceeds the bound raises an error. Attributes: `grid_`, `grid_size_`, `mountain_` (before destruc-

tion), `potentials_`, `n_clusters_`, `cluster_centers_`; inference methods as for `SubtractiveClustering`.

Fitting is one line, and the fitted object exposes prototypes, memberships, and labels:

```
X, y_true = make_fuzzy_blobs(n_clusters=3, n_samples_per_cluster=100,
    random_state=0)

fcm = FCM(n_clusters=3, m=2.0, random_state=0).fit(X)
```

The membership function between two prototypes is (3.6) in three lines of `numpy`:

```
ms = [1.1, 1.5, 2.0, 3.0, 5.0]
x = np.linspace(-1.0, 2.0, 601)
v = np.array([0.0, 1.0])
fig, ax = plt.subplots(figsize=(6.5, 4.2))
for j, m in enumerate(ms):
    d2 = (x[:, None] - v[None, :]) ** 2 + 1e-10  # squared distances to
    v_1, v_2
    mu = d2 ** (-1.0 / (m - 1.0))
    mu /= mu.sum(axis=1, keepdims=True)
    ax.plot(x, mu[:, 0], lw=2.0, color=PALETTE[j], label=f"m = {m}")
```

The recommended way to run the three fuzzy algorithms on the same data chains them:

```
fcm = FCM(n_clusters=3, random_state=0).fit(X)
gk  = GK(n_clusters=3, random_state=0, n_init=5).fit(X)
gg  = GG(n_clusters=3, init=fcm.membership_).fit(X)   # GG started from
    the FCM partition
print(gg.priors_)                                     # estimated cluster
    proportions
U_new = gg.predict_proba(X_new)                       # memberships of
    unseen data
```

The four fuzzification mechanisms are interchangeable estimators:

```
models = {
    "FCM (m = 2)": FCM(n_clusters=4, random_state=0, n_init=3),
    "entropy (lambda = 0.5)": RegularizedFCM(n_clusters=4,
    regularization="entropy", lam=0.5, random_state=0, n_init=3),
    "quadratic (lambda = 2)": RegularizedFCM(n_clusters=4,
    regularization="quadratic", lam=2.0, random_state=0, n_init=3),
    "polynomial (beta = 0.5)": PolynomialFuzzifierFCM(n_clusters=4,
    beta=0.5, random_state=0, n_init=3),
}
for name, model in models.items():
    model.fit(X)
    zeros = np.mean(model.membership_ == 0)
    ari = adjusted_rand_score(y, model.labels_)
```

```
    print(f"{name:24s} exact zeros = {zeros:5.1%}   ARI = {ari:.2f}
    iterations = {model.n_iter_}")
```

Deterministic annealing is a flag of `RegularizedFCM`, and the history of the phase transitions is a `DataFrame`:

```
da = RegularizedFCM(n_clusters=4, lam=0.5, annealing=True,
    anneal_rate=0.9, tol=1e-6, random_state=0).fit(X)
history = da.cluster_history_  # one row per temperature: lam,
    n_effective, objective
```

Subtractive clustering returns the number of clusters and the centers, which initialize any of the iterative algorithms:

```
sub = SubtractiveClustering(radii=0.25).fit(X)
print(f"{sub.n_clusters_} centers, relative potentials at selection:
    {np.round(sub.potentials_, 2)}")
mountain = MountainClustering(radii=0.25).fit(X)  # 100 x 100 grid nodes
print(f"mountain method: {mountain.n_clusters_} centers on a
    {mountain.grid_size_} x {mountain.grid_size_} grid")
```

```
gk = GK(n_clusters=sub.n_clusters_, init=sub.cluster_centers_).fit(X)
```

On product-space data the number of centers is the number of rules of a Takagi–Sugeno model:

```
Z = np.column_stack([X_train, y_train])  # product space of input and
    output
for r in (0.2, 0.3, 0.4):
    n_rules = SubtractiveClustering(radii=r).fit(Z).n_clusters_
    ts = TakagiSugeno(n_clusters=n_rules, random_state=0,
    n_init=3).fit(X_train, y_train)
    rmse = np.sqrt(np.mean((ts.predict(X_test) - y_test) ** 2))
    print(f"radius {r:.1f}: {n_rules:2d} rules, test RMSE = {rmse:.4f}")
```

3.4 Examples

3.4.1 Fuzzy c-means: memberships and prototypes

The script `plot_01_fcm_basics.py` fits FCM to three well-separated Gaussian clusters (fig. 1.3). It converges in six iterations, and the memberships printed for the least and the most ambiguous samples show the two faces of a fuzzy partition: samples at the prototypes have memberships $(0, 1, 0)$, while samples between two clusters share theirs, for example $(0.22, 0.74, 0.03)$. Figure 3.4 shows the contour maps for $m = 1.2$, 2, and 4. For $m = 1.2$ the regions of high membership are wide and the boundaries sharp (partition coefficient $PC = 1.00$); for $m = 4$ only the immediate neighborhood of a prototype has a high membership ($PC = 0.54$). On well-separated data the prototypes barely move with m.

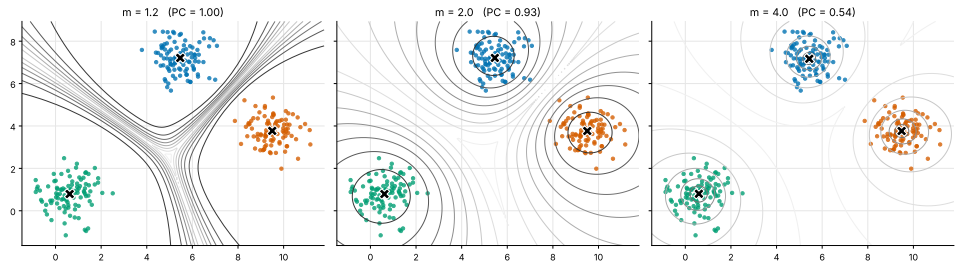


Figure 3.4. Fuzzy c-means on three separated clusters for $m = 1.2$, 2, and 4. The partition coefficient PC in the titles falls from 1.00 to 0.54: a small exponent gives wide regions of high membership and sharp borders, a large one nearly uniform memberships with the same prototypes.

3.4.2 The fuzziness exponent

The script `plot_09_fuzziness_exponent.py` isolates the exponent. Figure 3.5 plots (3.6) for two fixed prototypes on a line; for $m = 2$ it is the classical curve $\mu_1(x) = (x-1)^2/((x-1)^2 + x^2)$, and the return of every curve toward $1/2$ outside the prototypes is the mark of the column constraint. On three overlapping clusters (fig. 3.6) the same trend appears in the plane. The third figure of the script (not shown) records, for m from 1.1 to 10, how the partition coefficient falls from 0.98 to 0.34, close to $1/c$; how the mean distance of the prototypes from the grand mean, after a slight rise from 3.18 to 3.24 at $m = 1.75$–2, falls to 2.73 as they drift together; and how the number of iterations behaves. It is lowest, 14, near $m = 1.5$–1.75, and grows above $m \approx 2$ to 47 at $m = 10$ because the memberships change more slowly; at $m = 1.1$ it is 19, because the nearly crisp partition takes several iterations to settle.

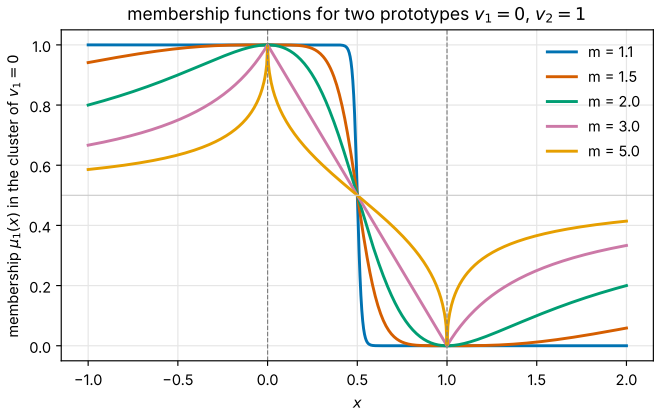


Figure 3.5. Membership in the cluster of $v_1 = 0$ along the line to $v_2 = 1$ for five exponents. $m \to 1$ gives the Voronoi step of k-means, large m a flat function at $1/2$; far from both prototypes every curve tends to $1/2$.

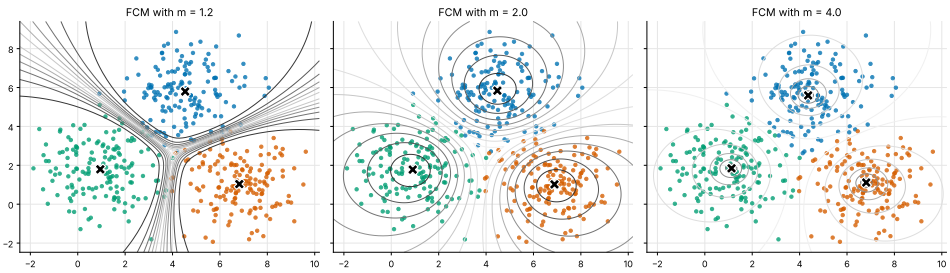


Figure 3.6. FCM on three overlapping clusters for $m = 1.2$, 2, and 4: as m grows, the partition becomes less sharp and the prototypes move slightly toward the grand mean.

3.4.3 Hard versus fuzzy partitions

The script `plot_02_hard_vs_fuzzy.py` runs k-means, k-medoids, and FCM on three overlapping clusters. The hard labels of the three methods are identical here (adjusted Rand index 0.975 with the generating labels for each of them), and the script verifies that the medoids are rows of the data matrix. Figure 3.7 shows what FCM adds: the mixed colors mark the overlap regions, and the confidence map identifies the samples that a crisp label would misrepresent; with $m = 2$, 3.9 % of the samples have a maximal membership below 0.6. The parameter study of the script shows how this share depends on the exponent while the labels stay put: it is 0.3 % for $m = 1.2$, 11.7 % for $m = 2.5$, and 61 % for $m = 4$, with the agreement with the generating labels between 0.97 and 0.98 throughout.

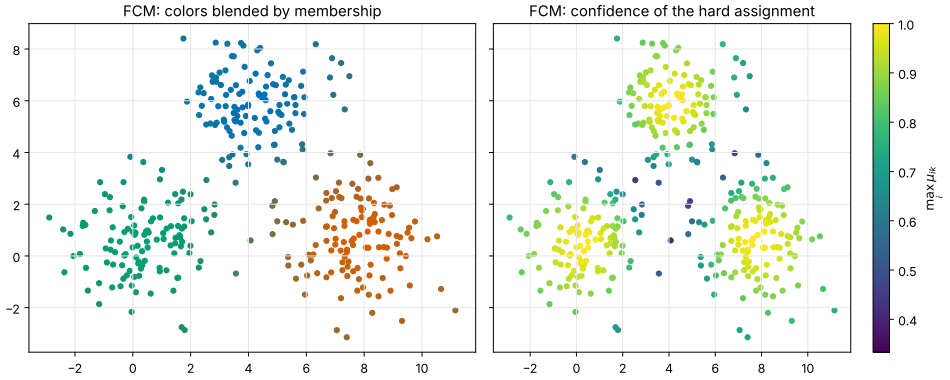


Figure 3.7. The information a fuzzy partition carries beyond the labels. Left: FCM memberships blended into the cluster colors; the overlap regions are mixed. Right: the maximal membership as the confidence of the hard assignment; the ambiguous samples lie between the prototypes.

3.4.4 Gustafson–Kessel: shape, volume, and regularization

The script `plot_03_gk_ellipses.py` generates three elongated clusters with different orientations. In the comparison of FCM and GK (fig. 3.8), the 2σ ellipses

of the fuzzy covariance matrices show what each algorithm "sees." The elongation ratios $\sqrt{\lambda_{i2}/\lambda_{i1}}$ read from `eigenvalues_` are 6.9, 8.7, and 8.4. Figure 3.9 varies the volume of the first cluster, $\rho_1 \in \{0.2, 1, 5\}$: its membership region shrinks or expands while the neighbors adjust, which is the lever to use when one cluster is known to be larger. Figure 3.10 adds a perfectly collinear fourth cluster. Without regularization the condition number of its fuzzy covariance matrix is about 6000, two orders of magnitude above those of the other clusters (50 to 80), and its membership region is a razor-thin line. With $\gamma = 0.1$ it obtains a finite width, but the shrinkage acts on every cluster and rounds the other ellipses as well; the bound $\beta = 10^4$ is not active, because no condition number reaches it. The parameter study of the script separates the two safeguards: $\beta = 10^2$ or 10^3 bounds the collinear cluster alone and keeps the adjusted Rand index at 0.93–0.94 (against 0.945 without regularization), whereas $\gamma \geq 0.05$ shortens every elongation ratio to below 3 and lowers the index to 0.81 and less.

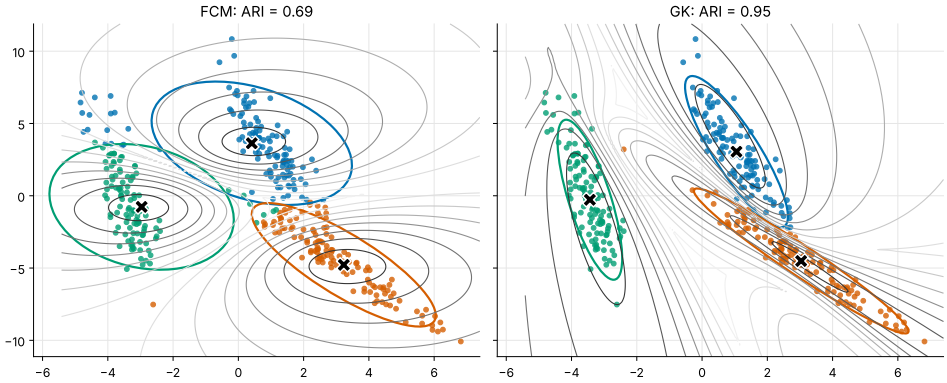


Figure 3.8. Three elongated clusters. FCM (left) draws circular contours that cut across the groups; GK (right) adapts an ellipse to each cluster and recovers the generating labels. Colored ellipses: 2σ ellipses of the fuzzy covariance matrices.

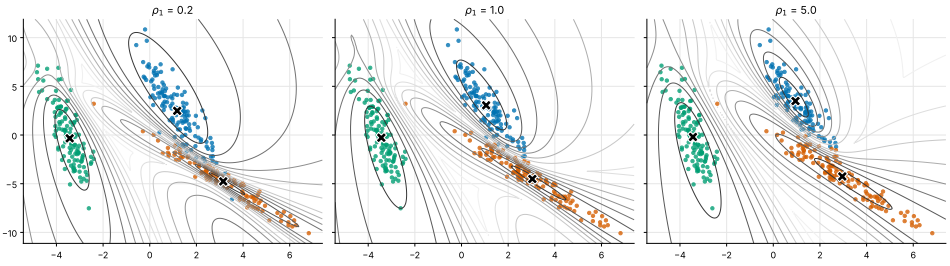


Figure 3.9. The volume parameter of GK. The first cluster is given $\rho_1 = 0.2$, 1, and 5; its region of high membership shrinks and grows accordingly while the partition of the well-separated data stays stable.

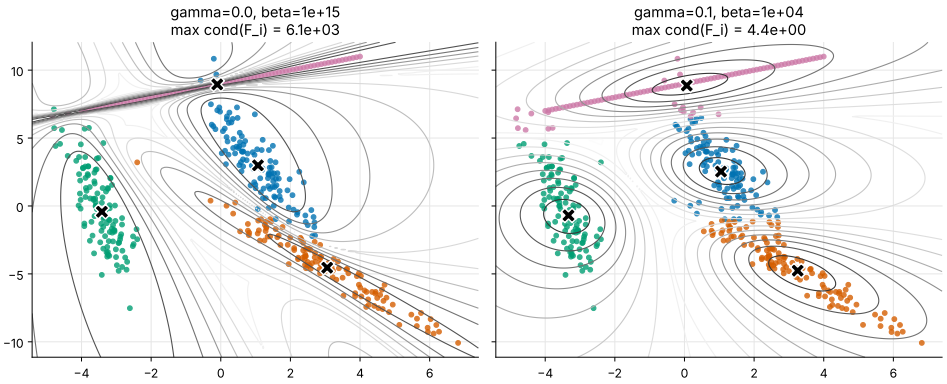


Figure 3.10. Covariance regularization on a perfectly collinear cluster. Left: without regularization the condition number of its covariance matrix is two orders of magnitude above those of the other clusters, and its membership region degenerates to a line. Right: the shrinkage $\gamma = 0.1$ gives the cluster a finite width but also rounds its neighbors; the condition bound $\beta = 10^4$ is not reached.

3.4.5 Local optima of Gustafson–Kessel

Elongated clusters are also where GK shows its start dependence. Section 2.4.3 reported the twenty random starts of `plot_10_initialisation_restarts.py` on three noisy line segments, together with the remedy (restarts) and the failure of k-means++ seeding on these data; fig. 3.11 shows what the two optima look like. The best run ($J = 142$) recovers the three lines. The worst ($J = 229$) fits one prototype along the diagonal line and splits the two upper lines into left and right halves, a configuration that is locally consistent with thin ellipsoids and from which the iteration cannot escape.

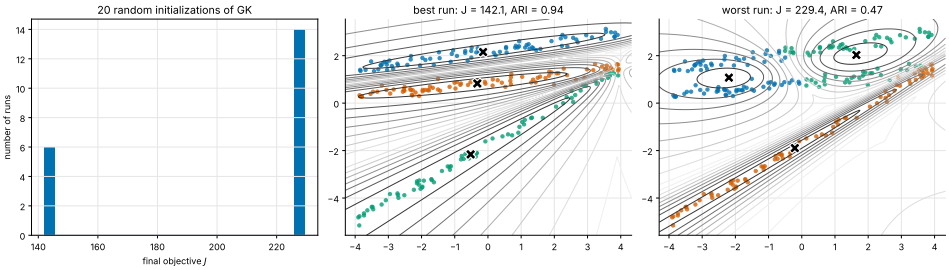


Figure 3.11. Twenty random starts of GK on three line segments. Left: the final objectives fall into two groups. Middle: the best run recovers the lines. Right: the worst run is a converged local optimum in which two lines are split into halves and a third prototype follows the diagonal.

3.4.6 Gath–Geva: size and density

The script `plot_04_gg_densities.py` generates a large diffuse cluster of 300 points next to two small dense ones of 60 points each. In fig. 3.12, FCM and GK, whose clusters have equal volume, let the two small clusters take over the margin of the big one (adjusted Rand indices 0.70 and 0.44), whereas GG, started from the FCM partition, gives the big cluster a large covariance and a large prior and leaves the small ones intact. The estimated priors $(0.14, 0.15, 0.71)$ match the true proportions $(0.71, 0.14, 0.14)$ up to the order of the clusters. The script also runs GG from ten random starts: the agreement with the generating labels varies between 0.83 and 0.98, and the FCM-initialized run matches the best of them without any restarts. Figure 3.13 displays the fitted components as the mixture density $\sum_i \alpha_i \, \mathcal{N}(\mathbf{x}; \mathbf{v}_i, \mathbf{F}_i)$, the reading of (3.13) as a Gaussian mixture.

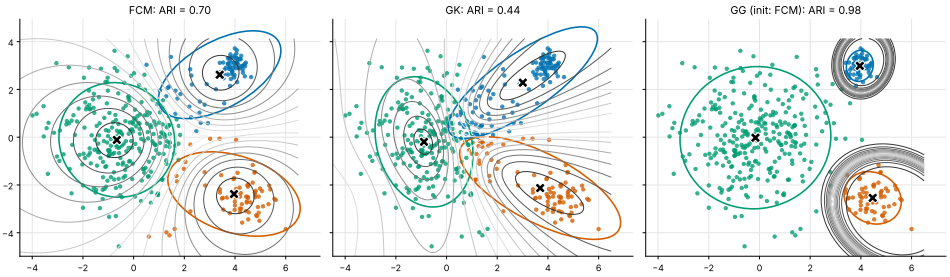


Figure 3.12. Clusters of very different size and density. FCM and GK give the margin of the large diffuse cluster to the small ones because their clusters have equal volume; GG, initialized from FCM, adapts volume and prior and recovers the structure.

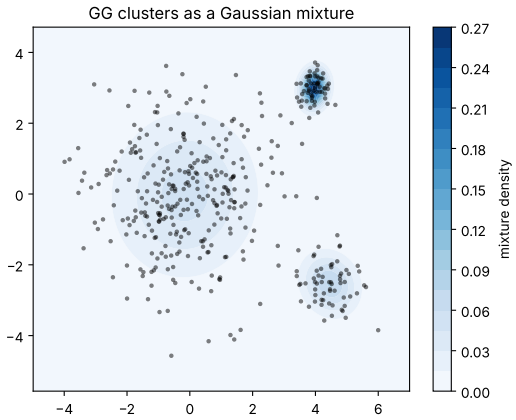


Figure 3.13. The GG result read as a Gaussian mixture: the density $\sum_i \alpha_i \mathcal{N}(\mathbf{x}; \mathbf{v}_i, \mathbf{F}_i)$ of the three fitted components over the data.

3.4.7 Evaluating new data

A fitted model defines a membership function over the whole space, not only a partition of the training data. The script `plot_06_predict_new_data.py` fits GK to 60 % of the samples of three elongated clusters and evaluates the rest with `predict_proba`, `predict`, and `transform`; the test samples fall into the regions of the training clusters (first figure of the script). Figure 3.14, the classical comparison figure of the toolbox manual drawn for these data, confirms the sketch of fig. 3.1. The script ends with a monitoring application: for a point far outside the data the maximal GK membership drops to 0.52 and its distance to the nearest prototype is 28, against a median of 1.2 for the training samples, so both quantities can serve as novelty scores. The same mechanism is used for validation in chapter 4.

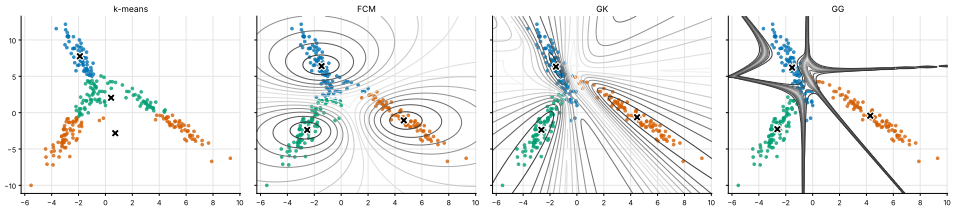


Figure 3.14. The membership function learned from the same training data by four algorithms: Voronoi cells (k-means), circles (FCM), ellipses (GK), and density-shaped regions with sharp borders (GG). Every algorithm defines its own distance, and `predict_proba` evaluates it for any new sample.

3.4.8 Real data: motorcycle and iris

Two scripts discussed in section 2.4 also belong here. On the standardized motorcycle data (`plot_07_motorcycle_scaling.py`), four GK clusters segment the acceleration signal into the phases before, during, and after the impact. In the scaled space (fig. 3.15) the orientation of each cluster is the local slope of the signal, the first appearance of the local-linear-model reading of GK that chapters 8 and 9 develop. On the iris data (`plot_08_sklearn_pandas_workflow.py`), GK with three clusters separates setosa cleanly and divides versicolor and virginica with the overlap reported in section 2.4.4; the misclassification rate of about 10 % agrees with the figures reported for the MATLAB toolbox.

3.4.9 Fuzzifiers, regularization, and deterministic annealing

The script `plot_12_fuzzifiers_annealing.py` puts the mechanisms of section 3.2.7 side by side. Figure 3.16 evaluates the membership formulas for three prototypes on a line at 0, 1, and 4. The FCM memberships are positive everywhere: the third cluster keeps a small membership even between the first two prototypes (about 0.01 at their midpoint) and 0.07 at $x = -2$. The entropy memberships are softmax functions that become crisp a short distance from every border. The

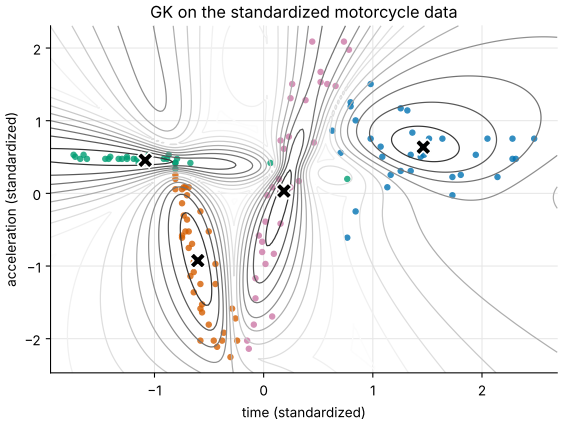


Figure 3.15. Gustafson–Kessel contours on the standardized motorcycle data. Each ellipse follows the local slope of the acceleration signal: the covariance of a GK cluster encodes a local linear model.

quadratic and polynomial memberships are exactly zero for at least one cluster almost everywhere, and between v_1 and v_2 the third cluster has no influence at all. On four groups in the plane (fig. 3.17) all four mechanisms recover the groups (adjusted Rand index 0.98), but the membership matrices differ: the quadratic regularization ($\lambda = 2$) gives 71 % and the polynomial fuzzifier ($\beta = 0.5$) 74 % exact zeros, close to the maximum $(c-1)/c = 75\,\%$, while FCM and the entropy regularization give none. In the contours this appears as flat plateaus of membership one inside the groups for the two sparse mechanisms and as ripples around every prototype for FCM.

Deterministic annealing on the same data (fig. 3.18) starts at $\lambda = 35.4$, just above the critical temperature $2\lambda_{\max}(\mathbf{F}_0) = 32.2$ of the whole data set, and cools by the factor 0.9. The four prototypes stay together until $\lambda = 28.7$, where they split into two pairs (the left and the right halves of the data), and the pairs split at $\lambda = 3.14$, just below the critical temperatures 3.27 and 3.53 of the two halves predicted by (3.19). The final partition at $\lambda = 0.5$ recovers the four groups with an adjusted Rand index of 0.98, without any random initialization. The parameter study (fig. 3.19) shows the same transition from the other side: at a fixed temperature, the partition coefficient of the entropy-regularized partition stays near one up to $\lambda = 1$ (0.97) and drops to $1/2$ for $\lambda \geq 5$ (0.500 and 0.498), because above the critical temperature of the halves the two prototypes of each half coincide and the four prototypes describe two clusters. The polynomial fuzzifier produces exact zeros as soon as $\beta > 0$ (70 % already at $\beta = 0.1$); a larger β only sharpens the borders, and the partition coefficient rises from 0.87 ($\beta = 0$, FCM) to 0.99 at $\beta = 0.5$.

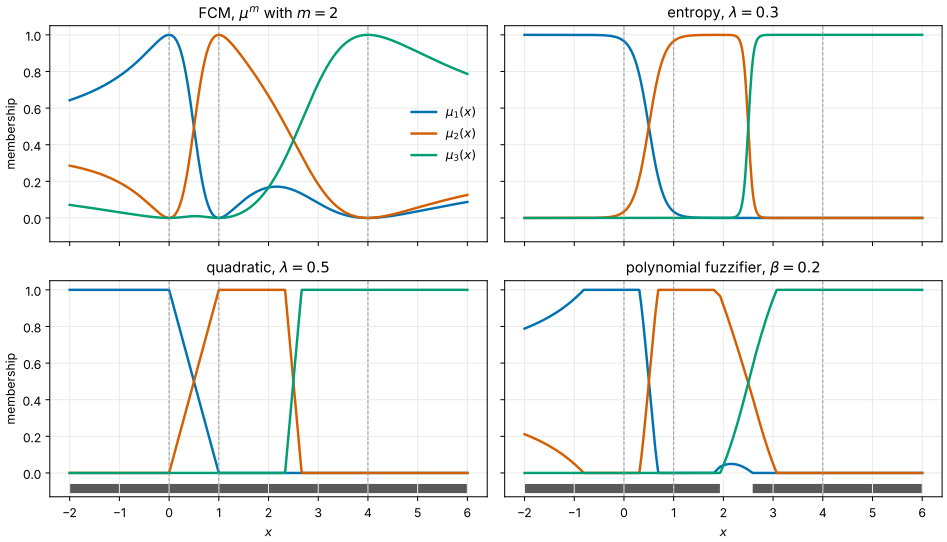


Figure 3.16. Memberships of three clusters with prototypes at 0, 1, and 4 on a line under four fuzzification mechanisms. The dark bar below the curves marks where at least one membership is exactly zero: nowhere for FCM and the entropy regularization, almost everywhere for the quadratic regularization and the polynomial fuzzifier.

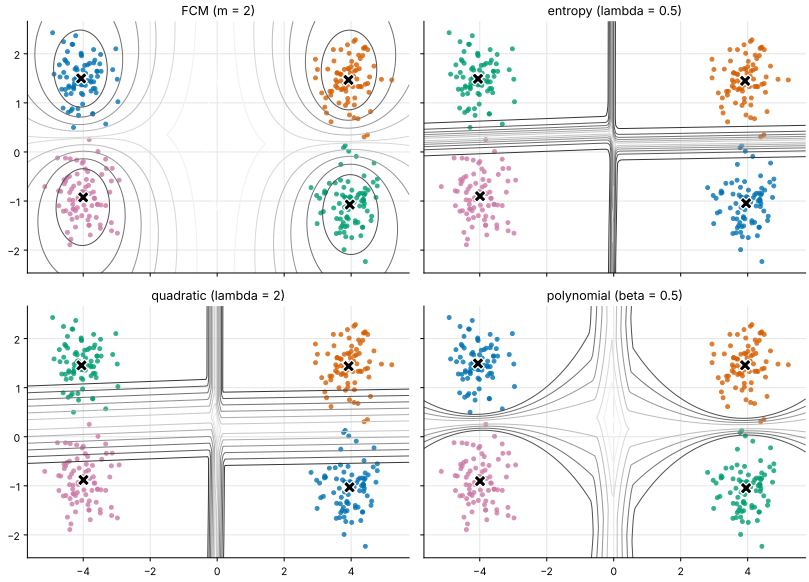


Figure 3.17. Maximal membership on four groups for FCM, entropy and quadratic regularization, and the polynomial fuzzifier: the sparse mechanisms are crisp inside the groups and fuzzy only between neighbors.

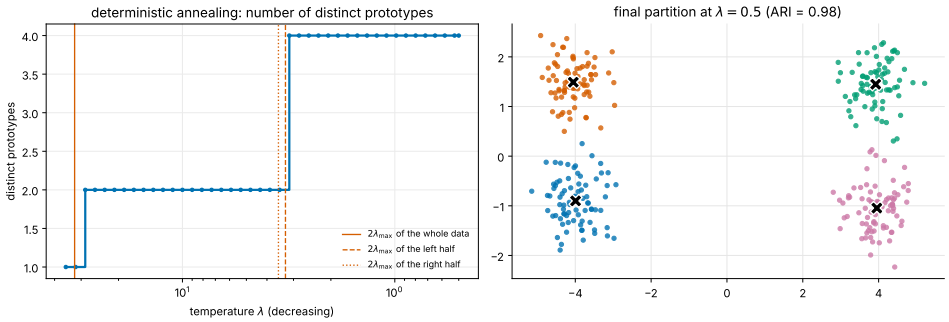


Figure 3.18. Deterministic annealing on four groups. Left: the number of distinct prototypes against the decreasing temperature jumps from one to two near the critical temperature of the whole data and from two to four near those of the two halves (vertical lines, $2\lambda_{\max}$ of the covariances). Right: the final partition.

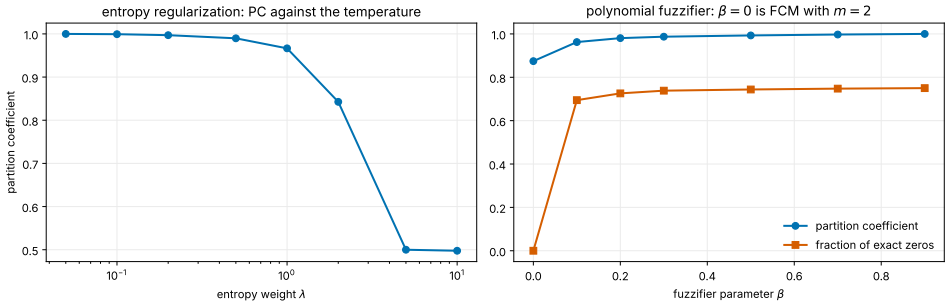


Figure 3.19. Parameter study: partition coefficient of the entropy regularization against λ (left; the drop to $1/2$ is the merging of the prototype pairs above the critical temperature) and partition coefficient and fraction of exact zeros of the polynomial fuzzifier against β (right).

3.4.10 Subtractive clustering: number of clusters, initialization, and rules

The script `plot_13_subtractive_clustering.py` generates eight elongated groups with 40 to 150 samples each. With $r_a = 0.25$, subtractive clustering selects eight centers, one per group, in the order of decreasing potential: the relative potentials at selection fall from 1 and 0.97 for the two large groups to 0.26 for the last small one (fig. 3.20, left). The mountain method on a 100×100 grid finds the same eight centers (right panel); its mountain function is visibly the same density estimate as the potential of the samples. Passed to GK as `init`, the subtractive centers lead to the optimum ($J = 353.9$, adjusted Rand index 0.97) in 33 iterations, against a median of 44 and a maximum of 126 for twenty random starts that all reach the same optimum (second figure of the script).

The radius study (fig. 3.21) shows how the method is used in practice. The number of centers falls with the radius and, with the default squash factor, stays

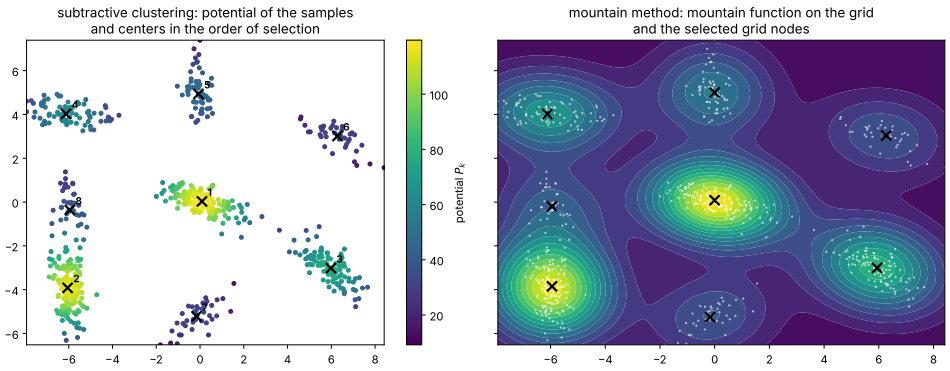


Figure 3.20. Subtractive clustering and the mountain method on eight elongated groups. Left: the potential of every sample and the centers numbered in the order of selection. Right: the mountain function on a 100×100 grid and the selected nodes.

at eight for $r_a \in [0.18, 0.30]$: below the plateau the elongated groups are cut into pieces, above it neighboring groups fall within one radius. A larger squash factor shifts the plateau to smaller radii ($[0.14, 0.24]$ for $\eta = 1.5$, $[0.22, 0.38]$ for $\eta = 1$), because the subtracted bump has the radius ηr_a; the mountain method follows the subtractive curve closely. The last part of the script uses subtractive clustering as Chiu did, as a rule-number estimator for a Takagi–Sugeno model of $\sin(x)/x$ (fig. 3.22). Run on the product space of input and output, it suggests 12, 10, 6, 4, and 3 rules for $r_a = 0.15$, 0.2, 0.3, 0.4, and 0.5. The test error, computed for every number of rules, decreases steeply up to six rules (RMSE $= 0.039$) and flattens beyond ten (0.024); the radius 0.3 thus gives the smallest accurate model and 0.2 the accurate end of the curve, while 0.5 is too coarse (RMSE $= 0.13$ with three rules).

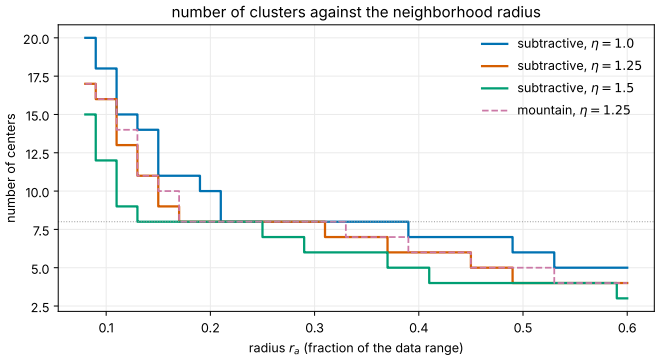


Figure 3.21. Number of centers against the radius for three squash factors and for the mountain method. The plateau at eight (dotted line) indicates the stable structure.

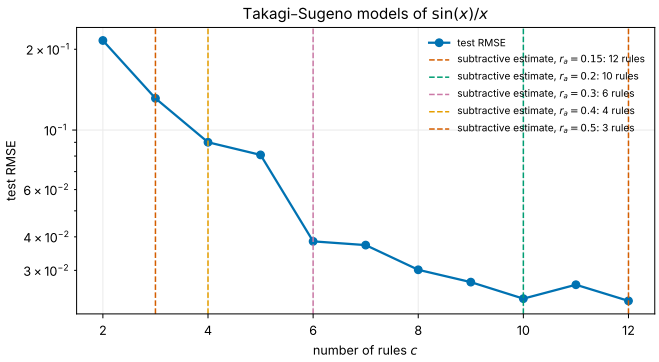


Figure 3.22. Test error of Takagi–Sugeno models of $\sin(x)/x$ against the number of rules, with the rule numbers estimated by subtractive clustering in the input–output space for five radii (dashed lines).

3.5 Further reading

k-means goes back to MacQueen [17], k-medoids to Kaufman and Rousseeuw [5]; k-means++ seeding is due to Arthur and Vassilvitskii [15]. Fuzzy c-means was introduced by Dunn [2] and generalized and analyzed by Bezdek and co-workers [3, 10, 11, 16]. The Gustafson–Kessel algorithm appeared in [18], its covariance regularization in Babuška, van der Veen, and Kaymak [19]; the use of the eigenstructure of $\mathbf{F}_i$ for local linear models is developed in Babuška's monograph [7]. Fuzzy maximum likelihood estimation is due to Bezdek and Dunn [20] and the Gath–Geva algorithm to Gath and Geva [21], whose paper also introduced the volume-based validity measures of the next chapter. The exposition of the four algorithms follows the MATLAB toolbox manual [12] and the same monograph [7]; Höppner et al. [6] discuss all of them with their shell and variety relatives. The entropy regularization is due to Li and Mukaidono [24] and Miyamoto and Mukaidono [23], the quadratic one to Miyamoto and Umayahara [26]; Rose [25] surveys deterministic annealing with its phase transitions and its applications beyond clustering. Klawonn and Höppner [22] analyze what makes a fuzzifier and propose the polynomial one. The mountain method is from Yager and Filev [27], subtractive clustering and its use for fuzzy model identification from Chiu [28].

CHAPTER

4



Cluster validity and the number of clusters

The algorithms of the previous chapter return a partition for whatever number of clusters they are given, and they return it with the same confidence whether or not the data contain that many groups. This chapter is about judging a partition and choosing the number of clusters. It distinguishes three routes—scanning c with validity indices, merging compatible clusters, and asking which partition is stable under resampling—and describes every index in `fuzzyclust.validity`: partition-based indices that look only at the memberships, compactness and separation indices that relate the within-cluster scatter to the distances between prototypes, crisp indices evaluated on the hardened partition, and the volume and density indices of Gath and Geva. Compatible cluster merging is given as an algorithm with a geometric and a similarity criterion, and the fuzzy Rand and adjusted Rand indices, which compare two fuzzy partitions, are the basis of stability selection. The practical part runs the three routes on the same data and shows how to combine indices and how evaluating new data with `predict_proba` is itself a form of validation; visual validation is the subject of chapter 5.

4.1 Problem and idea

Cluster validity is the question of whether a given partition fits the data at all. A clustering algorithm finds the best fit for a fixed number of clusters and a fixed family of cluster shapes, but the best fit need not be meaningful: the number of clusters may be wrong, the assumed shape may not correspond to the groups in the data, or the data may have no group structure. Since clustering is unsupervised, there are no labels to compare with; the quality of a partition must be read from

the partition and the data themselves. There are three routes to the number of clusters; the first two are sketched in fig. 4.1.

- *Validity scans.* The algorithm is run for every $c \in \{2, \dots, c_{\max}\}$, a *validity index* is computed for every partition, and the c at which the index is extremal (or has a knee) is selected. The index may evaluate the whole partition—the usual case, and the one treated here—or the individual clusters, in which case poor clusters are removed and similar ones joined.
- *Compatible cluster merging.* The algorithm starts with a deliberately large $c_{\max}$, and the number of clusters is reduced step by step by merging pairs of clusters that are compatible according to some criterion, re-clustering after every merge, until no compatible pair remains.
- *Stability selection.* The data are resampled, the algorithm is run on every sample for every candidate c, and the c whose partitions agree best across the samples is selected; agreement is measured by a fuzzy generalization of the Rand index.

The scan is general and simple but costs $c_{\max} - 1$ complete runs, and its outcome rests on the index. Merging exploits the geometry of the prototypes and is natural when the clusters stand for local models, as in chapters 8 and 9, or when they are nodes of a tree, as in chapter 7; it needs a compatibility criterion with thresholds. Stability selection makes no assumption about what a good cluster looks like beyond the algorithm's own, but it is the most expensive route, with two fits per resample and per c.

Figure 4.2 shows the two quantities from which almost every geometric index is built. *Compactness* is the scatter of the samples about their prototypes, $\sum_k \mu_{ik}^m \|\mathbf{x}_k - \mathbf{v}_i\|^2$ summed over clusters; it is small when the clusters are tight. *Separation* is the distance between the prototypes, $\|\mathbf{v}_i - \mathbf{v}_j\|^2$, aggregated in some way (the minimum over pairs, the sum, or the distance to the grand mean); it is large when the clusters are far apart. A good partition has small compactness and large separation, so most indices are a ratio or a difference of the two. The partition-based indices use only the memberships and ask how fuzzy the partition is, on the grounds that a wrong number of clusters shows up as ambiguous memberships; the volume-based indices of Gath and Geva measure the volume of the clusters through their covariance matrices. No index is perfect, which is why the toolbox implements sixteen and the recommended practice is to look at several.

4.2 Theory

Throughout, $\mathbf{U} \in \mathbb{R}^{c \times N}$ is the partition, $\mathbf{V}$ the prototypes, m the exponent the algorithm used, $N_i = \sum_k \mu_{ik}$ the fuzzy cardinality of cluster i, and $\bar{\mathbf{x}}$ the grand mean of the data. Apart from the volume indices, which use the fuzzy covariance matrices, all distances in the indices are Euclidean. This is the convention of the validity literature and makes the indices comparable across algorithms: the norm

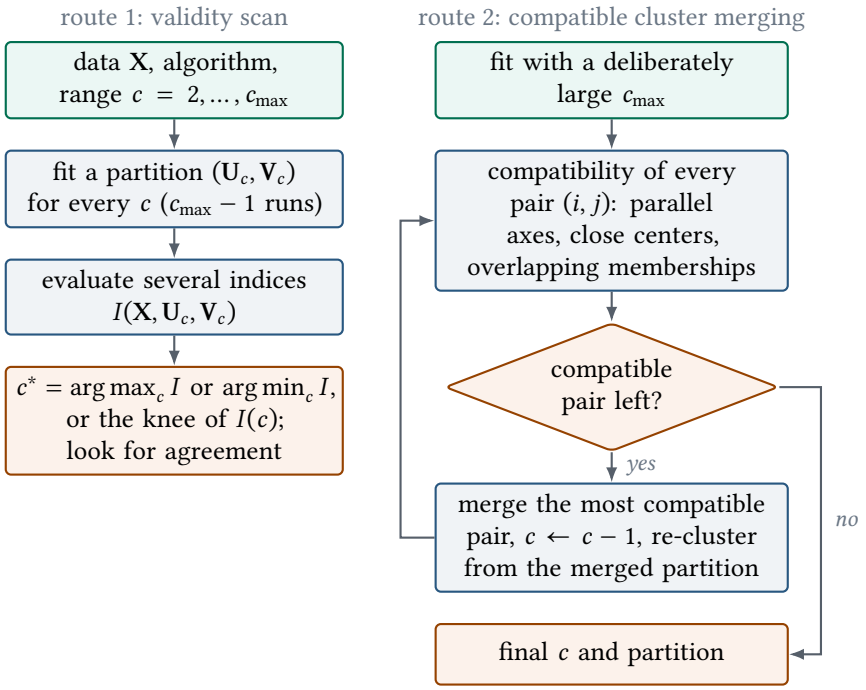


Figure 4.1. Two routes to the number of clusters. Left: the algorithm is run for every candidate c, and validity indices computed from the data, the partition, and the prototypes select the best one. Right: a deliberately large number of clusters is reduced by merging pairs that are compatible—parallel principal axes, close centers, similar memberships—and re-clustering after every merge. The toolbox implements the scan in `fuzzyclust.validity` and merging in `fuzzyclust.cluster.CompatibleClusterMerging`; merging also appears in the graph-based methods of chapter 7 and the time-series segmentation of chapter 9. A third route, stability selection, is described in section 4.2.9.

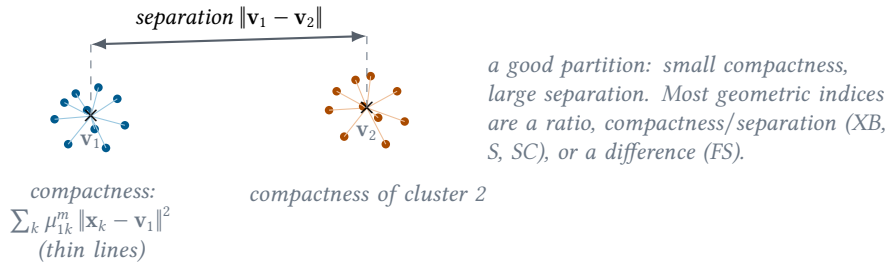


Figure 4.2. The two ingredients of the geometric validity indices: compactness, the membership-weighted scatter of the samples about their prototypes (thin lines), and separation, the distance between the prototypes (arrow).

of the algorithm is *not* used, so GK and GG partitions are judged in the same geometry as FCM partitions. We present the indices in four families (sections 4.2.1 to 4.2.4), add two further compactness and separation indices (section 4.2.5), and summarize them in section 4.2.6. The scan and merging procedures (section 4.2.7), the comparison of two fuzzy partitions (section 4.2.8), and the stability-based selection of the number of clusters (section 4.2.9) follow.

4.2.1 Partition-based indices

The *partition coefficient* of Bezdek [3],

$$PC = \frac{1}{N}\sum_{i=1}^{c}\sum_{k=1}^{N}\mu_{ik}^2, \qquad \tfrac{1}{c} \le PC \le 1, \tag{4.1}$$

measures the amount of overlap between the clusters: it is one for a hard partition and $1/c$ for the fuzziest one, and a larger value is better. The *classification entropy*

$$CE = -\frac{1}{N}\sum_{i=1}^{c}\sum_{k=1}^{N}\mu_{ik}\log\mu_{ik}, \qquad 0 \le CE \le \log c, \tag{4.2}$$

measures the same fuzziness with Shannon's entropy: it is zero for a hard partition and $\log c$ for the fuzziest one; lower is better. Both need nothing but $\mathbf{U}$, which is their appeal and their weakness: they have no direct connection to the geometry of the data, and they tend to change monotonically with c—PC decreases and CE increases even for data with clear groups, because more clusters mean more borders and more ambiguous samples. On well-separated data they still have a genuine extremum at the right c (section 4.4.1); on overlapping data the *knee* of the curve, not its extremum, is the informative point, which is how the MATLAB toolbox manual read its motorcycle scan. They also depend strongly on m, so they cannot compare partitions obtained with different exponents, and they are useless for hard partitions ($PC = 1$, $CE = 0$ always). Davé's *modified partition coefficient* [29],

$$MPC = 1 - \frac{c}{c-1}(1 - PC), \qquad 0 \le MPC \le 1, \tag{4.3}$$

rescales PC from its range $[1/c, 1]$ to $[0, 1]$ and thereby reduces the monotone trend; higher is better.

4.2.2 Compactness and separation indices

The *partition index* of Bensaid et al. [30],

$$SC = \sum_{i=1}^{c}\frac{\sum_{k=1}^{N}\mu_{ik}^m\left\|\mathbf{x}_k - \mathbf{v}_i\right\|^2}{N_i\sum_{j=1}^{c}\left\|\mathbf{v}_j - \mathbf{v}_i\right\|^2}, \tag{4.4}$$

is a sum of cluster-wise validities, the compactness of cluster i over its separation from all other prototypes, each normalized by the fuzzy cardinality N_i so that a

small cluster is not favored merely for being small. Lower is better. Its authors designed it for comparing different partitions with the *same* number of clusters, e.g. from different algorithms or initializations; across c it tends to decrease. The *separation index*, which the MATLAB manual attributes to the same paper, is the Xie–Beni index in its original form [31], with squared memberships,

$$S = \frac{\sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^{2} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2}{N \min_{i \neq j} \left\| \mathbf{v}_i - \mathbf{v}_j \right\|^2}. \tag{4.5}$$

It uses the minimum distance between prototypes as separation, which makes it sensitive to any two prototypes that come close; lower is better. The *Xie–Beni index* of the toolbox weights the scatter with the exponent m of the algorithm instead of the square, the form analyzed by Pal and Bezdek [32],

$$XB = \frac{\sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^{m} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2}{N \min_{i \neq j} \left\| \mathbf{v}_i - \mathbf{v}_j \right\|^2}. \tag{4.6}$$

It is the compactness (J/N for Euclidean FCM) over the minimal separation; for $m = 2$ it coincides with S, and SC, S, and XB differ mainly in how the separation is aggregated. XB is the most widely used fuzzy validity index. Lower is better, and it is usually inspected for the first pronounced minimum, because it decreases again for large c, when the minimal center distance shrinks only slowly while the compactness keeps improving. Pal and Bezdek showed that the index is reliable for m between about 1.5 and 2.5. The *Fukuyama–Sugeno index* [33],

$$FS = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^{m} \Big(\left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 - \left\| \mathbf{v}_i - \bar{\mathbf{x}} \right\|^2 \Big), \tag{4.7}$$

is compactness minus separation, where the separation of cluster i is its distance from the grand mean weighted by its fuzzy mass; lower (more negative) is better. Because it is a difference rather than a ratio, its value depends on the scale of the data, and it tends to prefer small c on data with one dominant group. The *PBM-F index* of Pakhira, Bandyopadhyay, and Maulik [34],

$$PBMF = \left(\frac{1}{c} \cdot \frac{E_1}{J_m} \cdot D_c \right)^2, \tag{4.8}$$

$$\text{with} \quad E_1 = \sum_{k=1}^{N} \left\| \mathbf{x}_k - \bar{\mathbf{x}} \right\|, \quad J_m = \sum_{i,k} \mu_{ik}^{m} \left\| \mathbf{x}_k - \mathbf{v}_i \right\|, \quad D_c = \max_{i,j} \left\| \mathbf{v}_i - \mathbf{v}_j \right\|,$$

combines the ratio of the scatter about the grand mean to the fuzzy within-cluster scatter (both with unsquared distances) with the *maximal* distance between prototypes and penalizes c explicitly by the factor $1/c$; higher is better. The maximal separation makes it insensitive to one pair of close prototypes but also lets it reward a single far-away cluster.

4.2.3 Crisp indices on the hardened partition

Three indices were designed for hard partitions and are evaluated on the hardened partition $\arg\max_i \mu_{ik}$. *Dunn's index* [35],

$$DI = \min_i \min_{j \neq i} \left\{ \frac{\min_{\mathbf{x} \in C_i,\, \mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y})}{\max_l \max_{\mathbf{x}, \mathbf{y} \in C_l} d(\mathbf{x}, \mathbf{y})} \right\}, \tag{4.9}$$

is the smallest distance between two clusters (single linkage) over the largest cluster diameter; higher is better. The index was proposed to identify "compact and well separated" clusters. Its drawbacks are the cost—all $N(N-1)/2$ pairwise distances are needed, so it is $\mathcal{O}(N^2)$—and its sensitivity: a single pair of close samples from different clusters or a single outlier that inflates a diameter changes the value completely, and for overlapping clusters the hardened partition has no meaningful inter-cluster distance at all. The *alternative Dunn index* of the MATLAB toolbox bounds the inter-cluster distance from below by the triangle inequality, $d(\mathbf{x}, \mathbf{y}) \geq \left| d(\mathbf{y}, \mathbf{v}_j) - d(\mathbf{x}, \mathbf{v}_j) \right|$, which needs only the distances of the samples to the prototypes:

$$ADI = \min_i \min_{j \neq i} \left\{ \frac{\min_{\mathbf{x} \in C_i,\, \mathbf{y} \in C_j} \left| d(\mathbf{y}, \mathbf{v}_j) - d(\mathbf{x}, \mathbf{v}_j) \right|}{\max_l \max_{\mathbf{x}, \mathbf{y} \in C_l} d(\mathbf{x}, \mathbf{y})} \right\}. \tag{4.10}$$

Higher is better; the bound is loose (two samples at the same distance from $\mathbf{v}_j$ but on opposite sides give zero), so *ADI* is often tiny, and the MATLAB manual itself noted that it "is not tested enough to know how reliable its results are". The *silhouette* of Rousseeuw [36] compares, for every sample, its mean distance a_k to the members of its own cluster with the smallest mean distance b_k to the members of another cluster,

$$SIL = \frac{1}{N} \sum_{k=1}^{N} \frac{b_k - a_k}{\max(a_k, b_k)} \in [-1, 1], \tag{4.11}$$

higher is better. It is the least sensitive of the three crisp indices because it averages over all samples, but it is also $\mathcal{O}(N^2)$ and, like all crisp indices, ignores the memberships. The toolbox computes it with `sklearn.metrics.silhouette_score`.

4.2.4 Volume and density indices

Gath and Geva [21] introduced three measures that use the fuzzy covariance matrices $\mathbf{F}_i$ of (3.10) and therefore fit the ellipsoidal clusters of GK and GG. The *fuzzy hypervolume*

$$FHV = \sum_{i=1}^{c} \sqrt{\det \mathbf{F}_i} \tag{4.12}$$

is the total volume of the clusters; compact clusters give small values, and lower is better. It is the natural companion of the GG algorithm and serves as the splitting criterion of the MST clustering in chapter 7. The *partition density*

$$PD = \frac{\sum_{i=1}^{c} S_i}{FHV}, \qquad S_i = \sum_{k:\,(\mathbf{x}_k-\mathbf{v}_i)^{\mathsf{T}}\mathbf{F}_i^{-1}(\mathbf{x}_k-\mathbf{v}_i)<1} \mu_{ik}, \tag{4.13}$$

is the membership mass inside the unit Mahalanobis ellipsoids of the clusters per unit of volume—how densely the "central" members fill the clusters; higher is better. The *average partition density*

$$APD = \frac{1}{c}\sum_{i=1}^{c} \frac{S_i}{\sqrt{\det \mathbf{F}_i}} \tag{4.14}$$

averages the density cluster by cluster, so that every cluster counts equally whatever its size; higher is better. A single small and very dense cluster can therefore dominate APD, whereas PD is dominated by the large clusters. The three indices reward tight, well-filled ellipsoids irrespective of their orientation, which the Euclidean compactness indices cannot do. Their weak point is the determinant, which collapses for nearly singular covariance matrices (a cluster of few points, a collinear cluster) and then makes FHV tiny and PD huge for a degenerate partition. They are computed from the fuzzy (not the hardened) partition, with $\mathbf{F}_i$ weighted by μ_{ik}^m, whatever algorithm produced $\mathbf{U}$.

4.2.5 Two further compactness and separation indices

The Xie–Beni index decreases again for large c because its separation, the minimal distance between prototypes, shrinks more slowly than the compactness improves. Kwon [37] removed this tendency with a *punishing term* in the numerator,

$$KWON = \frac{\sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^2 \left\|\mathbf{x}_k - \mathbf{v}_i\right\|^2 + \frac{1}{c}\sum_{i=1}^{c} \left\|\mathbf{v}_i - \bar{\mathbf{x}}\right\|^2}{\min_{i\neq j} \left\|\mathbf{v}_i - \mathbf{v}_j\right\|^2}, \tag{4.15}$$

the mean squared distance of the prototypes from the grand mean, which grows as prototypes are added toward the periphery of the data; lower is better. Since the numerator is not divided by N, the index equals $N \cdot S$ plus the punishing term divided by the minimal separation; it inherits the reaction of XB and S to a close pair of prototypes but not their decrease at large c. Wu and Yang [38] combined a cluster-wise partition coefficient with an *exponential* separation in the *partition coefficient and exponential separation* index,

$$PCAES = \sum_{i=1}^{c} \frac{\sum_{k=1}^{N} \mu_{ik}^2}{\mu_M} - \sum_{i=1}^{c} \exp\!\left(-\frac{\min_{j\neq i} \left\|\mathbf{v}_i - \mathbf{v}_j\right\|^2}{\beta_T}\right), \tag{4.16}$$

$$\text{with} \quad \mu_M = \min_{1\le i\le c} \sum_{k=1}^{N} \mu_{ik}^2, \qquad \beta_T = \frac{1}{c}\sum_{i=1}^{c} \left\|\mathbf{v}_i - \bar{\mathbf{x}}\right\|^2;$$

higher is better. The first sum rates the compactness of every cluster relative to the least compact one; the second charges every cluster up to one unit for a neighbor closer than the typical spread $\sqrt{\beta_T}$ of the prototypes. The index thus judges clusters individually rather than through one global ratio. Since the first sum is at least c and the penalty at most c, the index is nonnegative. Its weak point is the normalization by the *smallest* cluster mass μ_M: a spurious cluster with very few members inflates every ratio of the first sum, and the index then favors a large c (section 4.4.2), so its curve should be read, not only its maximum.

4.2.6 Summary of the indices

Table 4.1 collects the sixteen indices. Three practical observations help in reading a scan. First, PC, CE, and MPC answer a different question from the geometric indices: they ask whether the partition is *confident*, the others whether it is *good*; a wrong number of clusters usually spoils both, but a crisp partition of a bad c is possible (small m) and so is a fuzzy partition of the right one (overlapping groups). Second, the ratio indices XB and S react to the *closest* pair of prototypes, so they jump when a scan first places two prototypes into one group; this is what makes their first pronounced minimum informative and their behavior at large c erratic. Third, the crisp indices are the ones to trust least on overlapping data, and the volume indices the ones to trust most on well-separated ellipsoidal clusters.

4.2.7 Scanning the number of clusters and merging clusters

The scan is straightforward (algorithm 4.1): clone the estimator, set c, fit, evaluate the indices, and tabulate. Two details matter. First, the algorithm should be run with the same settings for every c—same m, same tolerance, same initialization strategy—and with enough restarts that a poor local optimum at one c does not masquerade as a bad c. Second, the suggested optimum of every index should be read together with its curve: an index that decreases monotonically has its extremum at $c_{\max}$ but may point elsewhere with its knee, and an index whose curve is flat over a range of c says that the data do not decide. The plotting helper `plot_validity_scan` min–max scales every index to $[0, 1]$ so that all curves fit one axis and their agreement is visible at a glance.

Compatible cluster merging turns the second route of fig. 4.1 into an algorithm (algorithm 4.2). It was introduced by Krishnapuram and Freg [39] for fitting an unknown number of lines and planes and adapted to fuzzy modeling by Kaymak and Babuška [40]. The partition with $c_{\max}$ clusters is computed, every pair of clusters is judged by a compatibility criterion, the most compatible pair is merged, and the algorithm is re-run from the merged partition with one cluster fewer, until no pair passes the thresholds. Merging adds the two membership columns, $\boldsymbol{\mu}_{(ij)} = \boldsymbol{\mu}_i + \boldsymbol{\mu}_j$; the first iteration of the refit then computes the membership-weighted center and covariance of the union, which is what Kelly's merging formulas [41] give in closed form. Two families of criteria are implemented.

Table 4.1. Validity indices of `fuzzyclust.validity`. "Opt." is the direction of the optimum; the last column names the main weakness. The function names are listed in section 4.3.

Index	Opt.	Measures	Weakness
PC	max	fuzziness of $\mathbf{U}$ only, eq. (4.1)	monotone in c, no link to the data, depends on m
CE	min	entropy of $\mathbf{U}$, eq. (4.2)	same as PC
MPC	max	PC rescaled to $[0, 1]$, eq. (4.3)	no link to the data
SC	min	compactness / summed separation per cluster, eq. (4.4)	decreases with c; meant for equal c
S	min	compactness (μ^2) / minimal separation, eq. (4.5)	sensitive to one close pair of prototypes
XB	min	compactness (μ^m) / minimal separation, eq. (4.6)	decreases again for large c; use the first minimum
FS	min	compactness − separation from the grand mean, eq. (4.7)	scale-dependent; favors small c
DI	max	min inter-cluster distance / max diameter (crisp), eq. (4.9)	$\mathcal{O}(N^2)$; unreliable for overlapping clusters, outliers
ADI	max	DI with a triangle-inequality bound, eq. (4.10)	loose bound, often near zero
SIL	max	mean silhouette of the hardened partition, eq. (4.11)	$\mathcal{O}(N^2)$; ignores memberships
FHV	min	total cluster volume $\sum_i \sqrt{\det \mathbf{F}_i}$, eq. (4.12)	collapses for degenerate covariances
PD	max	central membership mass per volume, eq. (4.13)	same as FHV
APD	max	mean density per cluster, eq. (4.14)	same as FHV
PBMF	max	scatter ratio × maximal separation / c, eq. (4.8)	rewards one far-away cluster
KWON	min	XB-type ratio with a prototype-spread penalty, eq. (4.15)	sensitive to one close pair of prototypes
PCAES	max	cluster-wise PC − exponential separation, eq. (4.16)	a tiny spurious cluster inflates it

- The *geometric* criterion follows Krishnapuram and Freg and suits thin clusters: lines, planes, and local linear models. With $\boldsymbol{\phi}_i$ the eigenvector of the smallest eigenvalue of $\mathbf{F}_i$ (the normal of the cluster) and $\sigma_i = \sqrt{\lambda_{i,\max}}$ its extent along the main axis, two thin clusters ($\lambda_{\min}/\lambda_{\max} \leq \gamma_4$) are compatible if their normals are parallel, $\left|\boldsymbol{\phi}_i^\mathsf{T} \boldsymbol{\phi}_j\right| \geq \gamma_1$, their centers lie on a common hyperplane, $\left|\frac{1}{2}(\boldsymbol{\phi}_i + \boldsymbol{\phi}_j)^\mathsf{T}(\mathbf{v}_i - \mathbf{v}_j)\right| \leq \gamma_2 \cdot \frac{1}{2}(\sigma_i + \sigma_j)$, and the centers are close, $\left\|\mathbf{v}_i - \mathbf{v}_j\right\| \leq \gamma_3(\sigma_i + \sigma_j)$.
- The *similarity* criterion works for clusters of any shape: two clusters are compatible if their membership functions, seen as fuzzy sets over the samples, are similar. The toolbox measures this with the fuzzy Jaccard similarity of

Algorithm 4.1 Selecting the number of clusters by a validity scan

Input: data $\mathbf{X}$, estimator with fixed m, `tol`, `n_init`; range $c = 2, \dots, c_{\max}$; set of indices

for $c = 2, \dots, c_{\max}$ **do**
  fit the estimator with c clusters; obtain $\mathbf{U}_c$, $\mathbf{V}_c$;
  compute every index $I(\mathbf{X}, \mathbf{U}_c, \mathbf{V}_c; m)$ with Euclidean distances;
for *every index* **do**
  record $c_I^* = \arg\max_c I$ or $\arg\min_c I$ according to its direction;
  inspect the curve $I(c)$: knee for monotone indices, first pronounced extremum for the ratio indices;

Output: table of index values, suggested c_I^* per index; choose the c on which the geometric indices agree and confirm it visually (chapter 5)

the two columns of $\mathbf{U}$ [42],

$$S_{ij} = \frac{\sum_{k=1}^{N} \min(\mu_{ik}, \mu_{jk})}{\sum_{k=1}^{N} \max(\mu_{ik}, \mu_{jk})} \geq \gamma_S, \tag{4.17}$$

with a fixed threshold γ_S, evaluated after every converged fit. Kaymak and Setnes [43], who introduced adaptive merging by membership similarity, use a fuzzy inclusion measure instead, $S_{ij} = \max(I_{ij}, I_{ji})$ with $I_{ij} = \sum_k \min(\mu_{ik}, \mu_{jk}) / \sum_k \mu_{ik}$, and merge during the iteration: as soon as the largest similarity has stopped changing and exceeds $1/(c-1)$ for the current number of clusters c. Two FCM clusters that share one Gaussian group have $S_{ij} \approx 0.3$–0.35 for $m = 2$ and any c, two clusters on separate groups less than 0.2; to make the threshold independent of the exponent, the toolbox evaluates S_{ij} on the partition rescaled to $m = 2$, $\mu'_{ik} \propto \mu_{ik}^{m-1}$.

Merging needs at most $c_{\max} - c^*$ refits, each started close to its solution, against $c_{\max} - 1$ refits from scratch for a scan, and the clusters it ends with are interpretable as local models. This is why it is also the route taken in the fuzzy segmentation of time series (section 9.2.1), with a similarity of local PCA models as criterion, and in the analysis of the MST clustering (section 7.2.3), where the membership similarity of the refined clusters shows which of them may be merged; the volume-prototype clustering of chapter 6 merges by (4.17) within a single run, each time the partition has converged, without refitting from scratch.

4.2.8 Comparing fuzzy partitions

The indices so far judge a partition against the data. A different question is how similar two partitions are, for example those of two algorithms, of two numbers of clusters, or of two runs on resampled data, or a partition and a known reference.

Algorithm 4.2 Compatible cluster merging

Input: data $\mathbf{X}$, base estimator (GK by default), deliberately large $c_{\max}$, minimal $c_{\min}$, criterion (geometric with $\gamma_1, \dots, \gamma_4$, or similarity with γ_S)

fit the base estimator with $c \leftarrow c_{\max}$ clusters;
repeat
  compute the compatibility C_{ij} of every pair: geometric degree (zero unless all four conditions hold) or S_{ij} of (4.17) (zero below γ_S);
  if $\max_{ij} C_{ij} > 0$ *and* $c > c_{\min}$ **then**
    $(i, j) \leftarrow \arg\max C_{ij}$; merge: $\boldsymbol{\mu}_i \leftarrow \boldsymbol{\mu}_i + \boldsymbol{\mu}_j$, delete column j; $c \leftarrow c - 1$;
    refit the base estimator with c clusters, initialized with the merged partition;
until *no pair is compatible or* $c = c_{\min}$;

Output: final number of clusters c, the fitted estimator, the merge history (pairs and compatibilities)

In a nutshell Compatible cluster merging

- Starts with too many clusters and merges the most compatible pair, refitting after every merge, until no compatible pair remains: the number of clusters is a result, not an input.
- Geometric criterion (parallel normals, common hyperplane, close centers) for thin clusters and local linear models; membership similarity S_{ij} for clusters of any shape.
- Cheaper than a scan ($c_{\max} - c^*$ warm-started refits) and insensitive to $c_{\max}$ as long as it is large enough.
- The thresholds are the parameters; for the similarity criterion $\gamma_S = 0.2$ separates "one group" ($S \approx 0.3$–0.35) from "two groups".
- Parameters: `base_estimator`, `start_n_clusters`, `min_n_clusters`, `criterion`, `parallel_threshold`, `hyperplane_threshold`, `distance_threshold`, `thin_threshold`, `similarity_threshold`; attributes `merge_history_`, `estimator_`.

For hard partitions the Rand index answers it by counting the pairs of samples on which the partitions agree—together in both or apart in both—and the adjusted Rand index of Hubert and Arabie [44] removes the agreement expected by chance. Both have fuzzy generalizations, and both reduce to the classical indices when the partitions are hard.

Hüllermeier, Rifqi, Henzgen, and Senge [45] read a fuzzy partition as a *fuzzy equivalence relation*, the degree to which samples k and l are in the same cluster,

$$E_{\mathbf{U}}(k,l) = 1 - \tfrac{1}{2}\sum_{i=1}^{c} \left|\mu_{ik} - \mu_{il}\right|, \tag{4.18}$$

one for identical membership vectors and zero for disjoint ones, and define the fuzzy Rand index as one minus the mean distance between the relations of two partitions $\mathbf{U}_1$ and $\mathbf{U}_2$ (possibly with different numbers of clusters),

$$R(\mathbf{U}_1,\mathbf{U}_2) = 1 - \frac{\sum_{k<l} \left|E_1(k,l) - E_2(k,l)\right|}{N(N-1)/2}. \tag{4.19}$$

The paper defines no adjusted index. For a chance-corrected comparison the toolbox therefore builds fuzzy pair counts from the two relations with the Łukasiewicz t-norm $\top(a,b) = \max(a+b-1, 0)$; this construction is the book's own (the Jaccard-type measures of Hüllermeier et al. split the pairs differently, with the product t-norm). The degrees to which a pair is together in both partitions, together only in the first, only in the second, and apart in both are $n_{11} = \sum_{k<l} \top(E_1, E_2)$, $n_{10} = \sum_{k<l} \top(E_1, 1-E_2)$, $n_{01} = \sum_{k<l} \top(1-E_1, E_2)$ and $n_{00} = \sum_{k<l} \top(1-E_1, 1-E_2)$, and they give a count-based Rand index $(n_{11}+n_{00})/n_p$ with $n_p = n_{11}+n_{10}+n_{01}+n_{00}$. For hard partitions $n_p = N(N-1)/2$ and the count-based form equals (4.19); for soft partitions it does not, because the four degrees of a pair need not sum to one ($n_{11} + n_{00}$ per pair is $|E_1 + E_2 - 1|$, $n_{10} + n_{01}$ is $|E_1 - E_2|$), so the toolbox uses the distance form (4.19), the definition of [45], as the fuzzy Rand index. With Hubert and Arabie's correction the counts give the *fuzzy adjusted Rand index*

$$ARI = \frac{n_{11} - e}{\frac{1}{2}\big((n_{11}+n_{10}) + (n_{11}+n_{01})\big) - e}, \qquad e = \frac{(n_{11}+n_{10})(n_{11}+n_{01})}{n_p}, \tag{4.20}$$

where e is the agreement expected by chance, and the fuzzy Jaccard index $n_{11}/(n_{11} + n_{10} + n_{01})$. Campello [46] builds the counts differently, from the degrees that a pair is in the same cluster, $V_{kl} = \sum_i \top(\mu_{ik}, \mu_{il})$, and in different clusters, $X_{kl} = \sum_{i\neq j} \top(\mu_{ik}, \mu_{jl})$, combined with a t-norm across the two partitions ($n_{11} = \sum_{k<l} \top(V^{(1)}_{kl}, V^{(2)}_{kl})$, $n_{10} = \sum_{k<l} \top(V^{(1)}_{kl}, X^{(2)}_{kl})$, and so on). With the product t-norm, Campello's index is not reflexive for fuzzy partitions: a partition compared with itself scores below one, the more so the fuzzier it is. It is useful for ranking partitions against a reference, whereas the index of Hüllermeier et al. is a proper metric-based similarity.

The raw fuzzy Rand index is dominated by the pairs that are apart in both partitions, of which there are many when c is large, so it is close to one for almost

In a nutshell **Comparing fuzzy partitions**

- Fuzzy Rand, adjusted Rand, and Jaccard indices compare two partitions of the same samples, with any numbers of clusters, any algorithms, hard or fuzzy.
- Hüllermeier et al.: distance between the fuzzy equivalence relations $E_U(k, l)$; reflexive, reduces to the classical indices for hard partitions.
- Campello: t-norm-based counts of "same cluster" and "different clusters"; not reflexive for fuzzy partitions with the product t-norm.
- Read the adjusted index (zero for chance agreement); the raw index is near one for large c.
- Uses: comparing algorithms and numbers of clusters, comparing with a reference partition, and the stability selection below.

any pair of reasonable partitions; the adjusted index is the one to read. A fuzzy partition and the hard partition obtained by hardening it do not reach an adjusted index of one: the index measures the agreement of the relations, and memberships below one are a genuine difference.

4.2.9 Stability-based selection of the number of clusters

A third route to the number of clusters asks for reproducibility instead of geometry: if the data contain c^* groups, a partition into c^* clusters should be nearly the same on different samples of the data, whereas a partition into too many clusters splits a group at a place that depends on the sample and a partition into too few merges groups in a sample-dependent way. Ben-Hur, Elisseeff, and Guyon [47] cluster two random sub-samples of a fraction f of the data and compare the two partitions on the samples they share; Lange, Roth, Braun, and Buhmann [48] split the data into two halves, cluster both, let the model of the first half *predict* the second half, and compare the prediction with the second half's own partition. For a fuzzy clusterer the prediction is `predict_proba`, and the comparison is the fuzzy adjusted Rand index (4.20), which is zero for chance agreement and therefore needs no further normalization across c (the raw Rand index increases with c). Repeating the procedure gives a distribution of agreements for every c; a stable number of clusters has a mean near one *and* a small spread (algorithm 4.3). The sub-sample fraction must leave room for variation: with f close to one the sub-samples are nearly identical and every c looks stable.

Algorithm 4.3 Stability selection of the number of clusters

Input: data $\mathbf{X}$, estimator, range $c = 2, \dots, c_{\max}$, number of repetitions B, protocol (sub-sample with fraction f, or split half), comparison index (fuzzy ARI)

for $c = 2, \dots, c_{\max}$ **do**
  for $b = 1, \dots, B$ **do**
    if *sub-sample protocol* **then**
      draw two random sub-samples I_1, I_2 of size fN; fit the estimator with c clusters on each;
      memberships of the common samples $I_1 \cap I_2$ under both models (`predict_proba`): $\mathbf{U}_1, \mathbf{U}_2$;
    else
      split the data randomly into halves H_1, H_2; fit on each;
      $\mathbf{U}_1 \leftarrow$ memberships of H_2 predicted by the model of H_1;
      $\mathbf{U}_2 \leftarrow$ partition of H_2 by its own model;
    $s_{c,b} \leftarrow ARI(\mathbf{U}_1, \mathbf{U}_2)$;

Output: table $s_{c,b}$; the most stable c has the largest mean and a small spread of $s_{c,\cdot}$.

In a nutshell Stability selection

- Chooses the c whose partitions are reproducible across resamples: sub-sample pairs (Ben-Hur et al.) or split-half prediction (Lange et al.).
- Needs no assumption about cluster shape beyond the clusterer's own; compares partitions with the fuzzy adjusted Rand index.
- Read mean and spread per c; a large spread at a c means that the partition depends on the sample.
- Costs $2B$ fits per c (the most expensive route); the sub-sample fraction should lie between about 0.5 and 0.8.
- Parameters: `n_clusters_range`, `n_splits`, `method`, `subsample`, `index`, `random_state`.

In a nutshell **Validity indices**

- Sixteen indices in four families: fuzziness of $\mathbf{U}$ (PC, CE, MPC), compactness/separation (SC, S, XB, FS, PBMF, KWON, PCAES), crisp (DI, ADI, SIL), and volume and density (FHV, PD, APD); all with Euclidean distances.
- Direction of the optimum is in `INDEX_DIRECTION`; PC and CE are monotone in c on overlapping data (read the knee); XB and S jump when two prototypes first share a group (read the first minimum).
- Scan c with identical settings and enough restarts; prefer the c on which the geometric indices agree; confirm visually.
- Compatible cluster merging (section 4.2.7) is the alternative when clusters are local models, ordered segments, or nodes of a tree (chapters 7 to 9), stability selection (section 4.2.9) when reproducibility is the criterion.
- An index extremum is a hint, not a proof: indices measure geometric separation, not agreement with any external labels.

4.3 Using it in fuzzyclust

`fuzzyclust.validity.<index>(X, U, V, m=2.0)`

Every index is a function with the same signature: `X` (N, n), `U` the membership matrix (N, c) as stored by the estimators (i.e. $\mathbf{U}^\mathsf{T}$), `V` the prototypes (c, n) and `m` the exponent the algorithm used. PC, CE, and MPC ignore `X` and `V`; DI, ADI, and SIL harden `U` first; FHV, PD, and APD compute the fuzzy covariances from `X`, `U`, `V`, and `m`. The functions are `partition_coefficient` (PC), `classification_entropy` (CE), `modified_partition_coefficient` (MPC), `partition_index` (SC), `separation_index` (S), `xie_beni` (XB), `fukuyama_sugeno` (FS), `dunn_index` (DI), `alternative_dunn_index` (ADI), `silhouette` (SIL), `fuzzy_hypervolume` (FHV), `partition_density` (PD), `average_partition_density` (APD), `pbmf_index` (PBMF), `kwon_index` (KWON), and `pcaes_index` (PCAES); the indices that need a separation between prototypes (SC, S, XB, KWON, PCAES) return `nan` for a single cluster. `validity_indices(X, U, V, m=2.0, indices=None)` evaluates a subset or all of them and returns a `dict`. `INDEX_DIRECTION` maps every short name to `"max"` or `"min"`.

fuzzyclust.validity.scan_n_clusters(estimator, X, n_clusters_range=range(2, 11), indices=None)

Clones the estimator (any `fuzzyclust` clusterer, fitted or not), sets `n_clusters` to every value of the range, fits, and returns a `pandas.DataFrame` indexed by `n_clusters` with one column per index. The exponent is taken from the estimator's `m`. `suggest_n_clusters(table)` returns a `Series` with the arg-max or arg-min of every column according to `INDEX_DIRECTION`; for PC and CE the knee is often the better reading, so the raw suggestion is reported for completeness only.

fuzzyclust.plotting.plot_validity_scan(table, ax=None, normalize=True)

Plots every column of a scan table against c, min–max scaled to $[0, 1]$ by default so that all indices share one axis; the legend states the direction of each optimum. Pass a column subset, e.g. `table[["XB", "S", "SIL"]]`, to compare families separately.

fuzzyclust.validity.fuzzy_rand_index(U1, U2, method="huellermeier", tnorm=None)

Fuzzy Rand index (4.19) of two membership matrices of the same samples, (N, c_1) and (N, c_2); hard partitions are passed as one-hot matrices, which `fuzzyclust._utils.labels_to_membership` builds from labels. `method="campello"` uses Campello's counts; `tnorm` is `"product"`, `"minimum"`, or `"lukasiewicz"` (`None`: the distance form (4.19) for Hüllermeier, product for Campello; a t-norm selects the count-based form). `fuzzy_adjusted_rand_index` (4.20) and `fuzzy_jaccard_index` take the same arguments. The pairwise relations need $\mathcal{O}(N^2 c)$ memory, so compare subsamples of a few thousand samples at most.

fuzzyclust.validity.stability_selection(estimator, X, n_clusters_range=range(2, 11), n_splits=20, method="subsample", subsample=0.8, index="fuzzy_ari", random_state=None)

Stability selection as in section 4.2.9; `method` is `"subsample"` (Ben-Hur et al.) or `"split"` (Lange et al.); `index` one of `"fuzzy_ari"`, `"fuzzy_rand"`, `"fuzzy_jaccard"`. Returns a `DataFrame` indexed by `n_clusters` with one column per repetition; `table.mean(axis=1).idxmax()` is the most stable c, and `table.T` can be passed to a box plot.

`fuzzyclust.cluster.CompatibleClusterMerging`

`base_estimator=None`, `start_n_clusters=8`, `min_n_clusters=1`, `criterion="geometric"`, `parallel_threshold=0.95`, `hyperplane_threshold=0.5`, `distance_threshold=2.0`, `thin_threshold=0.2`, `similarity_threshold=0.2`. `base_estimator=None` means `GK(random_state=0)`; any `fuzzyclust` clusterer is accepted; its `n_clusters` and `init` are overridden. The four geometric thresholds are $\gamma_1, \ldots, \gamma_4$, `similarity_threshold` is γ_S. Attributes: `n_clusters_`, `estimator_` (the final fitted model), `merge_history_` (one `dict` per merge with `n_clusters`, `pair`, `compatibility`), `compatibility_`, and `cluster_centers_`, `membership_`, `labels_`, `distances_` of the final model; `predict`, `predict_proba`, and `transform` delegate to it. The helper functions `cluster_similarity(U)` and `geometric_compatibility(V, F, ...)` compute the two criteria.

The three routes in a few lines:

```
ccm = CompatibleClusterMerging(FCM(random_state=0), start_n_clusters=10,
    criterion="similarity",
                                similarity_threshold=0.2).fit(X)
for step in ccm.merge_history_:
    print(f"c = {step['n_clusters']:2d}: merge clusters {step['pair']}
    (similarity {step['compatibility']:.2f})")
print(f"final number of clusters: {ccm.n_clusters_}")
```

```
table = scan_n_clusters(FCM(random_state=0, n_init=3), X, range(2, 10),
                        indices=["XB", "KWON", "PCAES", "PBMF", "SIL"])
print(suggest_n_clusters(table).to_dict())
stab = stability_selection(FCM(random_state=0), X, range(2, 10),
    n_splits=20, method="subsample",
                           random_state=0)
stab_split = stability_selection(FCM(random_state=0), X, range(2, 10),
    n_splits=20, method="split",
                                 random_state=0)
print("most stable c (subsample, split):", stab.mean(axis=1).idxmax(),
    stab_split.mean(axis=1).idxmax())
```

and the comparison of partitions obtained in different ways:

```
partitions = {
    "truth": labels_to_membership(y, 5),
    "FCM c=5": FCM(n_clusters=5, random_state=0).fit(X).membership_,
    "GK c=5": GK(n_clusters=5, random_state=0).fit(X).membership_,
    "merging": ccm.membership_,
    "FCM c=4": FCM(n_clusters=4, random_state=0).fit(X).membership_,
    "FCM c=8": FCM(n_clusters=8, random_state=0).fit(X).membership_,
}
names = list(partitions)
```

```
ari = np.array([[fuzzy_adjusted_rand_index(partitions[a], partitions[b])
    for b in names] for a in names])
```

A complete scan with all indices and its plot:

```
from fuzzyclust.validity import scan_n_clusters, suggest_n_clusters
from fuzzyclust.plotting import plot_validity_scan

table = scan_n_clusters(FCM(m=2.0, random_state=0, n_init=5), X, range(2,
    10))
print(suggest_n_clusters(table))
plot_validity_scan(table[["SC", "S", "XB", "DI", "ADI", "SIL"]])
```

Because every index is a plain function of `(X, U, V, m)`, it can also score held-out data through `predict_proba`, which is how the grid search of section 2.4.4 selects c and m with the Xie–Beni index:

```
def neg_xie_beni(estimator, X):            # scikit-learn scorer: higher
    is better
    U = estimator.predict_proba(X)
    return -validity_indices(X, U, estimator.cluster_centers_,
    m=estimator.m, indices=["XB"])["XB"]
```

4.4 Examples

4.4.1 A validity scan on five clusters

The script `plot_05_validity_scan.py` generates five compact Gaussian clusters and scans FCM for $c = 2, \dots, 9$. Table 4.2 lists the index values; thirteen of the fourteen indices place their optimum at $c = 5$, and only SC, which is designed for comparing partitions with equal c and decreases with c, suggests $c_{\max}$. The two indices of section 4.2.5, which the script computes as well, disagree: $KWON$ has its minimum at five (20.8, against 26.7 at $c = 3$ and above 200 for $c \geq 6$), $PCAES$ its maximum at six (9.8 against 3.4 at five), a jump caused by a small sixth cluster whose mass μ_M divides the first sum of (4.16). On these well-separated data even PC and CE have a genuine extremum at five, whereas on the overlapping motorcycle data of the MATLAB manual they were monotone and only their knee was informative. Figure 4.3 shows the normalized curves. The ratio indices XB and S fall steeply from $c = 2$ to $c = 3$, reach their minimum at $c = 5$, and jump to a much larger value at $c = 6$, when a prototype is first placed inside an existing group and the minimal center distance collapses. DI has a sharp peak because five is the only c at which every pair of hardened clusters is separated by a clear gap; at every other c some pair nearly touches. The script also scans GK on the same data: with adaptive shapes its XB and SIL values are close to those of FCM, and the two algorithms agree on $c = 5$. Finally, it repeats the FCM scan for m from 1.3 to 3. XB, DI, SIL, and MPC select five clusters for every exponent, and S, FS,

and ADI deviate only at the ends of the range, whereas PC and CE select $c = 5$ for $m \leq 2$ and $c = 2$ for $m \geq 2.5$: a softer partition makes fewer clusters look less fuzzy, which is the dependence on m noted in section 4.2.1.

Table 4.2. Validity scan of FCM ($m = 2$) on five Gaussian clusters, $c = 2, \ldots, 9$ (script `plot_05_validity_scan.py`). Bold: optimum of each index. S (not shown) equals XB because $m = 2$; ADI (not shown) is 0.006 at $c = 5$ and below 0.0005 elsewhere.

c	PC	CE	MPC	SC	XB	FS	DI	SIL	FHV	PD	APD	PBMF
2	0.733	0.416	0.466	0.507	0.254	643	0.020	0.449	5.55	28.8	28.9	22.7
3	0.746	0.470	0.618	0.083	0.066	−2962	0.004	0.588	3.42	43.9	45.5	51.9
4	0.774	0.460	0.699	0.048	0.068	−3735	0.012	0.619	2.33	69.5	74.5	85.9
5	**0.834**	**0.383**	**0.793**	0.021	**0.049**	**−4335**	**0.155**	**0.715**	**1.52**	**105.3**	**105.5**	**98.4**
6	0.767	0.516	0.720	0.021	0.617	−4150	0.019	0.623	1.75	94.0	95.5	90.5
7	0.709	0.623	0.661	0.017	0.553	−3838	0.022	0.537	1.90	85.0	86.6	85.4
8	0.665	0.711	0.617	0.014	0.549	−3587	0.019	0.459	1.93	83.4	84.9	87.6
9	0.626	0.780	0.579	**0.011**	0.463	−3342	0.019	0.375	1.94	81.0	82.6	98.1

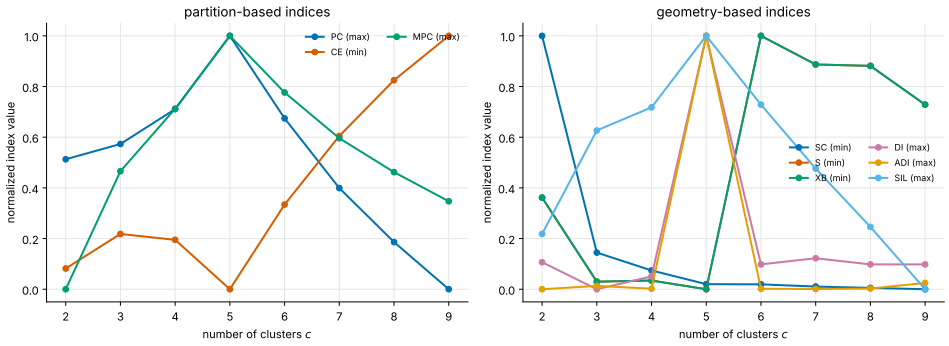


Figure 4.3. Validity scan of FCM on five Gaussian clusters, indices min–max scaled to $[0, 1]$. Left: the partition-based indices have their extremum at $c = 5$ because the clusters are well separated. Right: apart from SC, which decreases with c, the geometric indices agree on $c = 5$; XB and S (identical for $m = 2$, so only the XB curve is visible) jump when a sixth prototype enters an existing group, and DI and ADI peak sharply.

4.4.2 Three routes on the same data

The script `plot_14_merging_stability.py` runs merging, the validity scan, and stability selection on five Gaussian groups of different sizes and spreads. Compatible cluster merging with the similarity criterion starts from ten FCM clusters (fig. 4.4, left), in which three of the groups are covered by two or three prototypes each. The five merges join clusters with similarities between 0.36 and 0.30—each time two clusters on one group—and after the fifth merge the largest similarity left is 0.10, well below the threshold 0.2: the procedure stops at five clusters, whose partition coincides with that of FCM run directly with $c = 5$ (fuzzy adjusted Rand index 1.00).

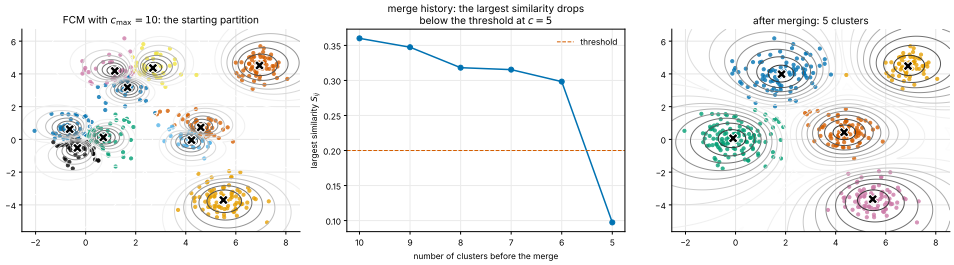


Figure 4.4. Compatible cluster merging with the similarity criterion on five groups. Left: FCM with $c_{\max} = 10$. Middle: the largest similarity S_{ij} before every merge; it falls below the threshold (dashed) after the partition has reached five clusters. Right: the final partition.

The validity scan and the stability selection are shown in fig. 4.5. XB, the Kwon index, and the silhouette select five clusters, $PBMF$ six, and $PCAES$ has its maximum at nine, where FCM creates a cluster of a few samples whose small mass inflates the first sum of (4.16); up to $c = 6$ its curve rises steadily, so it does not reject five either. The stability profile gives the clearest answer: for $c = 5$ the twenty sub-sample pairs agree with a mean fuzzy adjusted Rand index of 0.97 and a standard deviation of 0.008, while for $c = 4$ the mean is 0.88 and the spread twenty times larger, because which two groups are merged depends on the sub-sample; the split-half protocol peaks at five as well (0.92).

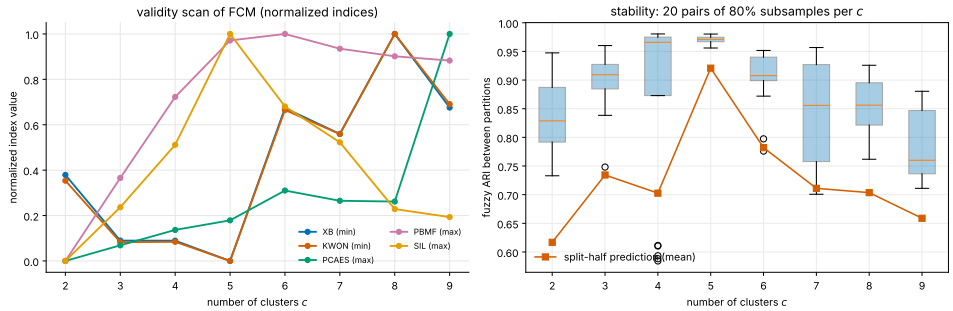


Figure 4.5. Validity scan (left) and stability selection (right) on the same five groups. The box plots show the fuzzy adjusted Rand index of twenty pairs of 80 % sub-samples for every c, the squares the mean of the split-half protocol; both peak at $c = 5$, where the box is also the narrowest.

Figure 4.6 compares six partitions of the data: the generating labels as a one-hot matrix, FCM and GK with five clusters, the merged partition, and FCM with four and eight clusters. The raw fuzzy Rand index (left) is above 0.8 for every pair and hardly discriminates; the adjusted index (middle) does: FCM, GK, and the merged partition agree with each other (0.95 to 1.00) and with the truth (0.70, the memberships below one keep the value below one), whereas FCM with four or eight clusters reaches only 0.58 and 0.56 against the truth. Campello's index with the product t-norm (right) ranks the partitions against the truth in the same order

but is not reflexive: every fuzzy partition compared with itself scores only 0.42 to 0.54.

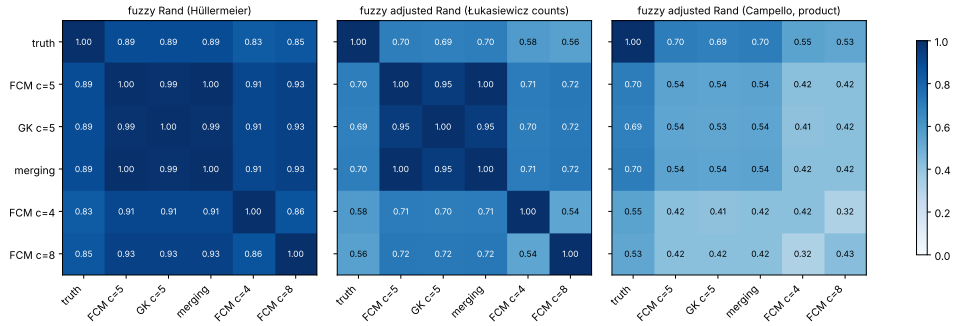


Figure 4.6. Fuzzy Rand index of Hüllermeier et al., the fuzzy adjusted Rand index from Łukasiewicz pair counts, and Campello's fuzzy adjusted Rand index between six partitions of the same data. Only the adjusted indices separate the good partitions from those with a wrong number of clusters; Campello's index is below one on the diagonal for fuzzy partitions.

The geometric criterion is demonstrated on three noisy line segments (fig. 4.7): Gustafson–Kessel with ten clusters cuts every segment into thin pieces, and merging pieces with parallel normals on a common line leaves three clusters (fuzzy adjusted Rand index 0.83 with the truth). The parameter study (fig. 4.8) shows a wide plateau of the similarity threshold: every value from 0.10 to 0.25 ends at five clusters, 0.30 stops one merge early, 0.35 and above hardly merge at all, and 0.05 merges distinct groups down to a single cluster. For the stability selection, sub-samples of 50 % and 80 % both select five clusters, the smaller fraction with lower agreements for every c except four, where the two are equal; with 95 % the sub-samples are nearly identical and four clusters appear almost as stable as five (0.983 against 0.987), which is why the fraction must leave room for variation.

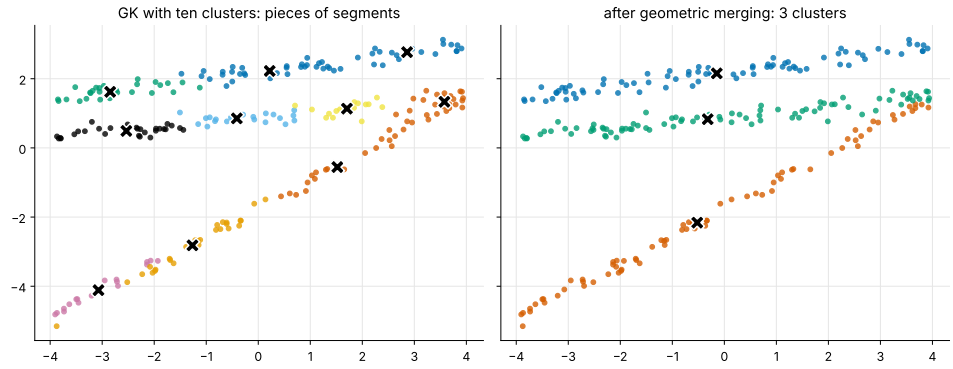


Figure 4.7. Compatible cluster merging with the geometric criterion on three line segments: GK with ten clusters (left) and the three clusters left after merging pieces with parallel normals on a common line (right).

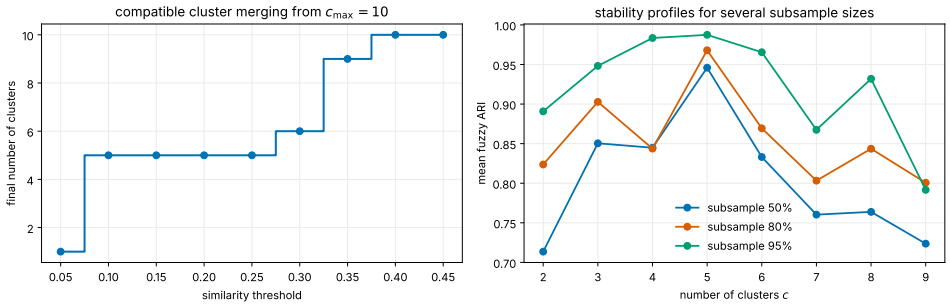


Figure 4.8. Parameter study: the final number of clusters of the similarity merging against its threshold (left) and the mean stability profile for three sub-sample fractions (right).

4.4.3 Combining indices and reading a scan

The scan above is the easy case. In practice the indices disagree, and the following rules of thumb, distilled from the MATLAB manual's motorcycle study and from the examples of this book, help in reading them. Prefer the c on which XB, S, and SIL agree, and take the *first* pronounced minimum of XB rather than its global one. Use SC and PC/CE to compare partitions with the *same* c and the same m—different algorithms or initializations—rather than across c. Treat DI and ADI as confirmations for well-separated data and ignore them when the clusters overlap. Use FHV, PD, and APD for GK and GG partitions of ellipsoidal clusters, and distrust them whenever a covariance matrix is nearly singular. When the differences between neighboring c are minor, choose the smaller c. And when the indices leave two candidates, let the picture decide. Figure 5.14 in chapter 5 shows FCM partitions of the breast cancer data for $c = 2, 3, 4$ projected with the fuzzy Sammon mapping. The indices PC, CE, and XB favor two clusters (SC, which decreases with c, points to four); the pictures show *why*—the third prototype splits the large benign group, the fourth the malignant one, and the clusters overlap in the projection—and also *where* the ambiguous cases lie.

4.4.4 New data as validation

A partition that describes structure rather than noise should generalize: samples that were not used for fitting should fall into the same regions with the same confidence as the training samples. The script of section 3.4.7 (`plot_06_predict_new_data.py`) fits GK on 60 % of three elongated clusters and evaluates the remaining 40 % with `predict_proba` (fig. 4.9). The test samples fall into the membership regions learned from the training data; their memberships and their distances to the prototypes (`transform`) have the same distribution as those of the training set. Three uses follow. Held-out validity indices computed from `predict_proba` are less optimistic than training indices and can drive a grid search over c and m (section 2.4.4). The stability of the partition across random train/test splits or bootstrap samples is itself a validity criterion (section 4.2.9): a c whose prototypes

move little between resamples is supported by the data. Finally, a new sample whose maximal membership is low and whose distance to the nearest prototype is large—the far-away point of the script has a maximal membership of 0.52 and a distance of 28, against a training median of 1.2—is a novelty, which is the basis of clustering-based process monitoring.

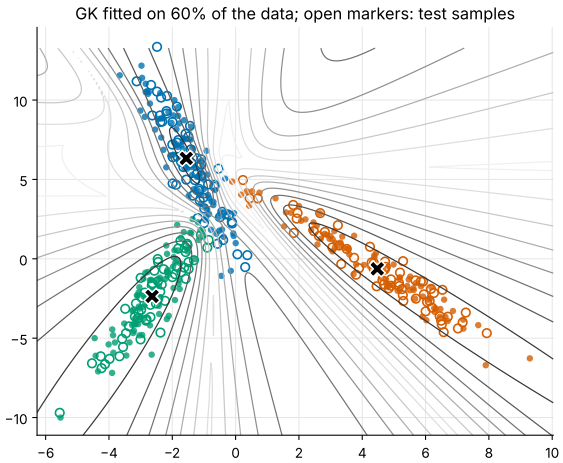


Figure 4.9. Gustafson–Kessel fitted on 60 % of the data and evaluated on the rest: the test samples (open markers, colored by predicted label) fall into the membership regions learned from the training data. Held-out memberships and distances validate the partition and detect novelties.

4.5 Further reading

The partition coefficient and classification entropy are Bezdek's [3], the modified partition coefficient Davé's [29]; the partition index is from Bensaid et al. [30], the Xie–Beni index, whose original form is the separation index S, from [31], with the analysis of its dependence on m by Pal and Bezdek [32], and the Fukuyama–Sugeno index from [33]. Dunn's index is in [35], the silhouette in Rousseeuw [36], and the PBM-F index in Pakhira, Bandyopadhyay, and Maulik [34]; the alternative Dunn index is from the MATLAB toolbox manual [12]. The fuzzy hypervolume and the partition densities are from Gath and Geva [21]. Compatible cluster merging goes back to Krishnapuram and Freg [39] and was brought to fuzzy modeling by Kaymak and Babuška [40], with Kelly's merging formulas [41]; adaptive merging by membership similarity is from Kaymak and Setnes [43], and the Jaccard similarity used by the toolbox from Setnes et al. [42]. The Kwon index is in [37], the PCAES index in Wu and Yang [38], who also compare it with many of the indices above. The fuzzy Rand indices are from Hüllermeier et al. [45] and Campello [46], the adjustment for chance from Hubert and Arabie [44]; stability-based validation is developed by Ben-Hur, Elisseeff, and Guyon [47] and Lange et al. [48]. Babuška [7] discusses validity scans and cluster merging in the context of fuzzy modeling, and Höppner et al. [6] survey validity measures for the whole objective-function family.

CHAPTER

5



Visualization of fuzzy clustering results

The previous chapter judged a fuzzy partition by validity indices, that is, by single numbers. This chapter projects the data *and* the cluster prototypes to the plane, so that the analyst can see whether the clusters are compact, whether two of them overlap, or whether a prototype of the wrong shape was fitted. We start with the two classical projections, PCA and Sammon mapping, explain why neither shows a fuzzy clustering faithfully, and derive the fuzzy Sammon mapping (FUZZSAM), which preserves the $N \times c$ sample–prototype distances the clustering algorithm worked with rather than the distances between samples; measures of the quality of a projection follow. The fuzzy self-organizing map takes the opposite route: it arranges the prototypes on a grid during clustering, so that the result is a map by construction. All methods are in `fuzzyclust.viz`.

5.1 Problem and idea

Clustering algorithms always fit their prototypes to the data, whether or not the assumed cluster structure is adequate. In two dimensions the analyst sees at once whether a spherical prototype was forced onto an elongated group or two groups were merged. In a typical data mining problem the data have tens of attributes, the clusters are high-dimensional geometric objects, and this inspection is impossible. Validity indices (chapter 4) compress the partition into a scalar; they are indispensable for ranking candidate partitions, but the compression loses the information on *where* and *why* a partition is poor.

A low-dimensional picture of the clustering is therefore a tool in its own right [49, 50, 51]. Two classical projection methods are at hand. Principal component

analysis (PCA) is a linear projection onto the plane of largest variance; Sammon's nonlinear mapping [52] places N points in the plane so that their mutual Euclidean distances reproduce the $N(N-1)/2$ distances of the original samples. Both methods project the *data*. A fuzzy clustering, however, is not defined by the distances between samples: it is defined by the $N \times c$ distances $D_{ik} = d(\mathbf{x}_k, \mathbf{v}_i)$ between the samples and the prototypes, measured with the norm of the algorithm (Euclidean for FCM, the adaptive norm $\mathbf{A}_i$ of Gustafson–Kessel, the exponential distance of Gath–Geva, the distance to a line or a shell for the variants of chapter 6), and by the memberships derived from them. Two consequences follow:

- a projection of the data cannot represent a clustering that uses a non-Euclidean distance; a Gustafson–Kessel ellipsoid appears as an ellipsoid in the PCA plane, although the algorithm "sees" it as a sphere in its own norm, and the prototype of a line or shell cluster has no natural position in the picture at all;
- the $N(N-1)/2$ pairwise distances of Sammon mapping make every iteration quadratic in N, while the clustering itself needs only Nc distances per iteration.

The fuzzy Sammon mapping resolves both problems with one idea (fig. 5.1): it looks for N projected samples $\mathbf{y}_k \in \mathbb{R}^q$ ($q = 2$) and c projected centers $\mathbf{z}_i \in \mathbb{R}^q$ such that the Euclidean distances $d^*_{ik} = \|\mathbf{y}_k - \mathbf{z}_i\|$ in the plane reproduce the distances D_{ik} of the clustering algorithm, each pair weighted by μ^m_{ik}. Every cluster becomes a single point in the plane regardless of the shape of its prototype, the memberships can be recomputed from the picture with the FCM formula, and the agreement between recomputed and fitted memberships tells how faithful the picture is. The fuzzy self-organizing map, finally, does not project an existing partition but constrains the prototypes to a grid while clustering; the grid *is* the picture.

5.2 Theory

Throughout the chapter $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^\mathsf{T} \in \mathbb{R}^{N \times n}$ is the data matrix, $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_c]^\mathsf{T}$ the prototypes, $\mathbf{U} = [\mu_{ik}] \in \mathbb{R}^{c \times N}$ the fuzzy partition, and $D_{ik} = d(\mathbf{x}_k, \mathbf{v}_i)$ the (not squared) distance of the clustering algorithm; in the code $\mathbf{U}$ is stored transposed as `membership_` of shape `(n_samples, n_clusters)` and D_{ik} is returned by `transform`. The projection has dimension q (two in all pictures); $\mathbf{y}_k \in \mathbb{R}^q$ denotes the projected sample and $\mathbf{z}_i \in \mathbb{R}^q$ the projected prototype. We first review the two classical data projections (sections 5.2.1 and 5.2.2), then derive FUZZSAM (section 5.2.3) and the measures of projection quality (section 5.2.4), and finally present the fuzzy self-organizing map (section 5.2.5).

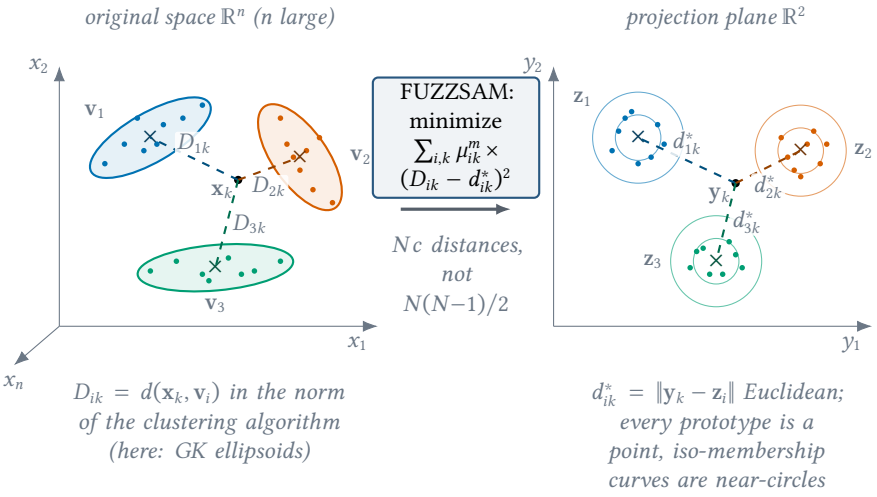


Figure 5.1. The idea of the fuzzy Sammon mapping. Left: in the original space the clustering algorithm measures the distance of every sample $\mathbf{x}_k$ to every prototype $\mathbf{v}_i$ in its own norm (dashed lines; for Gustafson–Kessel the clusters are ellipsoids). Right: the projected samples $\mathbf{y}_k$ and centers $\mathbf{z}_i$ are placed so that the Euclidean distances d_{ik}^* in the plane reproduce the D_{ik}, weighted by the memberships. Only $N \times c$ distances are involved, and in the plane every cluster is a point surrounded by nearly circular membership contours (exact circles for two clusters).

5.2.1 Principal component projection

PCA [53] projects the centered data onto the q eigenvectors $\mathbf{w}_1, \dots, \mathbf{w}_q$ of the covariance matrix

$$\mathbf{F} = \frac{1}{N} \sum_{k=1}^{N} (\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^{\mathsf{T}}, \qquad \mathbf{y}_k = \mathbf{W}^{\mathsf{T}}(\mathbf{x}_k - \bar{\mathbf{x}}), \quad \mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_q], \tag{5.1}$$

belonging to the q largest eigenvalues. The plane captures the largest possible share of the total variance, and the ratio of the retained eigenvalues to their sum tells how large this share is. Because the map is affine, the prototypes are projected with the same formula, $\mathbf{z}_i = \mathbf{W}^{\mathsf{T}}(\mathbf{v}_i - \bar{\mathbf{x}})$; for Euclidean prototypes (FCM) the $\mathbf{z}_i$ coincide with the membership-weighted means of the projected data. PCA is fast, deterministic, and applicable to new samples, but it preserves variance, not distances: clusters that differ in low-variance directions fall on top of each other, and nothing in the picture refers to the norm of the clustering algorithm.

5.2.2 Sammon mapping

Sammon's mapping [52] is a multidimensional scaling method. It looks for N points $\mathbf{y}_k \in \mathbb{R}^q$ whose pairwise distances $d_{kl}^* = \|\mathbf{y}_k - \mathbf{y}_l\|$ approximate the pairwise

Algorithm 5.1 Principal component projection of data and prototypes

Input: data $\mathbf{X}$, prototypes $\mathbf{V}$, dimension q

1 center the data: $\bar{\mathbf{x}} = \frac{1}{N}\sum_k \mathbf{x}_k$, $\tilde{\mathbf{x}}_k = \mathbf{x}_k - \bar{\mathbf{x}}$;

2 compute the covariance $\mathbf{F}$ of (5.1) and its eigen-decomposition $\mathbf{F} = \sum_j \lambda_j \mathbf{w}_j \mathbf{w}_j^\mathsf{T}$, $\lambda_1 \geq \lambda_2 \geq \ldots$;

3 $\mathbf{W} \leftarrow [\mathbf{w}_1, \ldots, \mathbf{w}_q]$;

4 $\mathbf{y}_k \leftarrow \mathbf{W}^\mathsf{T}\tilde{\mathbf{x}}_k$ for $k = 1, \ldots, N$; $\quad \mathbf{z}_i \leftarrow \mathbf{W}^\mathsf{T}(\mathbf{v}_i - \bar{\mathbf{x}})$ for $i = 1, \ldots, c$;

Output: Y, Z, explained variance $\sum_{j\leq q}\lambda_j / \sum_j \lambda_j$

In a nutshell PCA projection

- Linear, variance-preserving projection onto the plane of the first two principal components.
- Prototypes are mapped with the same affine transform, so data and centers stay consistent.
- Deterministic, cheap, extendable to new data; the natural starting point of the nonlinear methods.
- Ignores distances and the algorithm's norm; compact high-dimensional clusters may overlap in the plane.

distances $d_{kl} = \|\mathbf{x}_k - \mathbf{x}_l\|$ of the original data by minimizing the *stress*

$$E = \frac{1}{\lambda}\sum_{k<l}\frac{(d_{kl} - d^*_{kl})^2}{d_{kl}}, \qquad \lambda = \sum_{k<l} d_{kl}. \tag{5.2}$$

The division by d_{kl} gives small distances a larger weight than a plain least-squares fit would, so that the local structure is preserved; the constant λ only normalizes E and does not change the optimum. $E = 0$ means that every distance is reproduced exactly. The minimization in the Nq coordinates y_{kl} is carried out by Sammon's diagonal Newton step with step size ("magic factor") α,

$$y_{kl}(t+1) = y_{kl}(t) - \alpha\,\frac{\partial E/\partial y_{kl}}{\left|\partial^2 E/\partial y_{kl}^2\right|}, \tag{5.3}$$

with

$$\frac{\partial E}{\partial y_{kl}} = -\frac{2}{\lambda}\sum_{j\neq k}\frac{d_{kj} - d^*_{kj}}{d_{kj}d^*_{kj}}(y_{kl} - y_{jl}), \tag{5.4}$$

$$\frac{\partial^2 E}{\partial y_{kl}^2} = -\frac{2}{\lambda}\sum_{j\neq k}\frac{1}{d_{kj}d^*_{kj}}\left[(d_{kj} - d^*_{kj}) - \frac{(y_{kl} - y_{jl})^2}{d^*_{kj}}\left(1 + \frac{d_{kj} - d^*_{kj}}{d^*_{kj}}\right)\right]. \tag{5.5}$$

Sammon recommended $\alpha \approx 0.3$–0.4. The denominator in (5.3) is a curvature along one coordinate and can be arbitrarily small for single points, in which case the step

Algorithm 5.2 Sammon mapping

Input: data $\mathbf{X}$, dimension q, step size α, maximum number of iterations, tolerance ϵ

compute the pairwise distances d_{kl} and λ;
initialize $\mathbf{Y}$ (PCA scores or random), center it;
repeat
  compute d^*_{kl} for the current $\mathbf{Y}$;
  for *every point k and coordinate l* **do**
    evaluate the derivatives (5.4)–(5.5) and the step $\Delta y_{kl} = -\alpha\,(\partial E/\partial y_{kl})/\left|\partial^2 E/\partial y_{kl}^2\right|$;
  move all points simultaneously, $\mathbf{Y} \leftarrow \mathbf{Y} + \Delta\mathbf{Y}$ (halving $\Delta\mathbf{Y}$ while E would increase), re-center;
  compute the stress (5.2);
until *the relative change of E is below ϵ*;

Output: embedding $\mathbf{Y}$, stress E

In a nutshell Sammon mapping

- Nonlinear projection that preserves the $N(N-1)/2$ pairwise Euclidean distances of the samples; small distances weigh more.
- Iterative diagonal Newton descent of the stress E; step size $\alpha \approx 0.3$–0.4, initialize from PCA.
- Cost quadratic in N per iteration; local minima; no mapping of new samples.
- Shows the geometry of the *data*; knows nothing about the prototypes or the norm of the clustering algorithm.

throws them far away; `fuzzyclust` therefore clips outlying displacements and halves the step when the stress would increase, which in practice keeps the stress history monotone (the raw update is available with `backtracking=False`). The stress surface has local minima, so the initial configuration matters: random starts give different maps, whereas the principal component scores give a deterministic and usually good start [54]. Two properties limit the method for clustering: every iteration needs all $N(N-1)/2$ distances, which becomes impractical for large N [55], and there is no out-of-sample extension—a new point requires a new optimization [56]. Prototypes are not part of the mapping; for Euclidean clusterings they can be placed at the weighted means of the projected data afterward.

5.2.3 Fuzzy Sammon mapping (FUZZSAM)

A fuzzy clustering algorithm minimizes $J = \sum_i \sum_k \mu_{ik}^m D_{ik}^2$: only the distances between samples and prototypes matter, and each of them matters in proportion to the membership. The fuzzy Sammon mapping [49, 50, 51] carries this structure over to the projection. It looks for projected samples $\mathbf{y}_k$ and projected centers $\mathbf{z}_i$ such that

$$E_{\text{fuzz}} = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \left(D_{ik} - d_{ik}^*\right)^2, \qquad d_{ik}^* = \|\mathbf{y}_k - \mathbf{z}_i\|, \tag{5.6}$$

is minimal, where $D_{ik} = d(\mathbf{x}_k, \mathbf{v}_i)$ is the distance *of the clustering algorithm*. Three things change with respect to (5.2). First, only Nc distances enter, so an iteration costs $\mathcal{O}(Nc)$ instead of $\mathcal{O}(N^2)$. Second, the norm of the algorithm is built in: a Gustafson–Kessel cluster, whose points are all at a small Mahalanobis-type distance from the prototype, is mapped to a compact round group around $\mathbf{z}_i$, whatever its orientation and elongation in $\mathbb{R}^n$; a line or shell prototype becomes an ordinary point. Third, the weights μ_{ik}^m make the map accurate where it matters: the distance of a sample to a cluster it hardly belongs to need not be reproduced. The derivatives of (5.6) with respect to the coordinates of $\mathbf{y}_k$, with the $\mathbf{z}_i$ held fixed, are the analogs of (5.4)–(5.5):

$$\frac{\partial E_{\text{fuzz}}}{\partial y_{kl}} = -2 \sum_{i=1}^{c} \mu_{ik}^m \frac{D_{ik} - d_{ik}^*}{d_{ik}^*} \left(y_{kl} - z_{il}\right), \tag{5.7}$$

$$\frac{\partial^2 E_{\text{fuzz}}}{\partial y_{kl}^2} = -2 \sum_{i=1}^{c} \frac{\mu_{ik}^m}{d_{ik}^*} \left[\left(D_{ik} - d_{ik}^*\right) - \frac{\left(y_{kl} - z_{il}\right)^2}{d_{ik}^*} \left(1 + \frac{D_{ik} - d_{ik}^*}{d_{ik}^*}\right) \right], \tag{5.8}$$

and the points are moved by the same diagonal Newton step (5.3). After every sweep the projected centers are recomputed as the weighted means of the projected data, in analogy with the prototype update of the clustering algorithms,

$$\mathbf{z}_i = \frac{\sum_{k=1}^{N} \mu_{ik}^m \, \mathbf{y}_k}{\sum_{k=1}^{N} \mu_{ik}^m}. \tag{5.9}$$

The first version of the method [49] weighted with μ_{ik} instead of μ_{ik}^m in (5.6) and (5.9); the later papers and `fuzzyclust` use μ_{ik}^m (parameter `weight_exponent` selects either). The initial configuration is the PCA projection of the data, or, as in the original case studies, a classical Sammon map; from it the $\mathbf{z}_i$ follow by (5.9).

In the plane every cluster is represented by a point and the distances are Euclidean, so the picture can be read with the FCM membership formula: the memberships implied by the picture are

$$\mu_{ik}^* = \frac{1}{\sum_{j=1}^{c} \left(d_{ik}^* / d_{jk}^*\right)^{2/(m-1)}}, \tag{5.10}$$

and their level curves $\max_i \mu_i^*(\mathbf{y})$, drawn over the plane, are the contours of fig. 5.1. For two clusters they are exact circles (each level set of d_{1k}^*/d_{2k}^* is an Apollonius

Algorithm 5.3 Fuzzy Sammon mapping (FUZZSAM)

Input: fitted clustering: partition $\mathbf{U}$, distances $D_{ik} = d(\mathbf{x}_k, \mathbf{v}_i)$ in the norm of the algorithm, exponent m; dimension q, step size α, maximum number of iterations, tolerance ϵ

initialize $\mathbf{Y}$ by the PCA scores of $\mathbf{X}$ (or a Sammon map, or at random), center it;

$\mathbf{z}_i \leftarrow \sum_k \mu_{ik}^m \mathbf{y}_k / \sum_k \mu_{ik}^m$ for $i = 1, \ldots, c$;

repeat

  $d_{ik}^* \leftarrow \|\mathbf{y}_k - \mathbf{z}_i\|$ for all i, k;

  for *every sample k and coordinate l* **do**

    evaluate (5.7)–(5.8) and the step $\Delta y_{kl} = -\alpha \left(\partial E_{\text{fuzz}} / \partial y_{kl}\right) / \left|\partial^2 E_{\text{fuzz}} / \partial y_{kl}^2\right|$;

  $\mathbf{Y} \leftarrow \mathbf{Y} + \Delta\mathbf{Y}$ (halving $\Delta\mathbf{Y}$ while E_{fuzz} would increase), re-center;

  recompute the projected centers $\mathbf{z}_i$ by (5.9);

  compute E_{fuzz} by (5.6);

until *the relative change of E_{fuzz} is below ϵ*;

Output: embedding $\mathbf{Y}$, projected centers $\mathbf{Z}$, E_{fuzz}; recomputed memberships $\mathbf{U}^*$ by (5.10)

In a nutshell Fuzzy Sammon mapping

- Projects a *fitted fuzzy clustering*: preserves the $N \times c$ sample–prototype distances of the algorithm, weighted by μ_{ik}^m.
- Works with any prototype and norm (FCM, GK, GG, lines, shells); every cluster becomes a point with nearly circular membership contours.
- $\mathcal{O}(Nc)$ per iteration; same Newton step as Sammon ($\alpha \approx 0.4$); centers recomputed as weighted means after every sweep.
- Distorted or overlapping groups in the plane indicate a wrong prototype or a wrong number of clusters; the recomputed memberships $\mathbf{U}^*$ quantify the fidelity.

circle); for more clusters they are close to circles near a projected center and deform where the regions of neighboring clusters meet. If the prototype suits the data, the projected cluster is a compact group inside its contours; a cluster that spills across the contours, or two clusters whose contours interpenetrate, point to an inadequate prototype or a wrong number of clusters. A smaller fuzziness exponent m is visible too: the partition is crisper and the contours of high membership enclose more points.

5.2.4 Quality of a projection

Any two-dimensional picture is an approximation. For a projection of a fuzzy clustering the natural question is whether the memberships a viewer reads off the picture agree with the fitted ones. With the recomputed partition $\mathbf{U}^* = [\mu^*_{ik}]$ of (5.10), evaluated for the projected data and centers of whatever method produced them, the FUZZSAM papers use three numbers:

$$P = \frac{1}{Nc} \sum_{i=1}^{c} \sum_{k=1}^{N} |\mu_{ik} - \mu^*_{ik}| \,, \tag{5.11}$$

$$F = \frac{1}{N} \sum_{i=1}^{c} \sum_{k=1}^{N} \mu^2_{ik}, \qquad F^* = \frac{1}{N} \sum_{i=1}^{c} \sum_{k=1}^{N} (\mu^*_{ik})^2, \tag{5.12}$$

the mean absolute membership error and the partition coefficients [3] of the fitted and of the recomputed partition. $P = 0$ means that the picture shows exactly the fitted partition; $F^* \gg F$ means that the picture looks much crisper than the clustering really is (the typical failure of PCA, which pushes the samples apart from the wrong centers). The Sammon stress E of (5.2), evaluated for the embedding against the Euclidean distances of the data, completes the set; it measures how well the geometry of the *data* is kept, which is a different goal. In the published case studies FUZZSAM has by far the smallest P and an F^* closest to F, while classical Sammon mapping has the smallest E—each method is best at what it optimizes. Other validity indices computed from $\mathbf{U}$ and $\mathbf{U}^*$ can be compared in the same way.

5.2.5 Fuzzy self-organizing map

The methods so far project a finished clustering. Kohonen's self-organizing map [57] takes the opposite route: the prototypes (code vectors) sit on the nodes of a regular low-dimensional grid and are trained so that neighboring nodes hold similar prototypes; the grid thus becomes a topology-preserving map of the data. The classical SOM is a heuristic competitive-learning rule with a shrinking neighborhood and no objective function; several authors have therefore reformulated it as a modification of fuzzy c-means [58, 59]. Abonyi, Migály, and Szeifert [60] add a smoothness penalty to the FCM functional:

$$\begin{aligned} J(\mathbf{X}; \mathbf{U}, \mathbf{V}) &= \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^{m} \, \|\mathbf{x}_k - \mathbf{v}_i\|^2 + \vartheta \sum_{i=1}^{c} \left\| \frac{\partial^2 \mathbf{v}_i}{\partial \mathbf{r}^2} \right\|^2 \\ &\approx \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^{m} \, \|\mathbf{x}_k - \mathbf{v}_i\|^2 + \vartheta \, \operatorname{tr}\!\left(\mathbf{V}^{\mathsf{T}} \mathbf{L} \mathbf{V}\right), \end{aligned} \tag{5.13}$$

with $\vartheta \geq 0$, where $\mathbf{r} \in \mathbb{R}^s$ is the position of a node on the s-dimensional grid ($s = 1$: a chain, $s = 2$: a sheet) and $\mathbf{L} = \mathbf{G}^{\mathsf{T}}\mathbf{G}$ with $\mathbf{G} = [\mathbf{G}_1; \ldots; \mathbf{G}_s]$ stacks the second-order finite difference operators along the grid axes: for every interior node i of axis a

the row of $\mathbf{G}_a$ reads $\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}$ (fig. 5.2). The penalty is thus the sum of the squared discrete curvatures of the map; an affine (planar, regularly spaced) grid is not penalized at all, so the penalty orders the prototypes without pulling them together. Replacing the second-order by first-order differences gives the "smoothly distributed FCM" of [59], which penalizes the edge lengths and therefore contracts the map toward the data center; the fuzzy curve-tracing algorithm of [61] is the special case of a chain with second-order differences. For $\vartheta = 0$ plain FCM with c unordered prototypes is recovered.

The membership update is that of FCM, since the penalty does not involve $\mathbf{U}$. For fixed $\mathbf{U}$ the prototypes are coupled through $\mathbf{L}$; setting the gradient of (5.13) to zero gives the linear system

$$\mathbf{v}_i \sum_{k=1}^{N} \mu_{ik}^m + \vartheta \sum_{j=1}^{c} L_{ij}\mathbf{v}_j = \sum_{k=1}^{N} \mu_{ik}^m \mathbf{x}_k, \qquad i = 1, \dots, c,$$

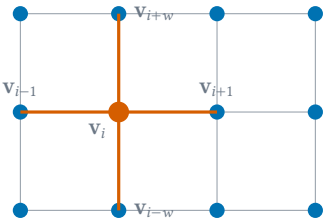
$$\text{i.e.} \qquad \Big(\operatorname{diag}\big(\textstyle\sum_k \mu_{ik}^m\big) + \vartheta\mathbf{L}\Big)\mathbf{V} = (\mathbf{U}^m)\mathbf{X}, \tag{5.14}$$

where $\mathbf{U}^m$ denotes the element-wise power of the $c{\times}N$ partition matrix. The system has a unique solution because the left matrix is the sum of a positive diagonal and a positive semidefinite matrix. `fuzzyclust` solves it exactly in every iteration; the original paper iterates its row-wise form $\mathbf{v}_i = \big(\sum_k \mu_{ik}^m \mathbf{x}_k - \vartheta \sum_{j\neq i} L_{ij}\mathbf{v}_j\big) / \big(\sum_k \mu_{ik}^m + \vartheta L_{ii}\big)$, whose fixed point is the same. The alternating optimization of the base class (chapter 2) therefore applies unchanged with a modified prototype step. The prototypes are initialized on a regular grid in the subspace of the first s principal components, which already gives an ordered start; the regularization then keeps the order while the map folds into the data.

Once fitted, the map is read like a SOM. The *U-matrix* shows for every node the mean distance of its prototype to the prototypes of its grid neighbors: ridges of large distance separate groups of similar prototypes, i.e. clusters of the data. A *component plane* shows one coordinate of the prototypes over the grid and reveals which variables drive the organization. The *fuzzy hit histogram* $h_i = \sum_k \mu_{ik}$ shows where the data (or a subset, e.g. one class) fall on the map. A smaller exponent m sharpens these views when many prototypes are fitted in high dimension, where the FCM memberships for $m = 2$ tend to become uniform.

5.3 Using it in fuzzyclust

All methods live in `fuzzyclust.viz`. FUZZSAM accepts any fitted `fuzzyclust` clusterer and obtains the distances in the norm of the algorithm from the `transform` method of the clusterer.



rows of $\mathbf{G}$ at node i:
$\mathbf{G}_1$: $\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}$
$\mathbf{G}_2$: $\mathbf{v}_{i-w} - 2\mathbf{v}_i + \mathbf{v}_{i+w}$
penalty $\vartheta\ \mathrm{tr}(\mathbf{V}^\mathsf{T}\mathbf{G}^\mathsf{T}\mathbf{G}\mathbf{V})$ $= \vartheta \sum \|\text{rows}\|^2$, zero for a regular planar grid

grid of $c = w \times h$ nodes in $\mathbb{R}^s$ $(s = 2)$; every node carries a prototype $\mathbf{v}_i \in \mathbb{R}^n$

Figure 5.2. The regularization of the fuzzy self-organizing map. The prototypes are indexed by the nodes of a grid; at every interior node the second-order differences along the two grid axes (red stencil) form the rows of $\mathbf{G}$, and $\vartheta\ \mathrm{tr}(\mathbf{V}^\mathsf{T}\mathbf{L}\mathbf{V})$ with $\mathbf{L} = \mathbf{G}^\mathsf{T}\mathbf{G}$ penalizes the curvature of the map. The membership update of FCM is unchanged; the prototype update becomes the linear system (5.14).

Algorithm 5.4 Fuzzy self-organizing map (regularized FCM)

Input: data $\mathbf{X}$, grid shape (c nodes in s dimensions), exponent m, regularization weight ϑ, tolerance ϵ

1 build the difference operator $\mathbf{G}$ of the grid and $\mathbf{L} = \mathbf{G}^\mathsf{T}\mathbf{G}$;
2 initialize the prototypes on a regular grid spanning the first s principal components; derive $\mathbf{U}^{(0)}$ from the Euclidean distances;
3 **repeat**
4 prototypes: solve the linear system (5.14) for $\mathbf{V}$;
5 distances: $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$;
6 memberships: $\mu_{ik} = 1/\sum_j (D_{ik}/D_{jk})^{2/(m-1)}$;
7 **until** $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: ordered prototypes $\mathbf{V}$, partition $\mathbf{U}$, grid positions; U-matrix, component planes, hit histogram

In a nutshell Fuzzy self-organizing map

- FCM whose prototypes live on a grid and are coupled by a curvature penalty $\vartheta\ \mathrm{tr}(\mathbf{V}^\mathsf{T}\mathbf{L}\mathbf{V})$; an exact objective is minimized by alternating optimization.
- $\vartheta = 0$: plain FCM, unordered; moderate ϑ: ordered, topology-preserving map; very large ϑ: flat grid that ignores the curvature of the data.
- Second-order differences (default) order without contracting; first-order differences shrink the map toward the data center.
- Read the result like a SOM: U-matrix, component planes, hit histograms; use a smaller m for many nodes in high dimension.

`fuzzyclust.viz.pca_projection(X, n_components=2, centers=None)`

Returns `(Y, Z, pca)`: the projected data, the projected prototypes (or `None`) and the fitted `sklearn.decomposition.PCA` object (`explained_variance_ratio_`, `transform` for further data).

`fuzzyclust.viz.Sammon`

`n_components=2, max_iter=100, alpha=0.3, init="pca", tol=1e-6, backtracking=True, max_backtracks=10, random_state=None.`
`fit(X)`, `fit_transform(X)`; attributes `embedding_`, `stress_`, `stress_history_`, `n_iter_`. `init` may also be an array (e.g. a previous embedding). `transform` raises `NotImplementedError`: Sammon mapping has no out-of-sample extension. The function `sammon_stress(X, Y)` evaluates (5.2) for any embedding.

`fuzzyclust.viz.FuzzySammon`

`n_components=2, max_iter=100, alpha=0.4, init="pca", tol=1e-6, backtracking=True, max_backtracks=10, m=None, weight_exponent=None, random_state=None.`
`fit(estimator, X)` takes a fitted clusterer and the data (memberships and distances are re-evaluated on `X`, so any data the estimator can score may be mapped); alternatively `fit(membership=U, distances=D)` with explicit arrays. Attributes `embedding_`, `embedded_centers_`, `stress_`, `stress_history_`, `n_iter_`, `membership_`, `distances_`, `m_` (the exponent used); methods `recomputed_membership()` and `quality(X=None)` returning P, F, F^* (and E). `m=None` takes the exponent of the estimator (2.0 if it has none); `weight_exponent=1.0` selects the μ_{ik}-weighted first variant.

`fuzzyclust.viz.recomputed_membership(Y, Z, m=2.0) / projection_quality(U, Y, Z, m=2.0, X=None)`

(5.10) and the dictionary `{"P", "F", "F_star", "E"}` of (5.11)–(5.12); `E` is included when `X` is given. Works for any projection, so PCA, Sammon, and FUZZSAM maps are compared on the same footing.

`fuzzyclust.viz.FuzzySOM`

`grid_shape=(5, 5), m=2.0, regularization=1.0, difference_order=2, tol=1e-4, max_iter=300, init="pca", n_init=1, random_state=None, verbose=0.`
A `BaseFuzzyClusterer`: `fit`, `predict`, `predict_proba`, `transform`, `cluster_centers_`, `membership_`, `labels_`, `objective_history_`;

the number of nodes is `n_clusters_ = prod(grid_shape)`. Map views: `grid_positions_`, `regularization_matrix_`, `u_matrix()`, `component_planes()`, `hits(X=None)`, `neighbor_distance()`. `grid_laplacian(grid_shape, order=2)` returns $\mathbf{L}$ and the node positions without fitting a map.

plot helpers: plot_projection, plot_stress, plot_map, plot_som_lattice

`plot_projection(Y, Z, labels, U, contours=True, m, levels, misclassified, label_prefix)` draws the embedding with the projected centers and the level curves of $\max_i \mu_i^*$; `plot_stress` the stress histories; `plot_map(som, kind="umatrix"|"component"|"hits", feature, X)` a fuzzy SOM as a colored grid; `plot_som_lattice(som, X, dims)` the prototypes in the data space with grid neighbors connected. All four take `ax=None` and return the `Axes`.

A typical session projects a Gustafson–Kessel result with PCA and FUZZSAM and compares the two pictures by their quality measures:

```
Y_pca, Z_pca, _ = pca_projection(X, 2, centers=gk.cluster_centers_)
fsam_fcm = FuzzySammon(n_components=2, max_iter=100, alpha=0.4,
    init="pca").fit(fcm, X)
fsam_gk = FuzzySammon(n_components=2, max_iter=100, alpha=0.4,
    init="pca").fit(gk, X)
quality_pca = projection_quality(gk.membership_, Y_pca, Z_pca, m=gk.m, X=X)
quality_gk = fsam_gk.quality(X)
```

and reproduces the three-way comparison of the papers on the wine data:

```
Y_pca, Z_pca, _ = pca_projection(Xw, 2, centers=fcm_w.cluster_centers_)
sammon = Sammon(max_iter=100, alpha=0.4, init=Y_pca).fit(Xw)
Um = fcm_w.membership_**2
Z_sam = (Um.T @ sammon.embedding_) / Um.sum(axis=0)[:, None]
fsam_w = FuzzySammon(max_iter=100, alpha=0.4,
    init=sammon.embedding_).fit(fcm_w, Xw)

rows = {
    "PCA": projection_quality(fcm_w.membership_, Y_pca, Z_pca, m=2.0,
    X=Xw),
    "Sammon": projection_quality(fcm_w.membership_, sammon.embedding_,
    Z_sam, m=2.0, X=Xw),
    "FUZZSAM": fsam_w.quality(Xw),
}
table = pd.DataFrame(rows).T.rename(columns={"F_star": "F*"})
```

A fuzzy self-organizing map is fitted like any other clusterer:

```
maps = {th: FuzzySOM(grid_shape=(6, 6), regularization=th,
    random_state=0).fit(Xh) for th in (0.0, 0.5, 5.0)}
```

5.4 Examples

5.4.1 PCA versus Sammon mapping

Five Gaussian clusters in five dimensions are projected by PCA and by Sammon mapping started from the PCA scores (fig. 5.3). The PCA plane keeps 76 % of the variance but its stress is twice that of the Sammon map; the groups that overlap in the PCA picture are separated once the distances rather than the variance are preserved. Figure 5.4 shows the convergence of the stress for several step sizes and for random initial configurations: a larger α converges in fewer iterations, and random starts need many more iterations than the PCA start and end in different local minima of comparable stress.

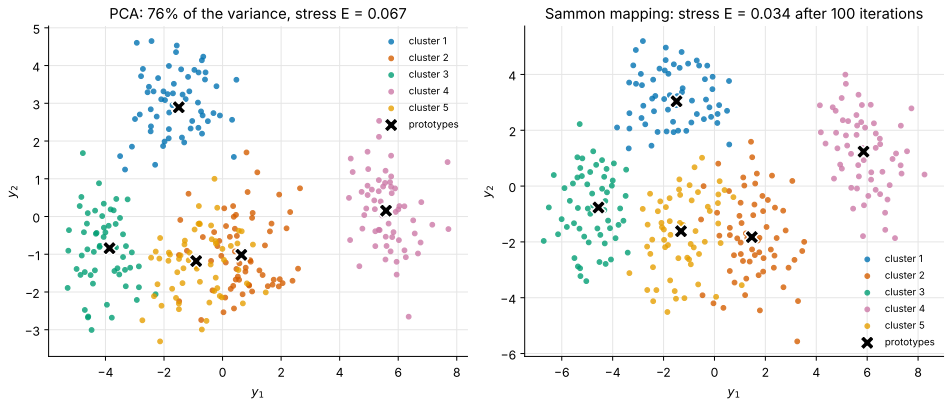


Figure 5.3. PCA keeps the directions of largest spread and lets clusters that differ in the discarded directions overlap; the Sammon mapping started from the same configuration bends the picture until the inter-point distances match, which separates the five groups and halves the stress.

5.4.2 Fuzzy Sammon mapping

Figure 5.5 illustrates the central property of FUZZSAM on three elongated Gaussian clusters in four dimensions. In the PCA picture of the Gustafson–Kessel result the memberships recomputed in the plane disagree with the fitted ones ($P = 0.078$). FUZZSAM of the FCM result reproduces its memberships closely ($P = 0.013$), but the groups spill into each other's contours, because the spherical prototype does not fit these data. FUZZSAM of the GK result shows three separated groups with round contours, since the GK norm already sees the ellipsoids as spheres and the picture inherits this view; the tails of the groups reach into the neighboring

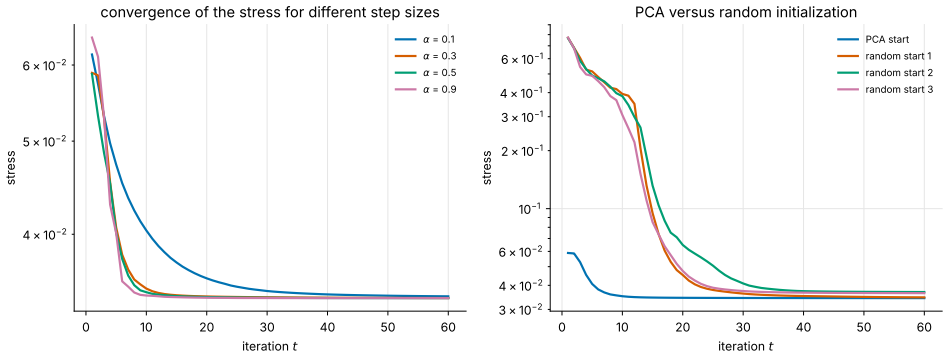


Figure 5.4. Convergence of Sammon's stress. Left: a larger step size α lowers the stress faster; with the safeguarded step none of the runs diverges. Right: the PCA start is already close to a minimum; random starts take longer and reach different, slightly worse minima.

contours, so its $P = 0.027$ is higher than that of FCM. The distortion of a FUZZSAM picture is thus a diagnostic of the prototype.

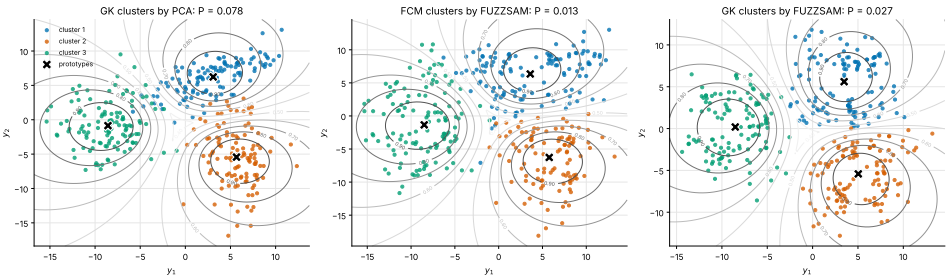


Figure 5.5. Three elongated clusters in four dimensions. Left: the GK partition projected by PCA, with the contours of the recomputed memberships. Middle: FUZZSAM of the FCM partition, whose spherical prototypes overlap. Right: FUZZSAM of the GK partition; the groups are separated and their contours are round, with tails reaching into the neighboring contours.

Figure 5.6 repeats the comparison of the papers on the wine data (178 wines, 13 range-scaled attributes, FCM with $c = 3$). Table 5.1 lists the quality measures: P drops from 0.13 (PCA) through 0.09 (Sammon) to 0.04 (FUZZSAM), and the partition coefficient read off the FUZZSAM picture ($F^* = 0.52$) is close to the fitted one ($F = 0.50$), whereas PCA suggests a far crisper partition ($F^* = 0.74$). The Sammon stress ranks the methods differently: classical Sammon mapping is best at preserving the data geometry, FUZZSAM at preserving the clustering. The circled samples are wines whose cluster does not match their cultivar (5 % of the data); they lie in the fuzzy region between the centers in every picture, but only the FUZZSAM picture places them where their memberships say.

The fuzziness of a partition is visible as well. Figure 5.7 shows the GK clustering of the iris data for $m = 2$ and $m = 1.4$, drawn with the same membership levels:

Table 5.1. Quality of three projections of the FCM partition ($c = 3$, $m = 2$) of the range-scaled wine data: mean membership error P, partition coefficient of the fitted (F) and of the recomputed (F^*) partition, Sammon stress E.

	P	F	F^*	E
PCA	0.130	0.503	0.742	0.130
Sammon	0.087	0.503	0.656	0.057
FUZZSAM	0.036	0.503	0.522	0.096

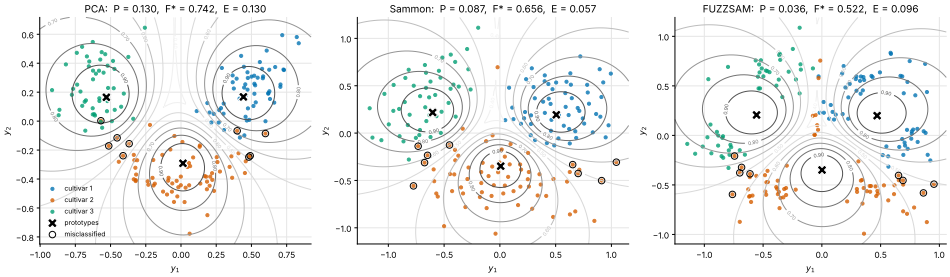


Figure 5.6. The FCM partition of the wine data projected by PCA, Sammon mapping, and FUZZSAM; colors are the cultivars, contours the recomputed membership levels, circles the wines whose cluster disagrees with their cultivar. Only in the FUZZSAM picture do the contours agree with the point clouds.

with the smaller exponent the partition is crisper ($F = 0.94$ against 0.73), and the curves of high membership expand and enclose almost all points of each cluster.

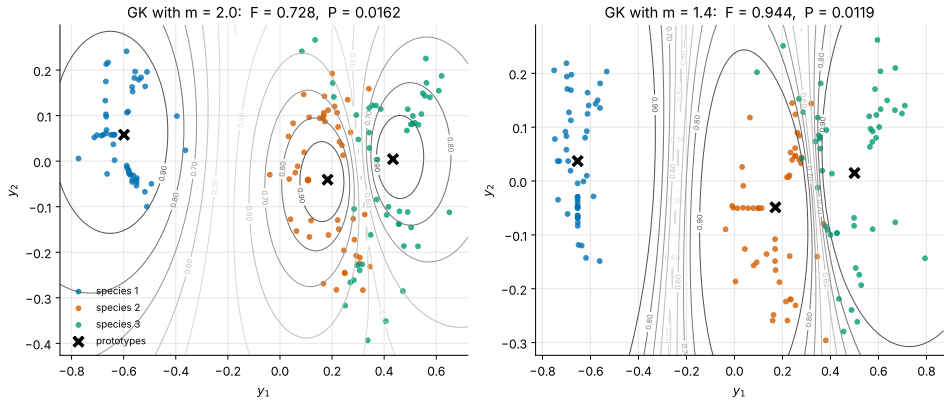


Figure 5.7. FUZZSAM pictures of the GK clustering of the iris data with $m = 2$ (left) and $m = 1.4$ (right), drawn with the same membership levels. The crisper partition shows up as wide contours of high membership around compact groups.

5.4.3 Fuzzy self-organizing map

The two geometric case studies of [60] are reproduced. In fig. 5.8 seven prototypes on a one-dimensional grid are fitted to 300 noisy points on one turn of a helix. Plain FCM ($\vartheta = 0$) finds seven groups along the curve, but the chain connecting consecutive nodes is tangled; with $\vartheta = 2$ the chain follows the curve. Figure 5.9 fits a 6×6 grid to 900 points on a hemisphere: without regularization the lattice is a tangle of crossing edges; with $\vartheta = 0.5$ it is a regular net folded onto the surface, as the component plane of the third coordinate in fig. 5.10 confirms; and with $\vartheta = 5$ the net is nearly planar and no longer follows the curvature. The Spearman correlation between grid distances and prototype distances, printed in the panel titles, rises from 0.93 for $\vartheta = 0$ to 0.98 for $\vartheta = 0.5$ and 0.99 for $\vartheta = 5$, while the objective grows from 6.76 to 7.43 and 8.36. Figure 5.11 compares the two difference operators: with first-order differences the extent of the map shrinks with ϑ and collapses to a point for $\vartheta = 2$; with second-order differences it stays constant, which is the argument for the curvature penalty.

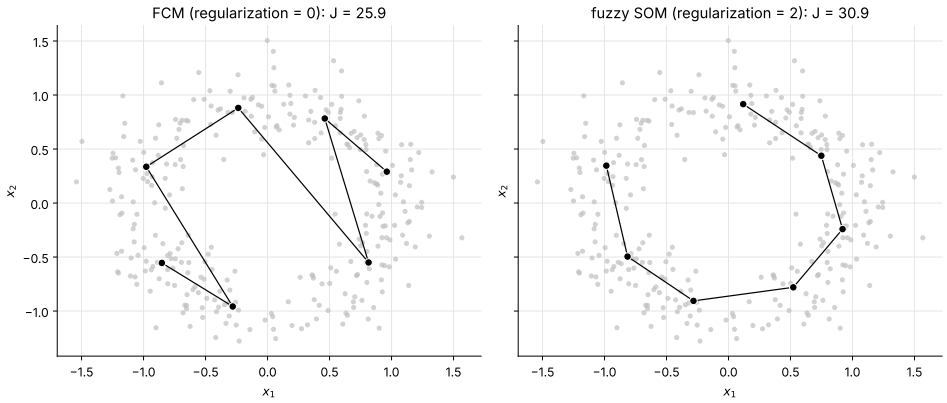


Figure 5.8. A chain of seven prototypes on a noisy helix. Left: fuzzy c-means places the prototypes along the curve but connects consecutive nodes in a tangled order. Right: with the curvature penalty ($\vartheta = 2$) the chain follows the curve.

Figures 5.12 and 5.13 show a 6×6 map of the standardized wine data ($m = 1.3$, $\vartheta = 0.5$). The U-matrix has dark ridges that divide the grid into three regions, the hit histograms of the three cultivars fall into exactly these regions, and the component planes of the flavanoid content and the color intensity show the chemistry behind the split.

5.4.4 Visual validation of the number of clusters

Figure 5.14 applies FUZZSAM to the FCM partitions of the Wisconsin breast cancer data (569 samples, ten standardized features) for $c = 2, 3, 4$. With two clusters the groups are compact and the diagnosis coincides with the clusters up to a band of ambiguous cases between the centers; with three clusters the extra prototype

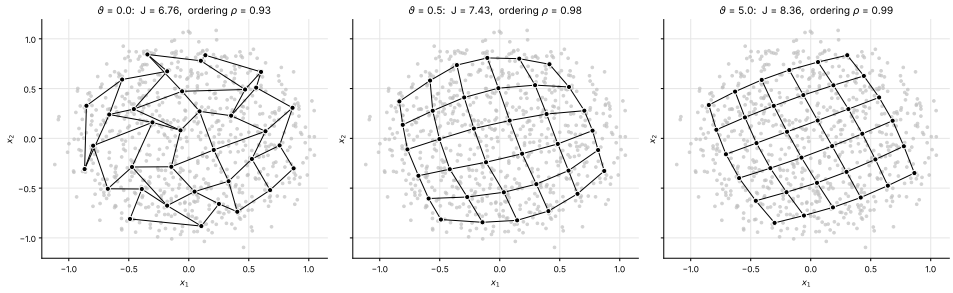


Figure 5.9. A 6×6 fuzzy SOM of points on a hemisphere, lattice drawn in the (x_1, x_2) plane. $\vartheta = 0$ is plain FCM and gives a tangled lattice; $\vartheta = 0.5$ gives an ordered net that follows the surface; $\vartheta = 5$ flattens the net.

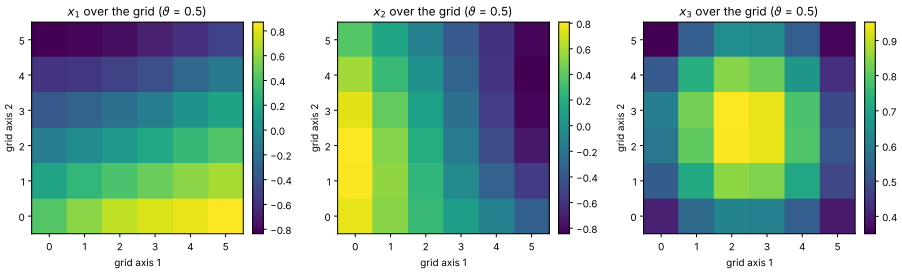


Figure 5.10. Component planes of the ordered map ($\vartheta = 0.5$): the first two coordinates change monotonically across the grid, each in its own direction; the third (height on the hemisphere) peaks in the middle of the grid, so the map has folded onto the surface.

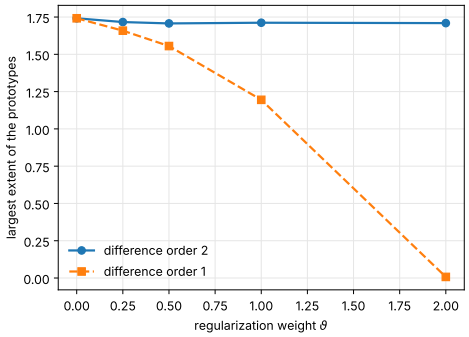


Figure 5.11. Largest extent of the prototypes as a function of ϑ. The first-order penalty of the smoothly distributed FCM contracts the map toward the data center and collapses it for $\vartheta = 2$; the second-order (curvature) penalty leaves a regular grid unpenalized and keeps the extent of the map.

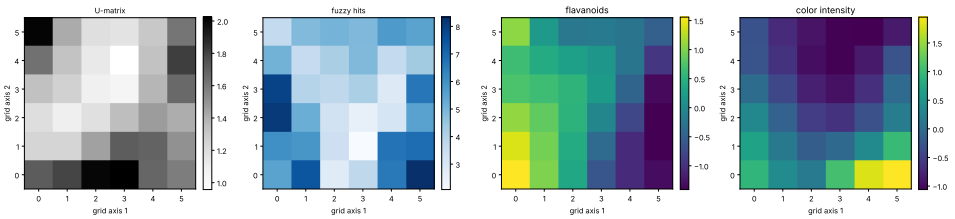


Figure 5.12. A 6×6 fuzzy SOM of the wine data: U-matrix (dark ridges separate groups of similar prototypes), fuzzy hit histogram, and the component planes of two attributes.

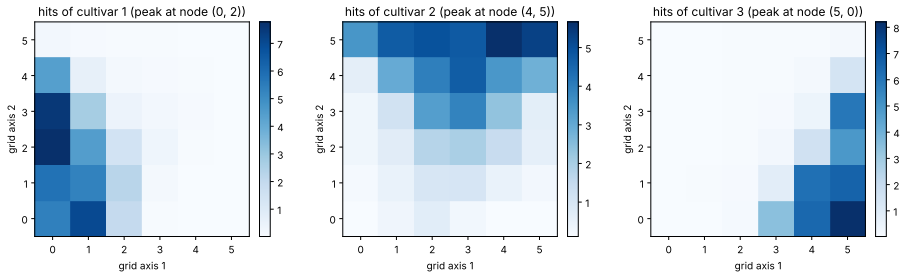


Figure 5.13. Hit histograms of the three cultivars on the same map: each occupies its own region, separated by the ridges of the U-matrix.

splits the large benign group, and with four the malignant group is split as well; the new clusters overlap in the plane, and the recomputed memberships fit worse (P grows from 0.004 to 0.02). The validity indices PC, CE, and XB also favor two clusters (SC, which decreases with c by design, does not), but only the picture shows *which* clusters were split and where the ambiguous samples lie. Figure 5.15 shows the convergence of E_{fuzz} for several step sizes; because only Nc distances enter, sixty iterations on 569 samples take a fraction of a second.

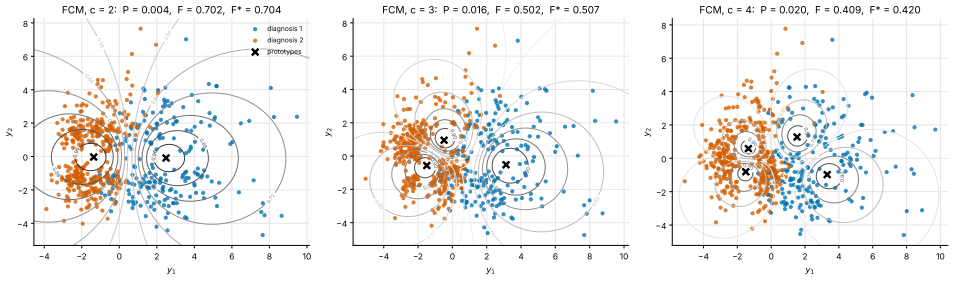


Figure 5.14. FUZZSAM pictures of FCM partitions of the breast cancer data with two, three, and four clusters (colors: diagnosis; diagnosis 1 is malignant, diagnosis 2 benign). Two compact groups for $c = 2$; for $c = 3$ the additional prototype splits the benign group, for $c = 4$ each diagnosis is split in two, and the clusters overlap.

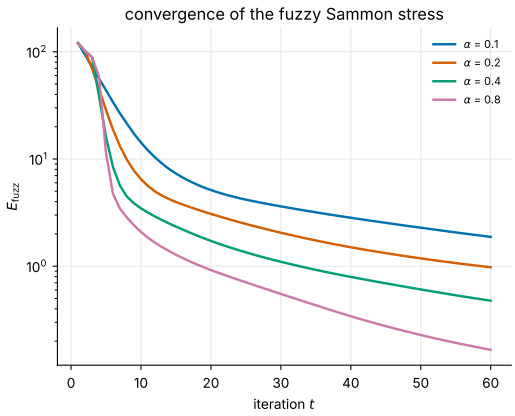


Figure 5.15. Convergence of the fuzzy Sammon stress for different step sizes; a larger α speeds up the decrease, and the safeguarded step keeps every run monotone.

5.5 Further reading

Sammon's original paper [52] remains the clearest account of the stress and of the diagonal Newton step; Mao and Jain [54] and de Ridder and Duin [55] discuss initialization, neural-network implementations, and the computational burden, Pal and Eluri [56] the missing out-of-sample extension. The fuzzy Sammon mapping was introduced by Kovács and Abonyi [49], presented with the Gustafson–Kessel examples and the quality measures by Abonyi and Babuška [50], and applied to the visualization of reconstructed phase-space trajectories of chaotic systems by Feil, Balaskó, and Abonyi [51], where the prototypes are local linear models and no classical projection would apply. For PCA see Jolliffe [53]; the partition coefficient used in F and F^* is Bezdek's [3]. The self-organizing map is Kohonen's [57]; its fuzzy reformulations include the fuzzy Kohonen clustering network [58], the smoothly distributed fuzzy c-means [59] and the fuzzy curve-tracing algorithm [61]; the last two are special cases of the regularized FCM of Abonyi, Migály, and Szeifert [60], obtained by the choice of $\mathbf{L}$. The SOM Toolbox [62] is the reference implementation of the U-matrix and component-plane views.

CHAPTER

6



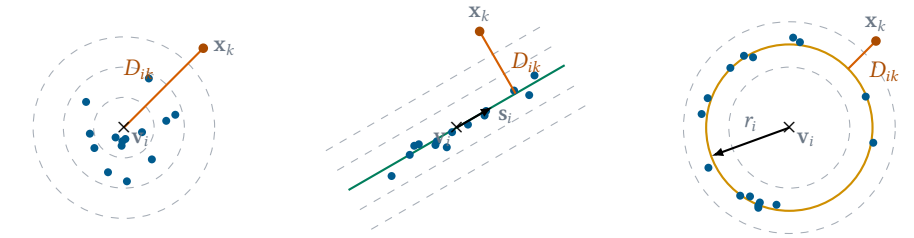
Robust and shape-oriented variants of fuzzy c-means

Fuzzy c-means rests on three assumptions: the memberships of every sample sum to one, every cluster is a point in the data space, and the distance to that point is Euclidean. This chapter relaxes them one at a time, each time by a small change to the FCM template of chapter 3—a different membership formula, distance, or prototype update—and all methods live in `fuzzyclust.extensions` with the common interface of the toolbox. Possibilistic c-means, possibilistic fuzzy c-means, and noise clustering drop or loosen the sum-to-one constraint, so that outliers are recognized instead of being shared out among the clusters. Kernel fuzzy c-means measures the distance in the feature space of a kernel, where rings and moons become spheres; fuzzy c-varieties and c-shells replace the point by a line or a circle; volume prototypes replace it by a core of zero distance, which removes the bias toward clusters of equal size and, with adaptive merging, finds the number of clusters during the iteration; and feature weights learned with the partition adapt the distance to the features that matter. The graph-based methods of the next chapter take the opposite route and abandon the prototype altogether.

6.1 Problem and idea

Two things can go wrong when fuzzy c-means meets real data. The first is the *partition*. The constraint $\sum_i \mu_{ik} = 1$ makes every membership a relative quantity: it says to which prototype a sample is closer, not whether it is close to any of them. A sample far from all prototypes still shares a total membership of one among them and, since the prototype update (3.7) weighs every sample by μ_{ik}^m, such an outlier pulls the prototypes toward itself. The second is the *prototype*. A point prototype with a fixed norm describes spherical clusters (section 3.2.3), one with

an adaptive norm ellipsoidal clusters (sections 3.2.4 and 3.2.5); a group of points along a line, a ring, or a curved arc has no useful center, and the algorithm tiles it into spherical or ellipsoidal pieces.



point (FCM, GK, GG): $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$

line / variety (FCV): $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2 - \left((\mathbf{x}_k - \mathbf{v}_i)^\mathsf{T}\mathbf{s}_i\right)^2$

shell (FCS, FCSS): $D_{ik}^2 = \left(\|\mathbf{x}_k - \mathbf{v}_i\| - r_i\right)^2$

Figure 6.1. Three prototypes and the distance each of them induces; the dashed curves are level sets of the distance. A point prototype measures how far a sample is from the center, so it detects compact groups. A linear variety—here a line through $\mathbf{v}_i$ with unit direction $\mathbf{s}_i$—measures only the orthogonal distance to the line, so samples anywhere along the line are equally close to it. A shell prototype, a center with a radius r_i, measures the radial deviation from the circle, so every point of a hollow ring is at zero distance from it. The membership formula of FCM is used unchanged with each of these distances.

The methods of this chapter attack both problems from inside the FCM template. Figure 6.1 shows the prototype side: a line and a shell are prototypes just as a point is, each with its own distance, and the alternating optimization of section 2.2.5 needs nothing but that distance and a rule to update the prototype from the memberships. A kernel reaches the same end by a different route: it keeps the point prototype but places it in a feature space in which the Euclidean distance is a curved function of the input coordinates, so that one sphere there can enclose a ring.

Figure 6.2 shows the partition side. Along the line through two prototypes, the FCM membership of either cluster crosses $1/2$ at the midpoint and, on the far side of its own prototype, *stays above* $1/2$ however far away the sample lies, because out there that prototype is always the closer one. A membership that expresses how typical a sample is of a cluster must instead decay with the distance to that cluster alone. This is the *possibilistic* membership of Krishnapuram and Keller [14], introduced in section 2.2.3 as the partition space M_{pc}. Decoupling the clusters has a price—they may coincide—and two further designs keep the coupling: the possibilistic fuzzy c-means [63] carries both kinds of membership, and Davé's noise clustering [64] keeps the sum-to-one constraint but adds one more cluster to share with, a *noise cluster* at the same distance δ from every sample (fig. 6.3). A sample farther than about δ from all real prototypes gives most of its membership to the noise cluster, and the real clusters receive only what is left.

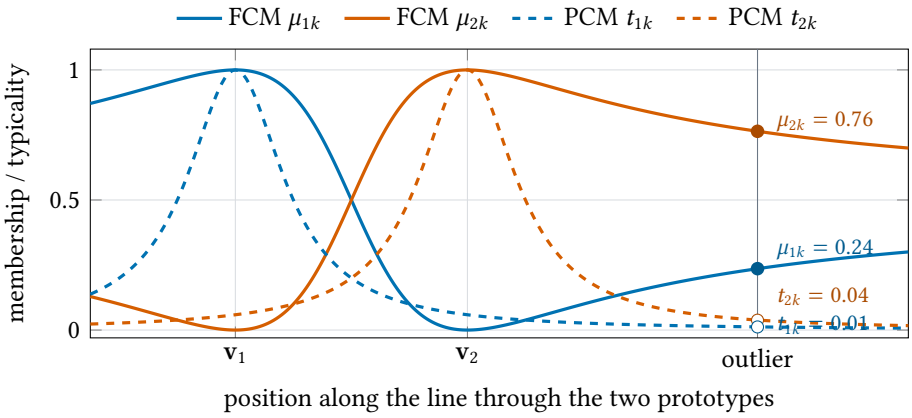


Figure 6.2. Relative versus absolute membership along the line through two prototypes at 0 and 4 (both with $m = 2$ and, for PCM, $\eta_i = 1$). The FCM memberships cross at the midpoint and decay only slowly beyond the prototypes: an outlier at 9 is a member of cluster 2 with membership 0.76 and shifts its prototype. The PCM typicalities are bumps of width $\sqrt{\eta_i}$ around the prototypes and decay to zero on both sides, so the outlier is atypical of both clusters and has almost no weight in either prototype.

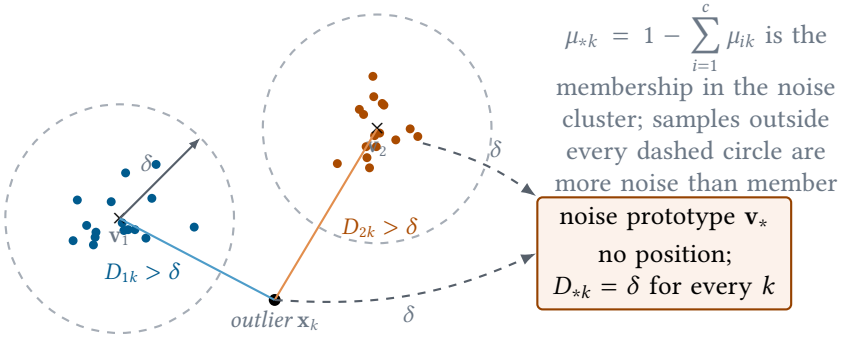


Figure 6.3. Noise clustering adds a fictitious prototype $\mathbf{v}_*$ that has no position but is at the same distance δ from every sample. The real memberships keep the FCM form and, together with the noise membership μ_{*k}, still sum to one; the dashed circles of radius δ mark where a sample is as close to the noise prototype as to a real one. The outlier is farther than δ from both real prototypes, so most of its membership goes to the noise cluster and it no longer moves $\mathbf{v}_1$ or $\mathbf{v}_2$, while the real clusters still compete for every sample near the groups.

6.2 Theory

The notation is that of the book: $\mathbf{X} \in \mathbb{R}^{N \times n}$ with rows $\mathbf{x}_k$, memberships μ_{ik} in $\mathbf{U} \in \mathbb{R}^{c \times N}$ (stored transposed in the code), prototypes $\mathbf{v}_i$ with index i or j, samples with index k, squared distances D_{ik}^2. The reference point throughout is fuzzy c-means (section 3.2.3): the functional $J = \sum_i \sum_k \mu_{ik}^m D_{ik}^2$ (3.3), the membership formula (3.6), and the prototype update (3.7). For each method we state which of the three it changes. Sections 6.2.1 to 6.2.3 change the partition, section 6.2.4 the distance, sections 6.2.5 and 6.2.6 the prototype; section 6.2.7 turns the point prototype into a volume and section 6.2.8 learns the distance as a weighting of the features.

6.2.1 Possibilistic c-means

Krishnapuram and Keller [14] drop the constraint $\sum_i \mu_{ik} = 1$. Without it the FCM functional is minimized by the trivial partition $\mu_{ik} = 0$, so a term is added that rewards memberships close to one, with a cluster-specific weight $\eta_i > 0$:

$$J(\mathbf{X}; \mathbf{T}, \mathbf{V}) = \sum_{i=1}^{c} \sum_{k=1}^{N} t_{ik}^m D_{ik}^2 + \sum_{i=1}^{c} \eta_i \sum_{k=1}^{N} (1 - t_{ik})^m. \tag{6.1}$$

The memberships are now called *typicalities* and written t_{ik}; the matrix $\mathbf{T}$ replaces $\mathbf{U}$. Setting the derivative with respect to t_{ik} to zero gives $m\, t_{ik}^{m-1} D_{ik}^2 = m\, \eta_i (1 - t_{ik})^{m-1}$, hence

$$t_{ik} = \frac{1}{1 + \left(D_{ik}^2 / \eta_i\right)^{1/(m-1)}}, \qquad \mathbf{v}_i = \frac{\sum_k t_{ik}^m \mathbf{x}_k}{\sum_k t_{ik}^m}. \tag{6.2}$$

The prototype update is (3.7) with typicalities in place of memberships; the typicality replaces (3.6). Three properties follow directly. First, t_{ik} depends on D_{ik}^2 and η_i only, not on the distances to the other prototypes: it is an absolute measure, $t_{ik} = 1/2$ exactly at $D_{ik}^2 = \eta_i$, and $\sqrt{\eta_i}$ is the radius of the zone of influence of cluster i. Second, m no longer controls the fuzziness of a partition but the *decay* of the typicality: for large distances $t_{ik} \approx (\eta_i / D_{ik}^2)^{1/(m-1)}$, so an outlier at three times the radius $\sqrt{\eta_i}$ keeps a typicality of about 0.1 with $m = 2$ and of about 0.01 with $m = 1.5$. The follow-up paper [65] therefore recommends m around 1.5 for PCM, smaller than the FCM default. Third, (6.1) is a sum of c terms, each involving only the typicalities $\mathbf{t}_i$ and the prototype $\mathbf{v}_i$ of one cluster: the clusters are optimized independently, PCM amounts to c separate mode-seeking procedures, and nothing prevents two of them from converging to the same dense region. These *coincident clusters* are the characteristic failure of PCM [65], and they are the reason the method is best used as a robust refinement of a partition that FCM has already found rather than as a stand-alone clusterer.

Algorithm 6.1 Possibilistic c-means (PCM)

Input: data $\mathbf{X}$, number of clusters c, exponent m (≈ 1.5), scale factor K (or scales η_i), tolerance ϵ

run FCM on $\mathbf{X}$ (same c and m) to obtain $\mathbf{U}$ and $\mathbf{V}$;
if *no scales are given* **then** compute η_i from $\mathbf{U}$ and $\mathbf{V}$ by (6.3);
initial typicalities $\mathbf{T}$ by (6.2) from the FCM distances;
repeat
    prototypes $\mathbf{v}_i$ by (6.2) with weights t_{ik}^m;
    distances $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$;
    typicalities t_{ik} by (6.2) with the fixed η_i;
until $\max_{ik} \left| t_{ik}^{(l)} - t_{ik}^{(l-1)} \right| < \epsilon$;

Output: prototypes $\mathbf{V}$, typicalities $\mathbf{T}$ (rows not normalized), scales η_i, J (6.1)

The scales η_i govern the result. The original paper estimates them from a probabilistic partition—in practice from an FCM run, which also initializes the typicalities—as the fuzzy intra-cluster mean squared distance,

$$\eta_i = K \frac{\sum_k \mu_{ik}^m D_{ik}^2}{\sum_k \mu_{ik}^m}, \qquad K = 1, \tag{6.3}$$

or alternatively as the mean squared distance of the samples whose membership exceeds an α-cut. The factor K widens ($K > 1$) or narrows ($K < 1$) the zones of influence: too large a K lets the outliers back in and draws the prototypes together, too small a K leaves the tails of a cluster atypical. The original paper already proposes an optional second phase: after convergence, the scales are re-estimated from the α-cut of the typicalities and PCM is run once more. The 1996 paper recommends keeping the scales fixed during the iterations, since re-estimating them from the typicalities shrinks the clusters. `fuzzyclust` follows this recommendation: η_i is computed once from the FCM partition with the factor `eta_scale` and then kept fixed; explicit scales can be passed through `eta`, which is also the way to run the second phase. Since the rows of $\mathbf{T}$ do not sum to one, `predict_proba` returns typicalities, and the hard label $\arg\max_i t_{ik}$ of a sample that is atypical everywhere means little; the maximal typicality $\max_i t_{ik}$ is a natural outlier score.

6.2.2 Possibilistic fuzzy c-means

Pal, Pal, Keller, and Bezdek [63] keep both descriptions. Every sample has a probabilistic membership μ_{ik} (constrained, $\sum_i \mu_{ik} = 1$) *and* a typicality t_{ik} (uncon-

In a nutshell Possibilistic c-means

- Drops the sum-to-one constraint: the typicality t_{ik} depends only on the distance to prototype i relative to the scale η_i; outliers are atypical of every cluster and do not move the prototypes.
- Same prototype update as FCM; the typicality formula (6.2) replaces the membership formula. m sets the decay of the typicality (1.5 is a good value), η_i the zone of influence, estimated once from an FCM run with a factor K.
- The clusters do not interact: two prototypes may coincide when the scales are too wide or the groups are close—initialize with FCM, keep $K \leq 1$, and use PCM as a robust refinement and outlier detector.

strained), and both enter the functional,

$$J = \sum_{i=1}^{c} \sum_{k=1}^{N} \left(a\, \mu_{ik}^{m} + b\, t_{ik}^{\eta}\right) D_{ik}^{2} + \sum_{i=1}^{c} \gamma_i \sum_{k=1}^{N} (1 - t_{ik})^{\eta}, \tag{6.4}$$

with weights $a, b \geq 0$ (not both zero), the FCM exponent $m > 1$ for the memberships, an exponent $\eta > 1$ for the typicalities (the paper's symbol; it plays the role of m in PCM), and scales γ_i as in PCM. The necessary conditions separate: the memberships follow the FCM formula (3.6), the typicalities the PCM formula with the scale γ_i / b, and the prototypes are weighted by both,

$$t_{ik} = \frac{1}{1 + \left(b\, D_{ik}^{2} / \gamma_i\right)^{1/(\eta-1)}}, \qquad \mathbf{v}_i = \frac{\sum_k \left(a\, \mu_{ik}^{m} + b\, t_{ik}^{\eta}\right) \mathbf{x}_k}{\sum_k \left(a\, \mu_{ik}^{m} + b\, t_{ik}^{\eta}\right)}. \tag{6.5}$$

With $b = 0$ the method is FCM, with $a = 0$ it is PCM. The membership term keeps the prototypes apart: every sample has to be shared, so two prototypes on the same group would each receive half of its membership mass, while the samples of the other group, which must give their membership to one of the two, pull one of them away. The typicality term keeps the outliers out of the prototype update. The ratio b/a sets the balance, and the paper recommends $b > a$ for noisy data. Two details matter in practice. First, the weight b also appears inside the typicality, so raising b alone *tightens* the zone of influence to γ_i / b instead of reproducing PCM; it is the membership weight a relative to the effective scale that decides whether the prototypes stay separated, and PFCM tolerates scales several times wider than PCM before clusters coincide (section 6.4.2). Second, with tight scales the typicalities of the core samples themselves fall below one, the weight $b\, t_{ik}^{\eta}$ of the robust term shrinks, and the membership term, which the outliers still contaminate, regains influence; the scales should therefore not be made tighter than the clusters. The scales are set as in PCM, $\gamma_i = K \sum_k \mu_{ik}^{m} D_{ik}^{2} / \sum_k \mu_{ik}^{m}$, from the FCM partition that also initializes the method.

Algorithm 6.2 Possibilistic fuzzy c-means (PFCM)

Input: data $\mathbf{X}$, c, exponents m and η, weights a and b, scale factor K (or scales γ_i), tolerance ϵ

run FCM on $\mathbf{X}$ to obtain $\mathbf{U}$ and $\mathbf{V}$; scales γ_i from $\mathbf{U}$ and $\mathbf{V}$ as in (6.3) unless given;

repeat

  typicalities t_{ik} by (6.5) from the current distances;

  prototypes $\mathbf{v}_i$ by (6.5) with weights $a\,\mu_{ik}^m + b\,t_{ik}^\eta$;

  distances $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$; memberships μ_{ik} by the FCM formula (3.6);

until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: prototypes $\mathbf{V}$, memberships $\mathbf{U}$ (rows sum to one), typicalities $\mathbf{T}$, scales γ_i, J (6.4)

In a nutshell Possibilistic fuzzy c-means

- Memberships (relative, FCM formula) and typicalities (absolute, PCM formula) side by side; both enter the prototype update, with weights a and b.
- The memberships keep the prototypes apart, the typicalities keep the outliers out; $b/a \approx 2$–5 with scales from FCM is a sound choice, and a large a keeps the prototypes separated over a wide range of scales.
- η sets the decay of the typicality as m does in PCM; b also tightens the effective scale γ_i/b. `predict_proba` returns memberships, `predict_typicality` typicalities.

6.2.3 Noise clustering

Davé [64] keeps the probabilistic constraint and adds a $(c+1)$-th cluster: the *noise prototype* $\mathbf{v}_*$, which has no position and is defined to be at the same distance δ from every sample. The functional is the FCM functional with this extra cluster,

$$J = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^m D_{ik}^2 + \sum_{k=1}^{N} \mu_{*k}^m\,\delta^2, \qquad \mu_{*k} = 1 - \sum_{i=1}^{c} \mu_{ik}, \tag{6.6}$$

and the Lagrange conditions give the FCM membership formula extended by the noise term in the denominator,

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{c} \left(D_{ik}^2/D_{jk}^2\right)^{1/(m-1)} + \left(D_{ik}^2/\delta^2\right)^{1/(m-1)}}, \tag{6.7}$$

while the prototypes of the real clusters are the FCM weighted means (3.7). Only the membership formula changes. A sample much closer than δ to some prototype

Algorithm 6.3 Noise clustering (NC)

Input: data $\mathbf{X}$, c real clusters, exponent m, multiplier λ (or a fixed δ), tolerance ϵ

initial partition as in FCM; provisional δ by (6.8);
repeat
  prototypes $\mathbf{v}_i$ of the real clusters by (3.7);
  if δ *is not fixed* **then** δ by (6.8) from the current distances;
  memberships μ_{ik} by (6.7); noise membership $\mu_{*k} = 1 - \sum_i \mu_{ik}$;
until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;
labels: $\arg\max_i \mu_{ik}$, or "noise" where μ_{*k} exceeds every μ_{ik};

Output: prototypes $\mathbf{V}$, memberships $\mathbf{U}$ of the real clusters (rows sum to $1 - \mu_{*k}$), noise memberships μ_{*k}, δ, J (6.6)

hardly notices the extra term; a sample farther than δ from all of them has $D_{ik}^2/\delta^2 > 1$ for every i, so its real memberships shrink and the remainder μ_{*k} goes to the noise cluster (fig. 6.3). The real clusters still compete for every sample through the constraint, so the coincident clusters of PCM do not occur, and the memberships of the real clusters *close* around the groups instead of tiling the whole plane. The only parameter is δ, which acts as a radius beyond which a sample is more noise than member. Davé tied it to the scale of the data through the mean squared distance of all samples to all current prototypes,

$$\delta^2 = \lambda \frac{1}{Nc} \sum_{i=1}^{c} \sum_{k=1}^{N} D_{ik}^2, \tag{6.8}$$

with a user-chosen multiplier λ. Since that mean is dominated by the distances *between* the groups, λ has to be well below one to catch outliers near a group, while a very small λ starts to declare the tails of the groups noise; $\lambda \approx 0.1$–0.5 is the range worth scanning, and a fixed δ can be given when the noise radius is known. When δ is not given, `fuzzyclust` re-estimates it from the current prototypes at every iteration, so the functional changes slightly between iterations and J need not decrease monotonically. Davé and Krishnapuram [66] later unified the two approaches: PCM is noise clustering with one noise cluster *per* real cluster and $\delta_i^2 = \eta_i$, and both are instances of the robust M-estimators of statistics [67], in which the typicality weight t_{ik}^m and the noise-reduced membership act as redescending weight functions that give far samples a vanishing influence.

6.2.4 Kernel fuzzy c-means

A Mercer kernel $K(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$ is an inner product in a feature space $\mathcal{H}$ reached by a map Φ that is never computed [68]; the Gaussian kernel $K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)$ is the usual choice. Kernel fuzzy c-means runs FCM in $\mathcal{H}$: the

In a nutshell **Noise clustering**

- FCM with one extra, position-less prototype at distance δ from every sample; only the membership formula (6.7) changes.
- Samples farther than δ from all real prototypes migrate to the noise cluster, the real memberships close around the groups, and the prototypes are protected; the clusters keep competing, so they do not coincide.
- $\delta^2 = \lambda\overline{D^2}$ with $\lambda \approx 0.1$–0.5, or a fixed δ; the noise membership μ_{*k} is an outlier score, and noise-dominated samples are labeled -1. PCM is NC with one noise cluster per real cluster.

functional and the membership formula are those of FCM, and the distance is the feature-space distance $\|\Phi(\mathbf{x}_k) - \mathbf{w}_i\|^2$ to a prototype $\mathbf{w}_i \in \mathcal{H}$. Spheres in $\mathcal{H}$ are curved regions in the input space, which is how a ring or a moon can become one cluster. Following the comparative study of Graves and Pedrycz [69], we distinguish two variants by where the prototype lives.

Prototypes in the feature space. The prototype is the weighted mean of the mapped samples, $\mathbf{w}_i = \sum_k \omega_{ik}\Phi(\mathbf{x}_k)$ with $\omega_{ik} = \mu_{ik}^m / \sum_l \mu_{il}^m$—the minimizer of $\sum_k \mu_{ik}^m \|\Phi(\mathbf{x}_k) - \mathbf{w}_i\|^2$ over $\mathcal{H}$. Expanding the squared distance and replacing every inner product by a kernel value gives

$$D_{ik}^2 = K(\mathbf{x}_k, \mathbf{x}_k) - 2\sum_{l=1}^{N} \omega_{il} K(\mathbf{x}_k, \mathbf{x}_l) + \sum_{l,l'=1}^{N} \omega_{il}\omega_{il'} K(\mathbf{x}_l, \mathbf{x}_{l'}), \tag{6.9}$$

the fuzzy counterpart of kernel k-means [70]. The prototype never appears explicitly: it is represented by its weights ω_{il} over the training samples, the $N \times N$ kernel matrix is computed once, and (6.9) with a new sample $\mathbf{x}$ in place of $\mathbf{x}_k$ gives its distances and memberships. Any Mercer (positive semidefinite) kernel can be used; the price is $\mathcal{O}(N^2)$ memory and prototypes that have no exact pre-image in the input space (`fuzzyclust` reports the weighted means $\sum_l \omega_{il}\mathbf{x}_l$ for plotting only). The code also accepts the sigmoid kernel $\tanh(\gamma\, \mathbf{x}^\mathsf{T}\mathbf{y} + c_0)$, which is not positive semidefinite in general: (6.9) can then turn negative, the toolbox clips it at zero, and the result is no longer a distance in any feature space.

Prototypes in the input space. Zhang and Chen [71] keep $\mathbf{v}_i$ in the input space and use the distance between the mapped sample and the *mapped prototype*, which for the Gaussian kernel is

$$D_{ik}^2 = \|\Phi(\mathbf{x}_k) - \Phi(\mathbf{v}_i)\|^2 = 2\big(1 - K(\mathbf{x}_k, \mathbf{v}_i)\big), \qquad \mathbf{v}_i = \frac{\sum_k \mu_{ik}^m K(\mathbf{x}_k, \mathbf{v}_i)\, \mathbf{x}_k}{\sum_k \mu_{ik}^m K(\mathbf{x}_k, \mathbf{v}_i)}, \tag{6.10}$$

where the prototype update, obtained by setting the gradient to zero, is a fixed-point iteration in which every sample is additionally weighted by its kernel similarity

Algorithm 6.4 Kernel fuzzy c-means (feature-space prototypes)

Input: data $\mathbf{X}$, c, exponent m (close to one), kernel and its parameters (γ), tolerance ϵ, restarts

kernel matrix $\mathbf{K} = [K(\mathbf{x}_k, \mathbf{x}_l)]$ once; initial partition as in FCM (Euclidean);

repeat

  weights $\omega_{ik} = \mu_{ik}^m / \sum_l \mu_{il}^m$ (the implicit prototypes $\mathbf{w}_i = \sum_k \omega_{ik} \Phi(\mathbf{x}_k)$);

  distances D_{ik}^2 by (6.9) from $\mathbf{K}$ and the weights;

  memberships μ_{ik} by the FCM formula (3.6);

until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

keep the restart with the smallest $J = \sum_i \sum_k \mu_{ik}^m D_{ik}^2$;

Output: weights ω_{ik} (prototypes), memberships $\mathbf{U}$, approximate pre-images $\sum_k \omega_{ik} \mathbf{x}_k$; new samples $\mathbf{x}$ by (6.9) with $\mathbf{x}$ in place of $\mathbf{x}_k$. Input-space variant: replace the distance and the update by (6.10)

to the current prototype. The distance saturates at 2 for far samples, so outliers lose their influence: this variant is a *robust* FCM. The distance is, however, a monotone function of $\|\mathbf{x}_k - \mathbf{v}_i\|$, so its hard partition boundaries are the same bisecting hyperplanes as those of FCM, and it cannot find rings.

For both variants the decisive parameter is γ, which sets the width of the kernel (a large γ means a narrow kernel). As $\gamma \to 0$ the Gaussian kernel becomes linear in the squared distance and kernel FCM collapses to FCM; as $\gamma \to \infty$ all mapped samples become mutually orthogonal, every sample is equally far from every prototype, and the partition is arbitrary. The usable window, where $1/\sqrt{\gamma}$ is about the gap between the structures to be separated, spans less than a decade, and inside it the exponent m must be close to one, because in the nearly orthogonal feature space a fuzzy exponent spreads the memberships out and makes every prototype the mean of all samples. Graves and Pedrycz found over many data sets that kernel FCM is not uniformly better than FCM and that the kernel parameter is hard to set, and the examples of section 6.4.4 agree. Kernel FCM separates rings that no point prototype can, but for data on clear manifolds the geodesic methods of chapter 7 are the more forgiving tool.

In a nutshell Kernel fuzzy c-means

- FCM in the feature space of a kernel: same functional and membership formula, the distance is evaluated through kernel values only.
- Feature-space prototypes (weights over the samples) find rings and moons, at $\mathcal{O}(N^2)$ memory; input-space prototypes with the Gaussian kernel give a robust FCM whose cells are still convex.
- $1/\sqrt{\gamma}$ must match the gap between the structures, and the usable window is narrow; use a nearly crisp m, several restarts, and a scan over γ.

6.2.5 Fuzzy c-varieties and c-elliptotypes

Bezdek, Coray, Gunderson, and Watson [72, 73] replace the point prototype by an r-dimensional *linear variety* (affine subspace)

$$V_i = \left\{ \mathbf{v}_i + \sum_{j=1}^{r} \tau_j \mathbf{s}_{ij} \;:\; \tau \in \mathbb{R}^r \right\}, \qquad \mathbf{s}_{ij}^{\mathsf{T}} \mathbf{s}_{il} = \delta_{jl}, \tag{6.11}$$

which is a point for $r = 0$ (FCM), a line for $r = 1$ (*fuzzy c-lines*, FCL), and a hyperplane for $r = n - 1$. The squared distance of a sample to the cluster is its squared orthogonal distance to the variety, that is, the squared distance to the center minus the squared projections onto the directions,

$$D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2 - \sum_{j=1}^{r} \left((\mathbf{x}_k - \mathbf{v}_i)^{\mathsf{T}} \mathbf{s}_{ij} \right)^2, \tag{6.12}$$

and the functional is the c-means functional with this distance. Fixing the prototypes gives the FCM membership formula; fixing the memberships, the center is again the fuzzy mean (3.7), and the directions that minimize J are the r unit eigenvectors belonging to the largest eigenvalues of the fuzzy scatter matrix of the cluster,

$$\mathbf{S}_i = \sum_{k=1}^{N} \mu_{ik}^m (\mathbf{x}_k - \mathbf{v}_i)(\mathbf{x}_k - \mathbf{v}_i)^{\mathsf{T}}, \tag{6.13}$$

because $\sum_k \mu_{ik}^m D_{ik}^2 = \operatorname{tr} \mathbf{S}_i - \sum_j \mathbf{s}_{ij}^{\mathsf{T}} \mathbf{S}_i \mathbf{s}_{ij}$ is minimized by directing the $\mathbf{s}_{ij}$ along the largest spread. The prototype step is a membership-weighted principal component analysis per cluster; the retained eigenvalues measure the spread *along* the variety, the discarded ones the scatter *around* it, and their ratio tells how line-like the cluster is. The ellipsoids of Gustafson–Kessel (section 3.2.4) stretch along a line but still measure the position along it; a line prototype measures only the distance from the line, so it separates lines that cross at a point, which GK and FCM have to cut into pieces (fig. 6.19).

The price of a prototype without extent is that collinear groups merge into one cluster: two segments on the same line, however far apart, lie at zero distance from

Algorithm 6.5 Fuzzy c-varieties (FCV) and c-elliptotypes

Input: data $\mathbf{X}$, c, exponent m, variety dimension $r < n$, elliptotype weight α (0 for pure FCV), tolerance ϵ, restarts

initial partition as in FCM;
repeat
  centers $\mathbf{v}_i$ by the fuzzy mean (3.7);
  scatter matrices $\mathbf{S}_i$ (6.13); directions $\mathbf{s}_{i1}, \dots, \mathbf{s}_{ir}$ = eigenvectors of the r largest eigenvalues;
  distances by (6.12), mixed with the point distance by (6.14) if $\alpha > 0$;
  memberships μ_{ik} by the FCM formula (3.6);
until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: centers $\mathbf{V}$, directions $\mathbf{s}_{ij}$, eigenvalues (spread along / around), memberships $\mathbf{U}$, J

the same line prototype. Bezdek et al. [73] therefore consider convex combinations $\sum_r \alpha_r J_r$ of the FCV functionals of different dimension, whose prototypes are "elliptotypes" between a point and a line. The two-term combination of $r = 0$ and $r = 1$ is the *fuzzy c-elliptotypes* algorithm (FCE): for the same center and direction its distance is the convex combination of the point and the line distance,

$$D^2_{ik,\mathrm{FCE}} = \alpha \, \|\mathbf{x}_k - \mathbf{v}_i\|^2 + (1 - \alpha) \, D^2_{ik,\mathrm{FCL}}, \qquad \alpha \in [0, 1], \tag{6.14}$$

so that the position along the line counts with weight α and collinear groups separate; $\alpha = 1$ is FCM. The weight should be as small as the data allow. In the example of section 6.4.5, $\alpha = 0.05$ already separates two collinear segments, whereas three crossing lines lose accuracy at $\alpha = 0.1$ and are no longer found from $\alpha = 0.2$ on, because near a crossing the position along a line must not count. Lines through a common point are hard for a random start, since a single run often puts two prototypes on one line, so restarts are advisable.

6.2.6 Fuzzy c-shells and c-spherical shells

A ring has no samples in its interior, so the point prototype at its center is far from every member, and two concentric rings have the *same* center: no point prototype can tell them apart. Shell clustering describes cluster i by a center $\mathbf{v}_i$ *and* a radius r_i and measures the distance of a sample as its deviation from the sphere. The two published algorithms implemented here differ in this distance. Davé's fuzzy c-shells algorithm (FCS) [74] uses the squared *radial* (geometric) deviation, and the fuzzy c-spherical shells algorithm (FCSS) of Krishnapuram, Nasraoui, and Frigui [75] the squared *algebraic* distance,

$$D^2_{ik,\mathrm{FCS}} = \left(\|\mathbf{x}_k - \mathbf{v}_i\| - r_i \right)^2, \qquad D^2_{ik,\mathrm{FCSS}} = \left(\|\mathbf{x}_k - \mathbf{v}_i\|^2 - r_i^2 \right)^2, \tag{6.15}$$

In a nutshell Fuzzy c-varieties

- Prototype = r-dimensional linear variety (line, plane) through the fuzzy mean, spanned by the leading eigenvectors of the fuzzy scatter matrix; distance = squared orthogonal distance to it; membership formula unchanged.
- Groups samples by the subspace they lie on, not by position: separates crossing lines that FCM and GK cut into pieces; the discarded eigenvalues measure the scatter around the variety.
- Merges collinear groups—a small elliptotype weight α of the point distance (0.05 in the example) is the remedy, but $\alpha \geq 0.1$ already spoils crossing lines; use restarts for lines through a common point.

both used in the c-means functional with the FCM membership formula. The two algorithms differ in the prototype update. For FCS the necessary conditions,

$$r_i = \frac{\sum_k \mu_{ik}^m \|\mathbf{x}_k - \mathbf{v}_i\|}{\sum_k \mu_{ik}^m}, \qquad \sum_k \mu_{ik}^m \Big(1 - \frac{r_i}{\|\mathbf{x}_k - \mathbf{v}_i\|}\Big)(\mathbf{x}_k - \mathbf{v}_i) = \mathbf{0}, \tag{6.16}$$

are coupled and nonlinear in $\mathbf{v}_i$, and Davé solved them by Newton's method in every iteration. FCSS avoids this inner iteration: the algebraic distance is $\|\mathbf{x}_k\|^2 + \mathbf{p}_i^\mathsf{T}\mathbf{y}_k$ with the reparametrized prototype $\mathbf{p}_i = [-2\mathbf{v}_i^\mathsf{T},\ \mathbf{v}_i^\mathsf{T}\mathbf{v}_i - r_i^2]^\mathsf{T}$ and the regressors $\mathbf{y}_k = [\mathbf{x}_k^\mathsf{T},\ 1]^\mathsf{T}$, i.e. *linear* in $\mathbf{p}_i$, so the weighted least-squares minimizer is in closed form,

$$\begin{aligned} \mathbf{p}_i &= -\Big(\sum_k \mu_{ik}^m \mathbf{y}_k \mathbf{y}_k^\mathsf{T}\Big)^{-1} \sum_k \mu_{ik}^m \|\mathbf{x}_k\|^2 \mathbf{y}_k, \\ \mathbf{v}_i &= -\tfrac{1}{2}\mathbf{p}_{i,1:n}, \qquad r_i^2 = \mathbf{v}_i^\mathsf{T}\mathbf{v}_i - p_{i,n+1}, \end{aligned} \tag{6.17}$$

one $(n+1) \times (n+1)$ solve per cluster and iteration. Near the shell the two distances are related by $\|\mathbf{x}\|^2 - r^2 = (\|\mathbf{x}\| - r)(\|\mathbf{x}\| + r) \approx 2r(\|\mathbf{x}\| - r)$: the algebraic distance is about $2r_i$ times the radial deviation, so the same radial deviation counts more on a large shell than on a small one, and FCSS memberships favor small shells. The bias shows when nested shells of similar radius have to be separated or when only arcs are observed; FCS does not have it. `fuzzyclust` implements both: `algorithm="fcss"` is the default, and `algorithm="fcs"` refines the closed-form fit in every iteration by `newton_steps` Gauss–Newton steps on the radial residuals, a numerically convenient form of Davé's Newton solution. The initialization deserves attention. Shell clustering has a notorious local minimum—one big circle through everything, with the other shells fitting noise—and a random start rarely escapes it when the shells intersect; the hardened partition of an FCM run is then a good start, because the FCM groups lie on different shells. For *concentric* shells FCM halves every ring and both shells converge to the mean radius; here the further restarts of `n_init`, which start from random point prototypes and are selected by J, find the rings.

Algorithm 6.6 Fuzzy c-(spherical) shells (FCSS / FCS)

Input: data $\mathbf{X}$, c shells, exponent m, algorithm (FCSS or FCS with Newton steps), tolerance ϵ, restarts

initial partition: hardened FCM partition (first start) or random point prototypes;

repeat

  for *every shell* i **do**

    closed-form fit $(\mathbf{v}_i, r_i)$ by (6.17) with weights μ_{ik}^m;

    if *FCS* **then** refine $(\mathbf{v}_i, r_i)$ by Gauss–Newton steps on the residuals $\|\mathbf{x}_k - \mathbf{v}_i\| - r_i$;

  distances by (6.15) (algebraic for FCSS, radial for FCS); memberships by the FCM formula;

until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

keep the restart with the smallest J;

Output: centers $\mathbf{V}$, radii r_i, memberships $\mathbf{U}$, J; signed radial deviations $\|\mathbf{x}_k - \mathbf{v}_i\| - r_i$

In a nutshell Fuzzy c-shells

- Prototype = center and radius; distance = deviation from the sphere (radial in FCS, algebraic in FCSS); the membership formula is that of FCM.
- FCSS has a closed-form least-squares update and is the default; FCS needs Newton steps but is unbiased with respect to the shell size (FCSS favors small shells).
- Start from the hardened FCM partition for separate or intersecting shells and use restarts for concentric shells; the results are the radii and the signed radial residuals.

6.2.7 Volume prototypes and adaptive cluster merging

A point prototype measures how far a sample is from the *center* of a cluster. A sample in the outer part of a large cluster can therefore be closer to the prototype of a small neighboring cluster than to its own, and the membership formula hands it to the neighbor: FCM and GK equalize the clusters, splitting a large diffuse group and letting small groups absorb its pieces (section 3.4.6). Gath–Geva solves the problem with densities and priors at the price of a sensitive iteration. Kaymak and Setnes [43] keep the stable FCM iteration and change the prototype: a *volume prototype* is a hypersphere (or, in the adaptive norm of GK, a hyperellipsoid) of radius r_i around $\mathbf{v}_i$, and only the distance to its surface counts,

$$\tilde{D}_{ik} = \max\bigl(0,\ D_{ik\mathbf{A}_i} - r_i\bigr), \qquad D^2_{ik\mathbf{A}_i} = (\mathbf{x}_k - \mathbf{v}_i)^{\mathsf{T}} \mathbf{A}_i (\mathbf{x}_k - \mathbf{v}_i), \tag{6.18}$$

with $\mathbf{A}_i = \mathbf{I}$ or the GK norm (3.10). The functional is the c-means functional with $\tilde{D}_{ik}^2$; the memberships follow the FCM formula (3.6) with $\tilde{D}_{ik}$ in place of D_{ik}, so a sample inside exactly one core belongs fully to that cluster; and the centers are the μ_{ik}^m-weighted means. The radius adapts to the cluster. Kaymak and Setnes derive it from the fuzzy covariance matrix as $(\det \mathbf{F}_i)^{1/(2n)}$, the geometric mean of the standard deviations along the principal axes, so that a sphere of that radius has the volume of the cluster, and they multiply it by a factor that grows as the clusters merge. The toolbox uses a simpler estimate, recomputed at every iteration: the membership-weighted mean distance of the data from the center,

$$r_i = \kappa \, \frac{\sum_k \mu_{ik}^m D_{ik\mathbf{A}_i}}{\sum_k \mu_{ik}^m}, \tag{6.19}$$

with a fixed scale factor κ (default one); $\kappa = 0$ recovers FCM or GK. A large cluster thus has a large core, its outer members are near the surface of that core, and they stay with it. Cores that grow too large are the failure mode of the method: a core that reaches across a neighboring group swallows it, so κ should not be raised above one (section 6.4.7).

Adaptive cluster merging. Kaymak and Setnes combine the volume prototypes with merging during the iteration. The algorithm starts with a deliberately large number of clusters. They measure the similarity of two clusters by the fuzzy inclusion measure $I_{ij} = \sum_k \min(\mu_{ik}, \mu_{jk}) / \sum_k \mu_{ik}$, symmetrized as $S_{ij} = \max(I_{ij}, I_{ji})$, and in every iteration in which the largest similarity has changed by less than a tolerance they merge the most similar pair, provided that its similarity exceeds the threshold $1/(c-1)$, which rises as the number c of clusters falls; the iteration continues with one cluster fewer. The toolbox uses a variant that is the similarity criterion of compatible cluster merging (section 4.2.7) built into the iteration: it merges only when the partition has converged for the current number of clusters, it measures the similarity by the ratio of fuzzy intersection and union, $S_{ij} = \sum_k \min(\mu_{ik}, \mu_{jk}) / \sum_k \max(\mu_{ik}, \mu_{jk})$ (4.17), and its threshold is fixed. Inside the cores the memberships are nearly crisp, which would hide the overlap of two clusters that share a group, so the toolbox evaluates the similarity on the memberships that the center distances alone would give, rescaled to $m = 2$; two clusters on one group then have $S_{ij} \approx 0.25$–0.35, and the default threshold is 0.2.

6.2.8 Learning which features matter

The Euclidean distance of FCM gives every feature the same weight. After standardization, a feature of pure noise contributes as much to D_{ik}^2 as a feature that separates the clusters, and enough such features drown the structure. Clusters that are defined by different features, one tight along x_1 and another along x_2, cannot be described by one shared norm at all. The two methods of this section learn feature weights together with the partition. Both use the per-feature squared

Algorithm 6.7 Fuzzy clustering with volume prototypes and adaptive merging (toolbox form)

Input: data $\mathbf{X}$, initial (maximal) number of clusters c, exponent $m > 1$, norm (Euclidean or adaptive), radius scale κ, merging on/off with threshold γ_S, tolerance ϵ

initialize $\mathbf{U}$;
repeat
  repeat
    centers $\mathbf{v}_i = \sum_k \mu_{ik}^m \mathbf{x}_k / \sum_k \mu_{ik}^m$; covariances $\mathbf{F}_i$ and norms $\mathbf{A}_i$ (identity or GK);
    radii r_i by (6.19); core distances $\tilde{D}_{ik}$ by (6.18);
    memberships by the FCM formula with $\tilde{D}_{ik}$;
  until $\max_{ik} |\Delta \mu_{ik}| < \epsilon$ *(or the iteration budget of this c is spent)*;
  if *merging* **then**
    S_{ij} of (4.17) from the center distances, rescaled to $m = 2$;
    if $\max_{i \neq j} S_{ij} \geq \gamma_S$ **then** add the two membership columns, $c \leftarrow c - 1$;
until *converged and (merging off or no pair with $S_{ij} \geq \gamma_S$)*;

Output: $\mathbf{V}$, radii r_i, $\mathbf{U}$, final number of clusters, merge history

In a nutshell Volume prototypes and adaptive merging

- A prototype is a core of radius r_i with zero distance inside; only the distance to the core surface counts, which removes the size-equalizing bias of point prototypes.
- Spherical (Euclidean) or ellipsoidal (GK norm) cores; the radius is re-estimated from the cluster, $\kappa = 0$ is FCM/GK; keep $\kappa \leq 1$.
- Adaptive merging starts with too many clusters and merges similar pairs during the iteration; the final number is insensitive to the starting number.
- Stays within the stable FCM iteration; a lighter alternative to Gath–Geva for clusters of different size.
- Parameters: `n_clusters`, `m`, `norm`, `radius`, `radius_scale`, `merge`, `similarity_threshold`, `gamma`, `beta`; attributes `radii_`, `n_clusters_`, `merge_history_`.

distances $d_{ikj}^2 = (x_{kj} - v_{ij})^2$ and the membership-weighted scatter of a cluster along a feature,

$$s_{ij} = \sum_{k=1}^{N} \mu_{ik}^m d_{ikj}^2, \tag{6.20}$$

and both reward a feature along which the members of the clusters scatter little.

Weighted fuzzy c-means. The simpler of the two gives all clusters a common weight vector $\boldsymbol{\alpha}$, $\sum_j \alpha_j = 1$, raised to an exponent $t > 1$,

$$J = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \sum_{j=1}^{n} \alpha_j^t \, d_{ikj}^2, \qquad \alpha_j = \frac{1}{\sum_{l=1}^{n} \left(s_j / s_l \right)^{1/(t-1)}}, \quad s_j = \sum_{i=1}^{c} s_{ij}. \tag{6.21}$$

This is the fuzzy form of the automated variable weighting that Huang, Ng, Rong, and Li [76] proposed for k-means; Keller and Klawonn [77] had introduced the same kind of functional with one weight for each variable *and each cluster*, which is the form of SCAD-2 below. The weight update has the form of the FCM membership formula with the total scatter of a feature in place of the distance to a prototype. The weights define the diagonal of a learned norm-inducing matrix $\mathbf{A} = \operatorname{diag}(\alpha_j^t)$, which places the method between FCM with the fixed diagonal norm and GK with its full, cluster-specific norm. As m does for the memberships, the exponent t controls how sharply the weight concentrates: $t \to 1$ puts all weight on the feature of smallest scatter, and $t \to \infty$ gives uniform weights.

SCAD. Frigui and Nasraoui's *simultaneous clustering and attribute discrimination* [78] gives every cluster its own weights w_{ij}, $\sum_j w_{ij} = 1$, as Keller and Klawonn did, so that different clusters can live in different subspaces. *SCAD-1* uses the weights linearly and prevents the trivial solution—all weight on the single feature of smallest scatter—by a quadratic term with a cluster-specific coefficient δ_i,

$$\begin{aligned} J_1 &= \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \sum_{j=1}^{n} w_{ij} d_{ikj}^2 + \sum_{i=1}^{c} \delta_i \sum_{j=1}^{n} w_{ij}^2, \\ w_{ij} &= \frac{1}{n} + \frac{1}{2\delta_i} \left(\frac{1}{n} \sum_{l=1}^{n} s_{il} - s_{ij} \right), \qquad \delta_i = K \, \frac{\sum_k \mu_{ik}^m \sum_j w_{ij} d_{ikj}^2}{\sum_j w_{ij}^2}, \end{aligned} \tag{6.22}$$

where the weight of a feature rises above $1/n$ when the cluster scatters less along it than on average. Negative weights are set to zero and the row is renormalized, and δ_i is re-estimated from the current partition so that the two terms of the functional stay balanced (K is a scale factor, default one). *SCAD-2* raises the weights to an exponent $q > 1$ instead and needs no δ_i,

$$J_2 = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \sum_{j=1}^{n} w_{ij}^q d_{ikj}^2, \qquad w_{ij} = \frac{1}{\sum_{l=1}^{n} \left(s_{ij} / s_{il} \right)^{1/(q-1)}}; \tag{6.23}$$

Algorithm 6.8 Weighted fuzzy c-means (global feature weights)

Input: standardized data $\mathbf{X}$, number of clusters c, exponents $m > 1$ and $t > 1$, tolerance ϵ

initialize $\mathbf{U}$; weights $\alpha_j \leftarrow 1/n$;
repeat
  centers $\mathbf{v}_i = \sum_k \mu_{ik}^m \mathbf{x}_k / \sum_k \mu_{ik}^m$;
  scatters $s_j = \sum_{i,k} \mu_{ik}^m (x_{kj} - v_{ij})^2$; weights α_j by (6.21);
  distances $D_{ik}^2 = \sum_j \alpha_j^t (x_{kj} - v_{ij})^2$; memberships by the FCM formula;
until $\max_{ik} |\Delta \mu_{ik}| < \epsilon$;

Output: $\mathbf{V}$, $\mathbf{U}$, feature weights $\boldsymbol{\alpha}$

In a nutshell Weighted fuzzy c-means

- One weight vector for all clusters, learned with the partition: a learned diagonal norm diag(α_j^t).
- Down-weights features along which the clusters scatter widely (noise features); reports a global feature relevance.
- Standardize first; keep t well above one ($t \approx 3$), otherwise the weight concentrates on one feature.
- Parameters: `n_clusters`, `m`, `t`; attribute `feature_weights_`.

its weights are never exactly zero. In both versions the memberships follow the FCM formula with the weighted distance $D_{ik}^2 = \sum_j w_{ij} d_{ikj}^2$ (or w_{ij}^q), and the centers are the μ_{ik}^m-weighted means, since each coordinate of a center minimizes its own weighted sum of squares. The level sets of the weighted distance are axis-parallel ellipsoids, stretched along the features a cluster ignores.

Two cautions apply to both methods. First, the weights reward a small within-cluster scatter, so the features must be on comparable scales: a feature of small variance attracts the weight regardless of its relevance, and even among standardized features one along which the data can be cut into c thin slices may attract all of it when the exponent is close to one. Second, the weights are learned from the partition: if the noise features already dominate the first iterations, the partition carries no information and the weights stay uniform. Feature weighting raises the number of noise features that FCM tolerates; it does not replace feature selection when most features are irrelevant (section 6.4.8).

6.3 Using it in fuzzyclust

The nine estimators of this chapter live in `fuzzyclust.extensions`. They derive from `BaseFuzzyClusterer` and share the common interface: `fit`, `predict`,

Algorithm 6.9 SCAD: simultaneous clustering and attribute discrimination

Input: standardized data $\mathbf{X}$, number of clusters c, exponent $m > 1$; SCAD-1 with scale K (or fixed δ_i), or SCAD-2 with exponent $q > 1$; tolerance ϵ

initialize $\mathbf{U}$; weights $w_{ij} \leftarrow 1/n$;
repeat
  centers $\mathbf{v}_i = \sum_k \mu_{ik}^m \mathbf{x}_k / \sum_k \mu_{ik}^m$; scatters s_{ij} by (6.20);
  if *SCAD-1* **then**
    δ_i from the current partition; w_{ij} by (6.22); clip at zero and renormalize every row;
  else
    w_{ij} by (6.23);
  distances $D_{ik}^2 = \sum_j w_{ij} d_{ikj}^2$ (SCAD-2: w_{ij}^q); memberships by the FCM formula;
until $\max_{ik} |\Delta \mu_{ik}| < \epsilon$;

Output: $\mathbf{V}$, $\mathbf{U}$, weights w_{ij} of every cluster

In a nutshell SCAD

- A weight vector per cluster: clusters may be defined by different features (subspace clusters), and an irrelevant feature is down-weighted in every cluster.
- SCAD-1: linear weights with a balancing quadratic term, may give exact zeros; SCAD-2: weights with an exponent q, never zero, sharper for q near one.
- Same cost as FCM plus $\mathcal{O}(Ncn)$ for the scatters; standardize first.
- Parameters: `n_clusters`, `m`, `variant`, `q`, `delta`, `delta_scale`; attributes `feature_weights_` (c, n), `delta_`.

`predict_proba`, `transform`, `score`, the `n_init` restarts, and pandas input. What `predict_proba` returns depends on the method and is stated in every docstring: memberships that sum to one (kernel FCM, FCV, shells, volume prototypes, the feature-weighted methods, and PFCM), typicalities that do not (PCM), or real memberships whose rows sum to $1 - \mu_{*k}$ (noise clustering).

`fuzzyclust.extensions.PCM`

`n_clusters=3, m=2.0, eta=None, eta_scale=1.0, tol, max_iter, init="fcm"|"random"|"kmeans++"|array, n_init=1, random_state, verbose.`

Attributes `cluster_centers_`, `typicality_` (= `membership_`, rows not normalized), `labels_`, `eta_`. `predict_proba` and `predict_typicality` return typicalities. Use $m \approx 1.5$; `eta_scale` is the factor K of (6.3). An `init` array holds either initial FCM prototypes (c, n) or an initial partition (N, c), such as the `membership_` of a fitted FCM or GK model.

`fuzzyclust.extensions.PFCM`

`n_clusters=3, m=2.0, eta=2.0, a=1.0, b=1.0, gamma=None, gamma_scale=1.0, tol, max_iter, init="fcm", n_init=1, random_state, verbose.`
Attributes `membership_` (rows sum to one), `typicality_`, `gamma_`, `labels_`. `predict_proba` gives memberships, `predict_typicality` typicalities. $b = 0$ is FCM, $a = 0$ is PCM.

`fuzzyclust.extensions.NoiseClustering`

`n_clusters=3, m=2.0, delta=None, delta_scale=0.5, tol, max_iter, init, n_init=1, random_state, verbose.`
Attributes `membership_` (real clusters, rows sum to $1 - \mu_{*k}$), `noise_membership_`, `delta_`, `labels_` (-1 for noise-dominated samples). `delta_scale` is λ of (6.8); `predict_noise_membership(X)` for new data.

`fuzzyclust.extensions.KernelFCM`

`n_clusters=3, m=2.0, kernel="rbf"|"poly"|"linear"|"sigmoid"|"cosine"|"laplacian", gamma=None, degree=3, coef0=1.0, prototypes="feature"|"input", tol, max_iter, init, n_init=1, random_state, verbose.`
Kernels through `sklearn.metrics.pairwise`; `gamma=None` uses $1/(n\,\mathrm{Var}(\mathbf{X}))$. `prototypes="input"` is the Zhang–Chen variant (Gaussian kernel only). Attributes `membership_`, `cluster_centers_` (approximate pre-images for `"feature"`), `gamma_`; `predict_proba` uses the kernelized distance (6.9).

`fuzzyclust.extensions.FuzzyCVarieties`

`n_clusters=3, m=2.0, n_dims=1, alpha=0.0, tol, max_iter, init, n_init=1, random_state, verbose.`
Attributes `cluster_centers_`, `directions_` ($c \times r \times n$), `scatter_matrices_`, `covariances_`, `eigenvalues_` (descending), `membership_`. `n_dims=0` is FCM, `alpha` the elliptotype weight of (6.14).

fuzzyclust.extensions.FuzzyCShells

`n_clusters=3`, `m=2.0`, `algorithm="fcss"|"fcs"`, `newton_steps=2`, `tol`, `max_iter`, `init="fcm"|"random"|"kmeans++"|array`, `n_init=1`, `random_state`, `verbose`.
Attributes `cluster_centers_`, `radii_`, `membership_`; `distances_` and `transform` are $|\,\|\mathbf{x}_k-\mathbf{v}_i\|^2-r_i^2\,|$ for FCSS and $|\,\|\mathbf{x}_k-\mathbf{v}_i\|-r_i\,|$ for FCS; `radial_deviation(X)` gives the signed radial deviation for both.

fuzzyclust.extensions.VolumePrototypeFCM

`n_clusters=3`, `m=2.0`, `norm="euclidean"|"adaptive"`, `radius="fuzzy_mean"`, `radius_scale=1.0`, `merge=False`, `similarity_threshold=0.2`, `gamma=0.0`, `beta=1e15`, `tol`, `max_iter`, `init`, `n_init=1`, `random_state`, `verbose`.
`radius` is `"fuzzy_mean"` (6.19), a number or an array of c fixed radii; `gamma`, `beta` regularize the adaptive norm as in GK. With `merge=True`, `n_clusters` is the starting number and `max_iter` applies per number of clusters. Attributes `radii_`, `n_clusters_`, `merge_history_` (iteration, number of clusters, pair, similarity), `covariances_`, `norm_matrices_`; `distances_` and `transform` return $\tilde{D}_{ik}$, zero inside the cores.

fuzzyclust.extensions.FeatureWeightedFCM

`n_clusters=3`, `m=2.0`, `t=2.0`, `tol`, `max_iter`, `init`, `n_init=1`, `random_state`, `verbose`.
`t`>1 is the weight exponent of (6.21). Attributes `feature_weights_` $(n,)$ summing to one, `covariances_` (unweighted); `distances_` and `transform` return the weighted distances.

fuzzyclust.extensions.SCAD

`n_clusters=3`, `m=2.0`, `variant="scad1"|"scad2"`, `q=2.0`, `delta=None`, `delta_scale=1.0`, `tol`, `max_iter`, `init`, `n_init=1`, `random_state`, `verbose`.
`delta=None` re-estimates δ_i by (6.22) with the factor `delta_scale`$=K$; a number or an array fixes it (SCAD-1). `q` is the exponent of SCAD-2. Attributes `feature_weights_` (c,n) with rows summing to one, `delta_`, `covariances_`.

data generators and plot helpers

In `fuzzyclust.extensions`: `make_circles` (intersecting circles, arcs, outliers), `make_close_blobs` (coincident-cluster test case), `make_crossing_lines`, and `make_collinear_segments`; the examples also use `make_noisy_blobs`, `make_rings`, `make_two_moons`, and `make_different_densities` from `fuzzyclust.datasets`. Plots: `plot_typicality(X, T)` (scatter colored by $\max_i t_{ik}$, also for $1 - \mu_{*k}$), `plot_varieties(X, model)`, and `plot_shells(X, model)`; the generic `fuzzyclust.plotting.plot_membership_contours` works for all estimators of the chapter through `predict_proba`.

The robust variants are drop-in replacements for FCM; the possibilistic ones are seeded by FCM internally:

```
fcm = FCM(n_clusters=3, random_state=0).fit(X)
pcm = PCM(n_clusters=3, m=1.5, random_state=0).fit(X)
print(f"scales eta_i = {pcm.eta_.round(2)}")
print(f"center error: FCM {center_error(fcm.cluster_centers_):.3f}, PCM
    {center_error(pcm.cluster_centers_):.3f}")
```

```
pfcm = PFCM(n_clusters=3, a=1.0, b=4.0, m=2.0, eta=2.0,
    random_state=0).fit(X)
U, T = pfcm.membership_, pfcm.typicality_
print(f"rows of U sum to one: {np.allclose(U.sum(axis=1), 1.0)};
    typicality mass per sample: "
      f"inliers {T[inliers].sum(axis=1).mean():.2f}, outliers
    {T[~inliers].sum(axis=1).mean():.2f}")
```

```
fcm = FCM(n_clusters=3, random_state=0).fit(X)
nc = NoiseClustering(n_clusters=3, delta_scale=0.3, random_state=0).fit(X)
n_noise = (nc.labels_ == -1).sum()
print(f"noise distance delta = {nc.delta_:.2f}; {n_noise} samples labeled
    noise, "
      f"{((nc.labels_ == -1) & ~inliers).sum()} of them true outliers")
print(f"center error: FCM {center_error(fcm.cluster_centers_):.3f}, noise
    clustering {center_error(nc.cluster_centers_):.3f}")
```

The kernel and shape variants change the distance or the prototype and expose what they find (the kernel parameter, the line orientations, the radii):

```
fcm_r = FCM(n_clusters=2, random_state=0).fit(rings[0])
kfcm_r = KernelFCM(n_clusters=2, kernel="rbf", gamma=0.3, m=1.2, n_init=5,
    random_state=0).fit(rings[0])
fcm_m = FCM(n_clusters=2, random_state=0).fit(moons[0])
kfcm_m = KernelFCM(n_clusters=2, kernel="rbf", gamma=15.0, m=1.1,
    n_init=5, random_state=0).fit(moons[0])
for name, (_, y), f, k in (("rings", rings, fcm_r, kfcm_r), ("moons",
    moons, fcm_m, kfcm_m)):
    print(f"{name}: FCM ARI = {adjusted_rand_score(y, f.labels_):.2f}, "
```

```
          f"kernel FCM (gamma = {k.gamma_:g}) ARI =
    {adjusted_rand_score(y, k.labels_):.2f}")
```

```
fcm = FCM(n_clusters=3, n_init=5, random_state=0).fit(X)
gk = GK(n_clusters=3, n_init=5, random_state=0).fit(X)
fcl = FuzzyCVarieties(n_clusters=3, n_dims=1, n_init=5,
    random_state=0).fit(X)
angles = np.degrees(np.arctan2(fcl.directions_[:, 0, 1],
    fcl.directions_[:, 0, 0])) % 180
print(f"FCM ARI = {adjusted_rand_score(y, fcm.labels_):.2f}, GK ARI =
    {adjusted_rand_score(y, gk.labels_):.2f}, "
      f"fuzzy c-lines ARI = {adjusted_rand_score(y, fcl.labels_):.2f}")
print(f"line orientations found: {np.sort(angles).round(1)} degrees (true:
    0, 60, 120)")
```

```
fcm_r = FCM(n_clusters=2, random_state=0).fit(Xr)
fcss_r = FuzzyCShells(n_clusters=2, algorithm="fcss", n_init=5,
    random_state=0).fit(Xr)
fcss_c = FuzzyCShells(n_clusters=3, algorithm="fcss",
    random_state=0).fit(Xc)
print(f"rings: FCM ARI = {adjusted_rand_score(yr, fcm_r.labels_):.2f},
    FCSS ARI = "
      f"{adjusted_rand_score(yr, fcss_r.labels_):.2f}, radii found
    {np.sort(fcss_r.radii_).round(2)} (true 1, 3)")
ce, re = match(fcss_c, true_c, true_r)
print(f"circles: FCSS ARI = {adjusted_rand_score(yc, fcss_c.labels_):.2f},
    "
      f"largest center error {ce:.3f}, largest radius error {re:.3f}")
```

Volume prototypes are a drop-in replacement for FCM, and with `merge=True` the number of clusters becomes a result:

```
fcm = FCM(n_clusters=3, n_init=5, random_state=0).fit(X)
vp = VolumePrototypeFCM(n_clusters=3, n_init=5, random_state=0).fit(X)
print(f"ARI: FCM {adjusted_rand_score(y, fcm.labels_):.2f}, volume
    prototypes {adjusted_rand_score(y, vp.labels_):.2f}")
print("core radii r_i:", np.round(vp.radii_, 2))
```

```
vpm = VolumePrototypeFCM(n_clusters=8, merge=True,
    similarity_threshold=0.2, random_state=0).fit(X)
for step in vpm.merge_history_:
    print(f"iteration {step['iteration']:3d}: c = {step['n_clusters']} ->
    merge {step['pair']} "
          f"(S = {step['similarity']:.2f})")
print(f"final number of clusters: {vpm.n_clusters_}, ARI =
    {adjusted_rand_score(y, vpm.labels_):.2f}")
```

The feature-weighted methods return the learned weights next to the partition:

```
common = dict(n_clusters=3, m=1.3, init="kmeans++", n_init=5,
    random_state=0)
fcm = FCM(**common).fit(X)
wfcm = FeatureWeightedFCM(t=3.0, **common).fit(X)  # one weight vector for
    all clusters
scad = SCAD(variant="scad1", **common).fit(X)  # one weight vector per
    cluster
for name, model in (("FCM", fcm), ("weighted FCM", wfcm), ("SCAD-1",
    scad)):
    print(f"{name:13s} ARI = {adjusted_rand_score(y, model.labels_):.2f}")
print("global weights:", np.round(wfcm.feature_weights_, 2))
```

```
scad2 = SCAD(n_clusters=3, variant="scad2", q=2.0, n_init=5,
    random_state=0).fit(X_bands)
print("SCAD-2 weights (rows: clusters, columns: x1, x2):")
print(np.round(scad2.feature_weights_, 2))
```

6.4 Examples

The first three examples share one data set, generated by `make_noisy_blobs`: three Gaussian groups of 100 points ($\sigma = 0.6$, centers 4.2 to 7.5 apart) with 60 uniformly scattered outliers. The "center error" reported is the largest distance between an estimated prototype and the sample mean of the matched true group. FCM, pulled outward by the outliers, has a center error of 0.20 on this data set; without the outliers it would be 0.02.

6.4.1 PCM: typicalities, scales, and coincident clusters

With $m = 1.5$ and scales $\eta_i = (2.7, 3.4, 2.6)$ from the FCM run, PCM brings the center error down to 0.04 (fig. 6.4). The maximal typicality separates the two populations cleanly—median 0.97 for the inliers, 0.09 for the outliers—and the 44 samples below a threshold of 0.2 are all true outliers. The contour maps show the difference in kind: the FCM maximal membership never drops below $1/3$ and rises again far from the data, whereas the PCM typicality is a bump of width $\sqrt{\eta_i}$ around every prototype and vanishes elsewhere. Figure 6.5 shows the same along a probe line through two centers, the empirical counterpart of fig. 6.2. The parameter study (fig. 6.6) varies the scale factor K and the exponent. For $m = 1.5$ the center error is below 0.05 for $K = 0.5$ and $K = 1$ (0.08 at $K = 0.25$) and rises to 0.28 at $K = 2$; for $m = 2$ it is already 0.17 at $K = 1$; and for $m = 3$ the typicality decays so slowly that the outliers keep a mean maximal typicality of 0.3 at $K = 1$ and PCM is worse than FCM for every K (center error 0.24 or more). The coincident-cluster failure (fig. 6.7) appears on a group of 200 points next to a group of 40 at a distance of 2.5. With $K = 0.5$ the two prototypes are 2.5 apart; with $K = 1$ they coincide ($|\mathbf{v}_1 - \mathbf{v}_2| = 0.00$) and the small group is lost. FCM never shows this failure, because its memberships compete.

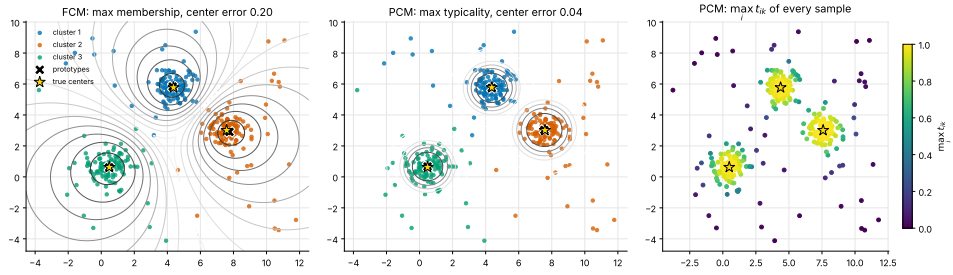


Figure 6.4. FCM memberships (left) and PCM typicalities (middle) on three groups with 60 outliers: the FCM contours tile the plane and the centers are pulled outward, the PCM contours are bumps around the prototypes. The maximal typicality of every sample (right) singles out the outliers; stars are the true group means.

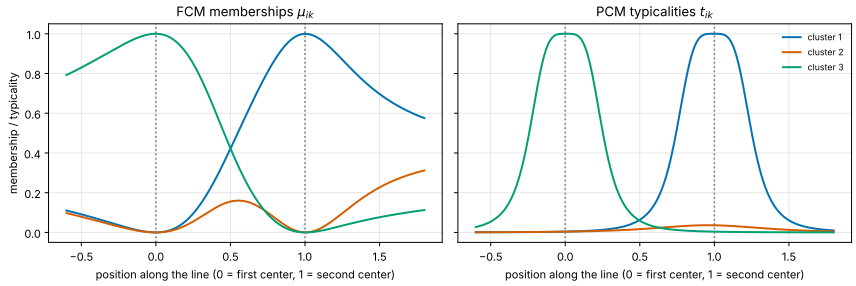


Figure 6.5. Along a line from one center through another and beyond, the FCM memberships cross at the midpoint and stay high beyond the prototypes, whereas the PCM typicalities peak at the prototypes and vanish on both sides.

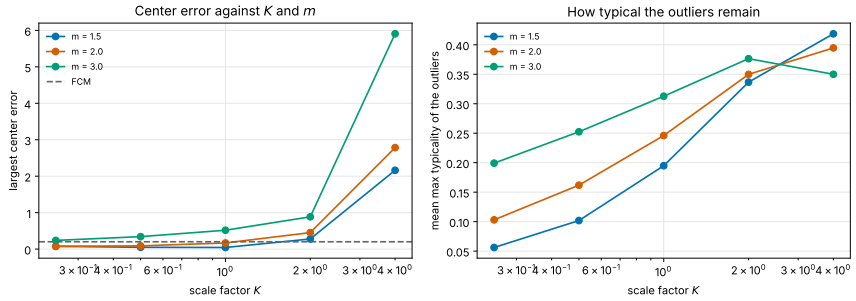


Figure 6.6. Center error (left) and mean maximal typicality of the outliers (right) against the scale factor K for three exponents: $m = 1.5$ with $K \leq 1$ is robust, too large a K lets the outliers back in, and too large an m makes the typicality decay too slowly.

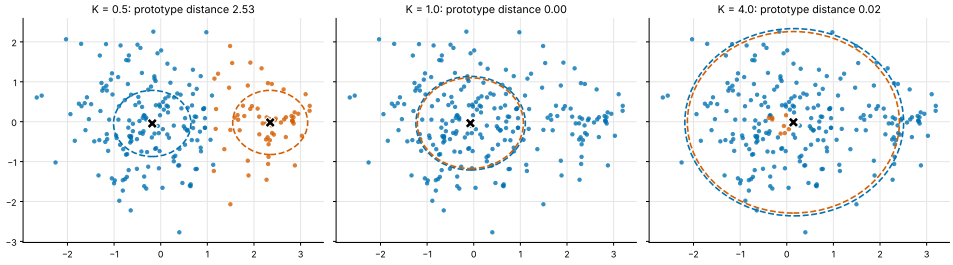


Figure 6.7. Coincident clusters: a large group next to a small, close one. With $K = 0.5$ the two PCM prototypes are separated, with $K = 1$ and beyond they drift into the large group together; the dashed circles have radius $\sqrt{\eta_i}$.

6.4.2 PFCM: the a/b trade-off

PFCM with $a = 1$ and $b = 4$ returns both matrices on the noisy blobs (fig. 6.8): memberships whose rows sum to one and whose contours are the familiar FCM cells, and typicalities that sum to 0.60 per inlier against 0.13 per outlier and flag the outliers as PCM does. The study of the weights (fig. 6.9, left) shows the center error dropping from the FCM value 0.20 as soon as the typicalities enter the update ($b \geq 0.5$ gives 0.05–0.09). The right panel shows what the memberships buy on the coincident-cluster data: PCM merges the two prototypes at $K = 1$, PFCM with $a = b = 1$ keeps them apart at $K = 1$, halves their distance at $K = 2$, and nearly merges them at $K = 4$, and with $a = 20$ the prototypes stay about 2 apart for every scale factor up to 16; fig. 6.10 shows the three partitions at $K = 4$. Raising b alone does not reproduce PCM, because b also tightens the effective scale γ_i/b inside the typicality. The printed study of η and the scale factor confirms that a smaller η makes the outliers less typical (mean maximal typicality 0.05 at $\eta = 1.5$ against 0.22 at $\eta = 3$, for $K = 1$). It also reveals the second effect discussed in section 6.2.2: with tight scales ($K = 0.5$) the center error rises to 0.12–0.13 for every η, because the typicalities of the core samples shrink and the membership term regains weight, whereas $K = 2$ with $\eta = 2$ gives 0.045.

6.4.3 Noise clustering: the multiplier λ

With $\lambda = 0.3$ the noise distance comes out at $\delta = 3.0$ (fig. 6.11): 31 samples are labeled noise, all of them true outliers, the center error is 0.06, and the membership contours of the real clusters close around the groups where those of FCM tile the plane. Along the probe line (fig. 6.12) the real memberships decay and the noise membership takes over beyond δ. The scan over λ (fig. 6.13) maps the trade-off. With $\lambda = 0.1$ ($\delta = 1.7$), 50 of the 60 outliers and one inlier are labeled noise, with a center error of 0.03; $\lambda = 0.02$ declares 128 inliers noise, and the partition of the inliers degrades (ARI 0.34); $\lambda \geq 2$ ($\delta \geq 7.8$) labels nothing noise, and the center error climbs back to 0.14–0.17. The outliers that escape at moderate λ are those that fell inside a group. Figure 6.14 shows three of these partitions with the circles

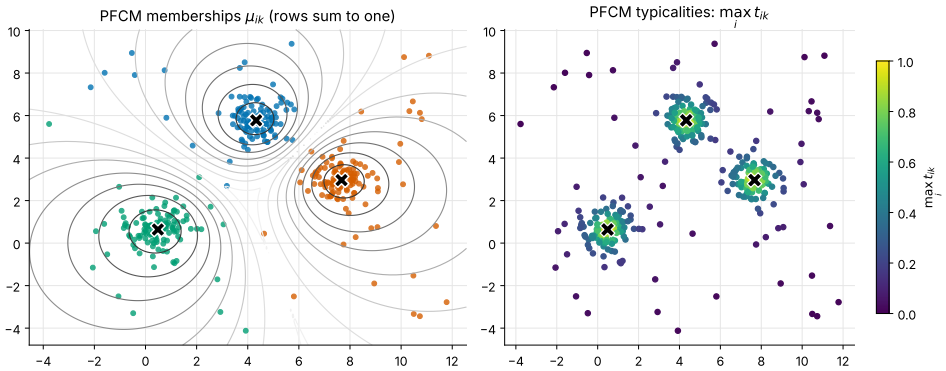


Figure 6.8. PFCM on the noisy blobs: the memberships (left) sum to one and partition the plane, the typicalities (right) are bumps around the prototypes and flag the outliers.

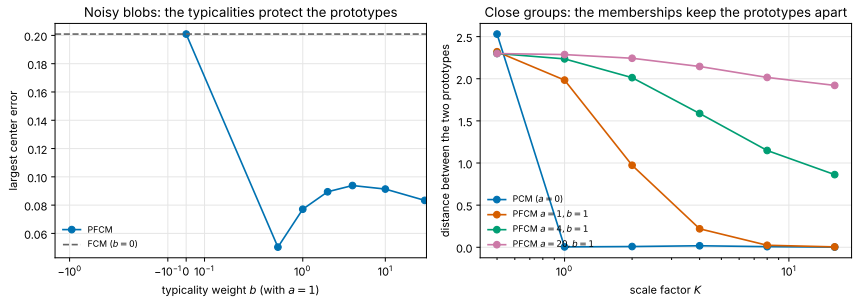


Figure 6.9. Left: center error on the noisy blobs against the typicality weight b (with $a = 1$); $b = 0$ is FCM. Right: distance between the two prototypes on the coincident-cluster data against the scale factor K for PCM and for PFCM with increasing membership weight a.

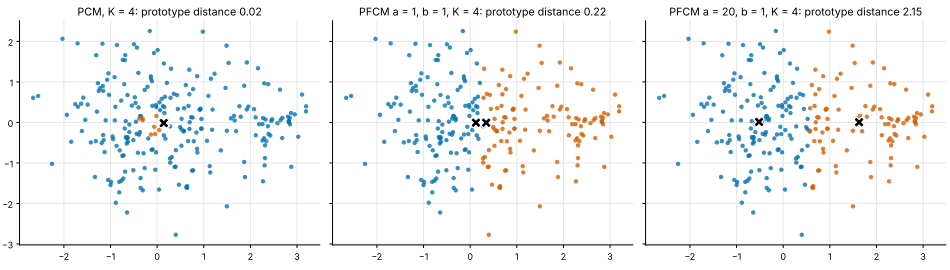


Figure 6.10. Partitions of the coincident-cluster data at $K = 4$: PCM merges the prototypes, PFCM with $a = b = 1$ nearly merges them, and PFCM with $a = 20$ keeps them on their groups.

of radius δ. A fixed $\delta = 10^6$ reproduces FCM exactly (largest noise membership $3 \cdot 10^{-11}$, center error 0.20).

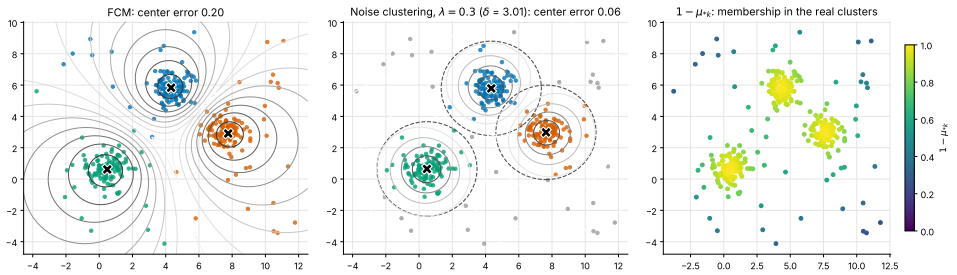


Figure 6.11. FCM (left) and noise clustering with $\lambda = 0.3$ (middle) on the noisy blobs; the dashed circles have radius δ and the gray samples are labeled noise. Right: the membership $1 - \mu_{*k}$ in the real clusters.

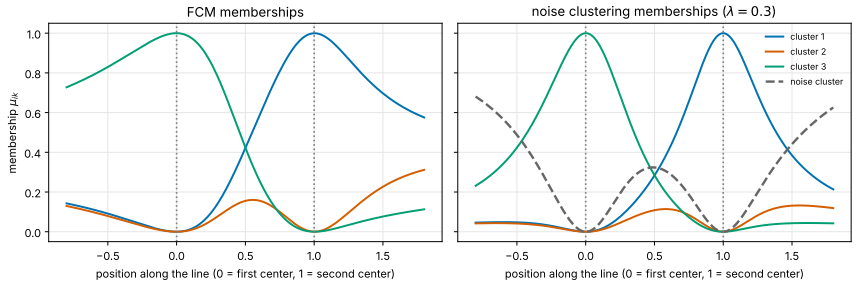


Figure 6.12. Memberships along a probe line through two centers: in noise clustering the real memberships decay and the noise membership (dashed) takes over beyond δ.

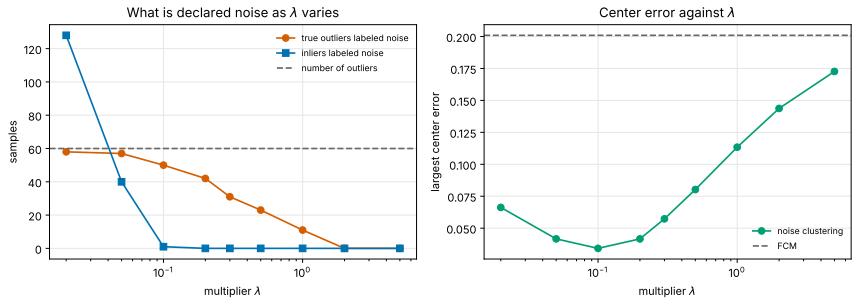


Figure 6.13. Number of true outliers and of inliers labeled noise (left) and center error (right) against the multiplier λ.

6.4.4 Kernel FCM on rings and moons

The data are two concentric rings of radii 1 and 3 and two moons with a gap of about 0.3. FCM reaches adjusted Rand indices of 0.00 and 0.26; kernel FCM with feature-space prototypes recovers both exactly, with $\gamma = 0.3$ and $m = 1.2$ on the

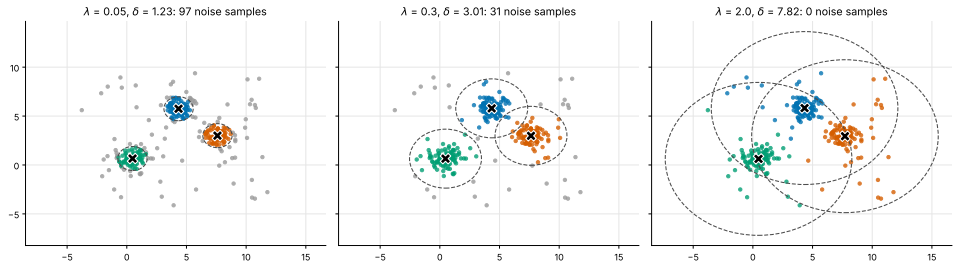


Figure 6.14. Partitions for $\lambda = 0.05$, 0.3, and 2: a small λ peels the tails off the groups, a large one lets every outlier back in.

rings and $\gamma = 15$ and $m = 1.1$ on the moons, each with five restarts (fig. 6.15). The contours of the maximal membership, evaluated on a grid through the kernelized distance (6.9), follow the rings and bend around the moons. The crosses are the approximate pre-images of the prototypes and play no role in the distances; on the rings both lie near the origin. The scan over γ (fig. 6.16) shows the narrow window: on the rings only $\gamma \in [0.2, 0.5]$ succeeds ($\gamma \leq 0.1$ behaves like FCM, $\gamma \geq 1$ fragments the rings), on the moons $\gamma \in [10, 30]$. At the chosen values $1/\sqrt{\gamma}$ is 1.8 and 0.26, the gap between the structures in each case. Figure 6.17 shows the rings at $\gamma = 0.05$, 0.3, and 5, the last with a mean maximal membership of 0.5, that is, an uninformative partition. The exponent matters as much: at $\gamma = 0.3$ the rings are found for $m \leq 1.5$, whereas with $m = 2$ the agreement drops to 0.96 and with $m = 3$ to 0.27, because every feature-space prototype approaches the mean of all mapped samples. A polynomial kernel of degree two or three does not recover the rings (ARI 0.19 and 0.18). In the quadratic monomials a circle is a hyperplane, and one such hyperplane does separate the two rings, but the outer ring is spread over that space far more widely than the inner one, and a partition that splits it has the smaller J; the linear kernel is FCM. The input-space variant (fig. 6.18) cannot find rings but is a robust FCM: on the noisy blobs its center error is 0.03–0.04 for γ between 0.1 and 0.3 and below 0.09 from $\gamma = 0.03$ to 1, against 0.20 for FCM.

6.4.5 Fuzzy c-varieties on crossing lines, and the elliptotype α

The data are three lines of 80 points through a common point, at 0°, 60°, and 120° (fig. 6.19). FCM tiles the plane into three cells (ARI 0.09); GK stretches its ellipsoids but cannot align one with each line without swallowing the centers of the others (ARI 0.33); fuzzy c-lines recovers the three lines (ARI 0.96) with orientations 60.2°, 120.2°, and 179.8° $\equiv -0.2°$. The only ambiguous points are those near the crossing, which are close to all three lines. The membership contours are bands around the lines, and the eigenvalues of the scatter matrices show a spread along each line about 600 times larger than the scatter around it (fig. 6.20). The elliptotype study (fig. 6.21) uses two collinear segments with a gap and an oblique third segment. Pure fuzzy c-lines ($\alpha = 0$) sees the collinear segments as one line and spends a prototype on noise (ARI 0.49), and $\alpha = 0.05$ already separates them (ARI 1.0). On

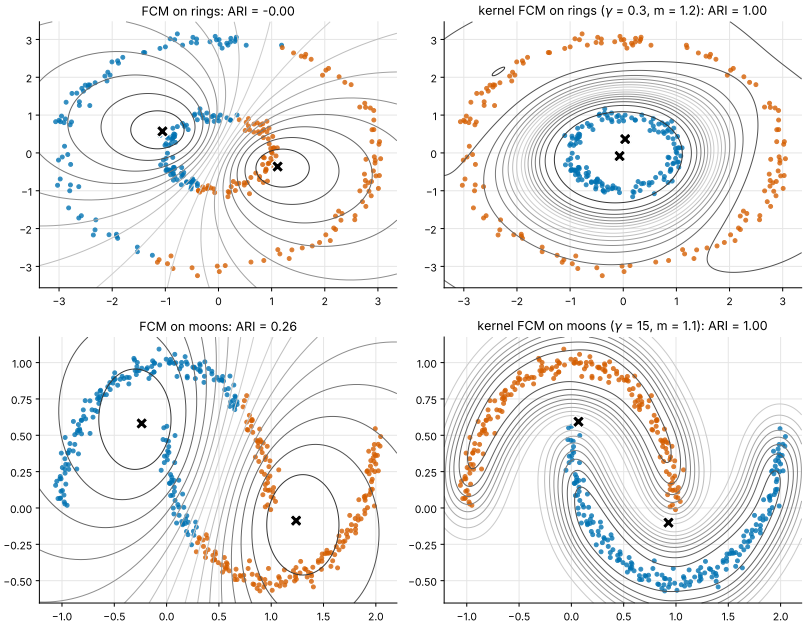


Figure 6.15. FCM (left) and kernel FCM with feature-space prototypes (right) on two rings (top) and two moons (bottom); contours of the maximal membership evaluated through the kernelized distance.

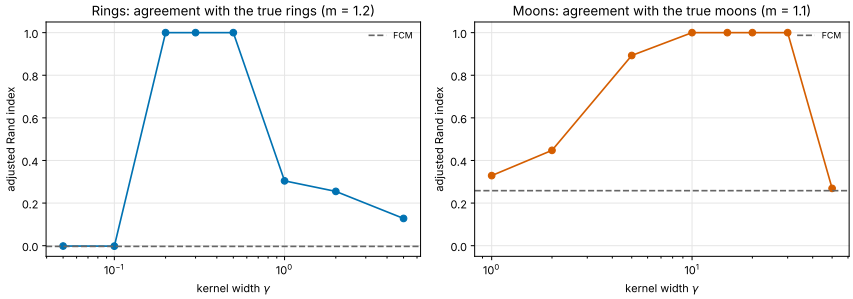


Figure 6.16. Agreement with the true groups against the kernel parameter γ on the rings ($m = 1.2$) and the moons ($m = 1.1$); the dashed line is FCM.

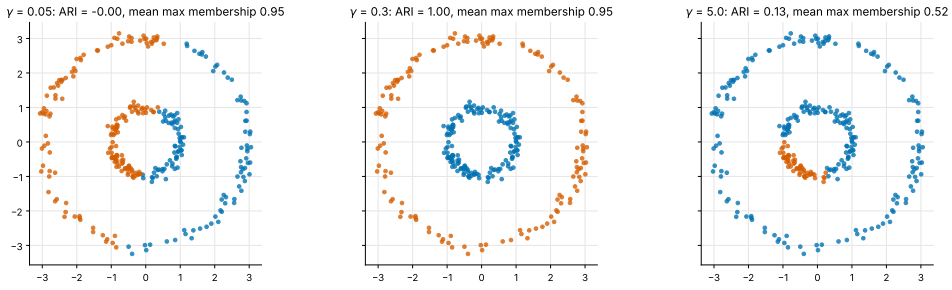


Figure 6.17. The rings with $\gamma = 0.05$ (FCM-like halves), $\gamma = 0.3$ (the rings), and $\gamma = 5$ (orthogonal feature space, arbitrary partition).

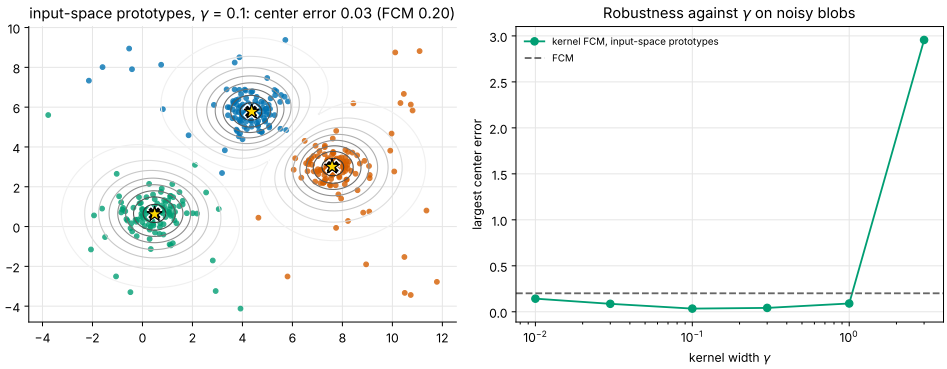


Figure 6.18. Kernel FCM with input-space prototypes on the noisy blobs: the saturating distance makes it a robust FCM (left), for a wide range of γ (right).

the crossing lines, in contrast, the agreement falls from 0.96 at $\alpha = 0$ and 0.95 at $\alpha = 0.05$ to 0.83 at $\alpha = 0.1$ and to 0.2 or less for $\alpha \geq 0.2$: where lines cross, the position along the line must not count. The printed studies show the need for restarts (mean ARI over ten seeds 0.73 with one start, 0.96 with ten), the usual role of m, and two planes in three dimensions recovered with `n_dims=2` (ARI 0.95, normals along the y and z axes).

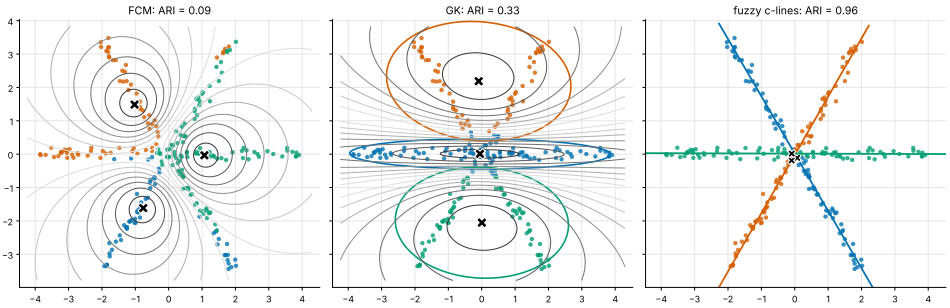


Figure 6.19. Three lines through one point: FCM tiles the plane (left), GK cuts the lines into ellipsoidal pieces (middle), and fuzzy c-lines finds the lines (right).

6.4.6 Fuzzy c-shells: radius recovery

FCM halves two concentric rings (ARI 0.00), whereas FCSS with five restarts recovers them with radii 1.00 and 3.00. On three circles of radii 2, 1.2, and 1, two of which intersect, FCSS started from the FCM-hardened partition finds the centers to within 0.015 and the radii to within 0.007, with an ARI of 0.97; the points near the intersections have shared memberships (fig. 6.22). The membership contours are rings around the fitted circles, and the signed radial residuals of the samples from their own shell are centered at zero, with a spread close to the generating noise level (fig. 6.23). The initialization study (fig. 6.24) shows the two faces of

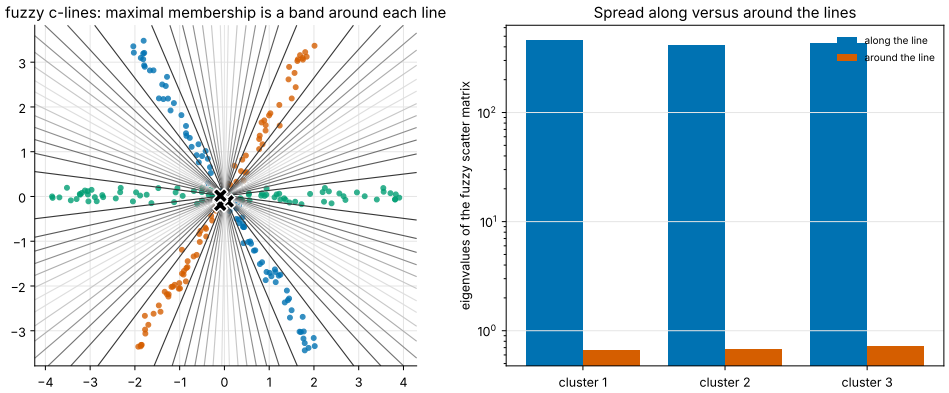


Figure 6.20. Fuzzy c-lines: the maximal membership is a band around each line (left); the eigenvalues of the fuzzy scatter matrices separate the spread along the lines from the scatter around them (right).

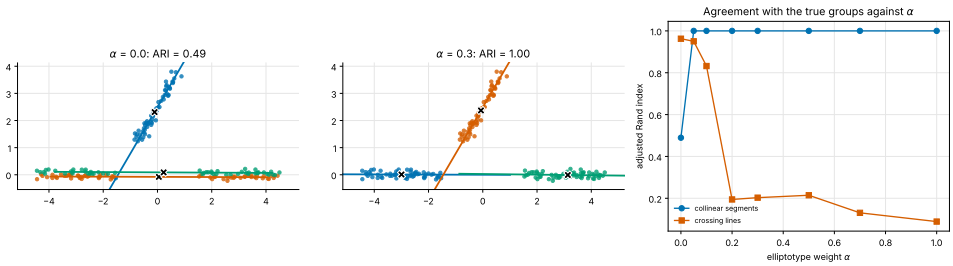


Figure 6.21. Two collinear segments and an oblique one, clustered with $\alpha = 0$ (left; the collinear segments form one line) and with $\alpha = 0.3$ (middle), and the agreement against α for these data and for the crossing lines (right).

the FCM start. On the concentric rings a single run converges to two circles of radius 2.1 and 2.2 through both rings ($J = 746$), five restarts find the rings ($J = 62$), and over ten seeds the FCM start needs ten restarts to succeed every time, while random point prototypes need three. On the intersecting circles, conversely, the FCM start succeeds at once (ARI 0.97) and a single random start fails (ARI 0.08). The comparison of the two algorithms on three concentric rings of radii 1, 1.6, and 3 (fig. 6.25) exposes the small-shell bias of the algebraic distance: both find the rings from the best of ten restarts, but over five seeds FCS reaches a mean ARI of 0.90 against 0.80 for FCSS, because the algebraic distance weights the same radial deviation about three times more on the outer ring than on the inner one. On the intersecting circles the two algorithms agree to within 0.01 in centers and radii for every m, and m only sets how strongly the memberships are shared at the intersections.

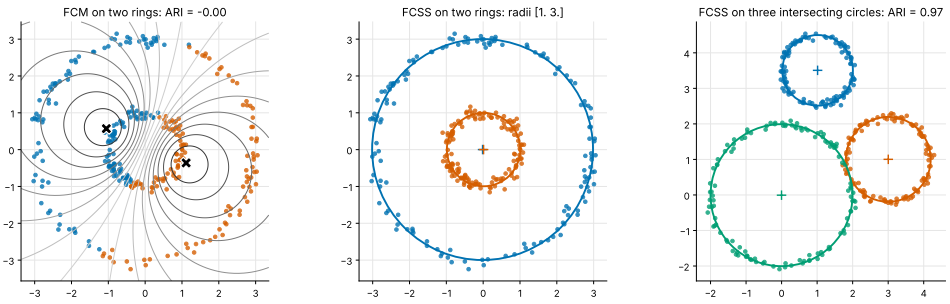


Figure 6.22. FCM on two concentric rings (left) and FCSS on the same rings (middle) and on three circles, two of which intersect (right); the fitted circles and their centers are drawn.

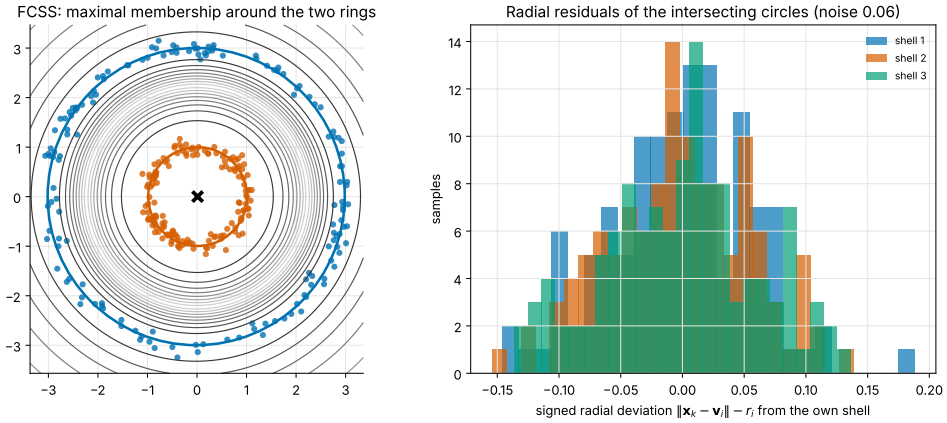


Figure 6.23. Maximal membership around the two fitted rings (left) and the signed radial residuals of the samples from their own shell on the intersecting circles (right).

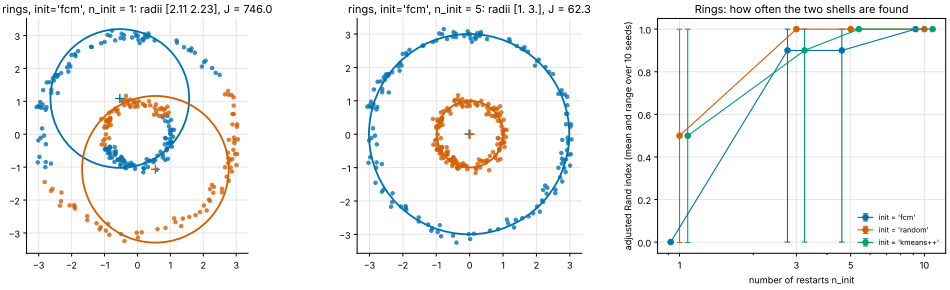


Figure 6.24. Concentric rings from the FCM-hardened start with a single run (left; two circles through both rings) and with five restarts (middle), and the agreement over ten seeds against the number of restarts for three initializations (right).

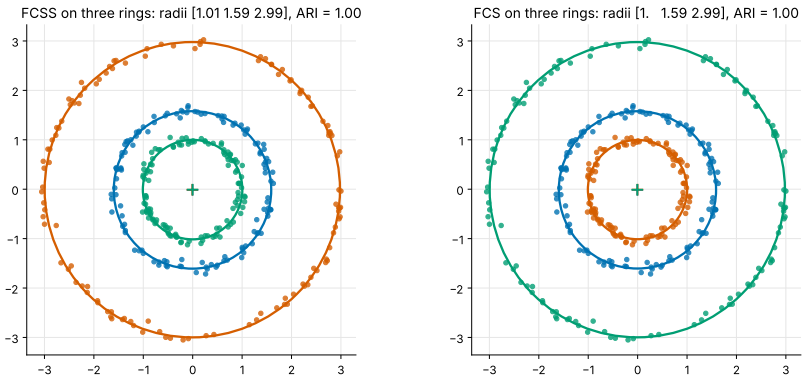


Figure 6.25. Three concentric rings of radii 1, 1.6, and 3 with FCSS (left) and FCS (right), both from the best of ten restarts.

6.4.7 Volume prototypes on clusters of different size

The script `plot_08_volume_prototypes.py` returns to the data of section 3.4.6: a large diffuse group of 300 samples next to two small dense ones. FCM splits the large group and gives a slice of it to each small cluster (adjusted Rand index 0.70); with volume prototypes the cores grow to radii 1.65, 0.83, and 0.56, and the large cluster keeps its members (ARI 0.85; fig. 6.26). The membership profile along the line from the center of the large group to that of the smallest one (fig. 6.27) shows the mechanism. The memberships are one inside both cores, and the border between the clusters, where the two memberships cross, lies at 0.61 of the way instead of 0.43 for FCM, whose small-cluster prototype has in addition been pulled toward the large group.

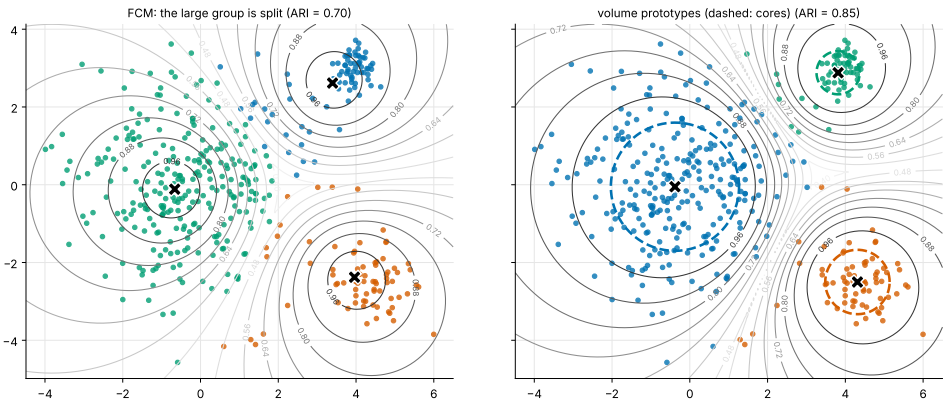


Figure 6.26. FCM (left) and fuzzy clustering with volume prototypes (right) on a large diffuse group and two small dense ones; the dashed circles are the cores, inside which the distance is zero.

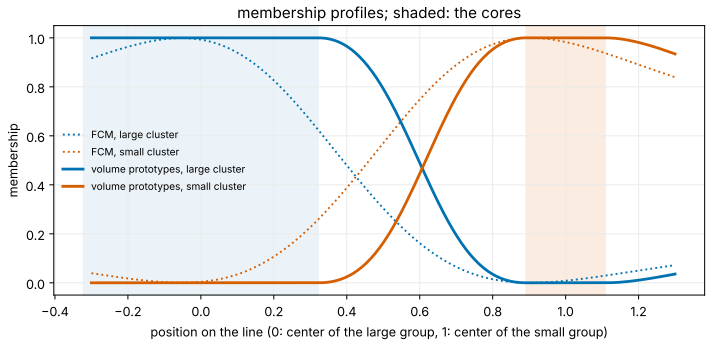


Figure 6.27. Memberships along the line from the center of the large group to the center of the smallest one: with volume prototypes (solid) the memberships are one inside the shaded cores and the border moves toward the small cluster; FCM (dotted) for comparison.

Started with eight clusters, adaptive merging performs five merges, at similarities between 0.26 and 0.34: the first joins the two clusters that share the wider of the two small groups, the other four join pieces of the large group. It ends with three clusters and the same partition as before (fig. 6.28). The parameter study (fig. 6.29) confirms the recommendations of section 6.2.7. The agreement rises with the radius scale from the FCM value 0.70 at $\kappa = 0$ to 0.85 at $\kappa = 1$ and collapses beyond it (0.29 at $\kappa = 1.25$), when a core grows over a neighboring group. The merging threshold has a plateau at three clusters from 0.15 to 0.25; a threshold of 0.30 stops at five clusters, thresholds of 0.35 and above do not merge at all, and 0.10 merges distinct groups down to one cluster. Started with any number of clusters from four to twelve, the merging ends at three.

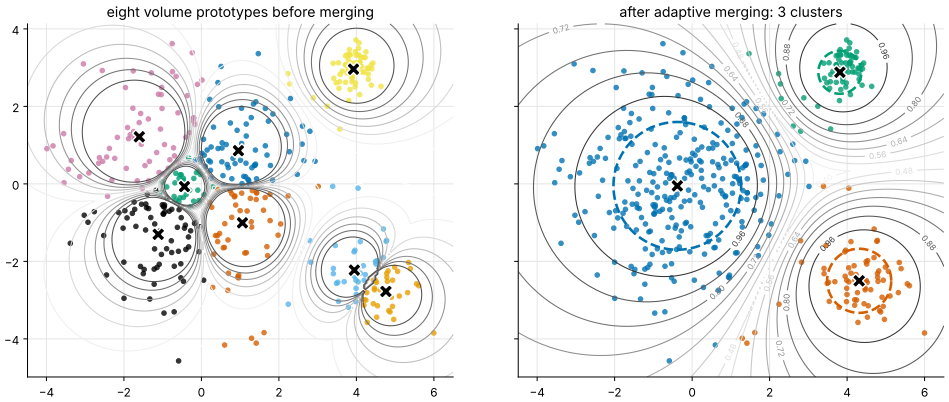


Figure 6.28. Adaptive cluster merging: eight volume prototypes before merging (left) and the three clusters with their cores after five merges (right).

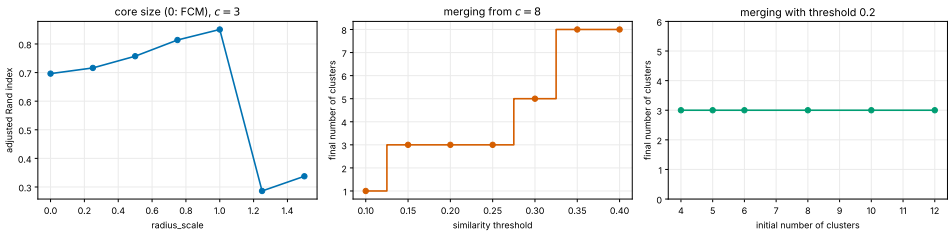


Figure 6.29. Parameter study: agreement with the true groups against the radius scale κ (left), final number of clusters against the similarity threshold (middle) and against the starting number of clusters (right).

6.4.8 Feature weights: noise features and subspace clusters

The script `plot_07_feature_weighting.py` adds six standardized noise features to three groups in two informative features ($m = 1.3$ for all methods, because in eight dimensions the FCM memberships with $m = 2$ are nearly uniform). FCM still finds the groups, but with errors (ARI 0.92); the weighted FCM with $t = 3$ and SCAD-1 recover them exactly (fig. 6.30). The global weights of the weighted FCM are 0.24 and 0.25 for the informative features and 0.09 for every noise feature. The SCAD-1 weights show the same pattern in every cluster, but less sharply (0.16 to 0.18 against 0.09 to 0.12), because the automatic δ_i keeps the weights near uniform unless the scatters differ strongly.

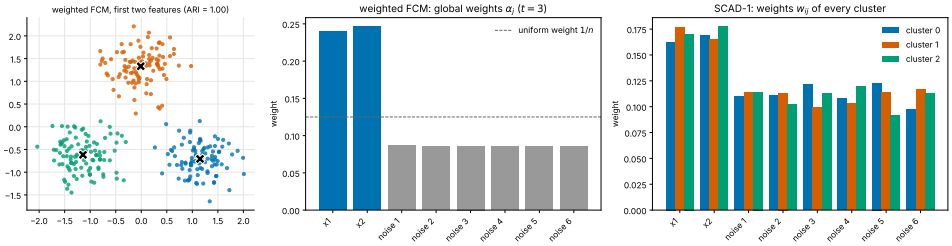


Figure 6.30. Two informative and six noise features. Left: the weighted FCM partition in the informative features. Middle: the global weights α_j concentrate on x_1 and x_2. Right: the SCAD-1 weights of every cluster.

On three clusters that live in different subspaces (a band tight along x_1, a band tight along x_2, and a compact blob), SCAD-2 assigns 0.98 and 0.99 of the weight to the feature along which each band is tight and splits the weight of the blob evenly (0.49, 0.51); the level sets of the weighted distances are ellipses stretched along the feature a cluster ignores (fig. 6.31). The partition itself is no better than that of FCM (ARI 0.90 for both, because the ends of the bands are ambiguous); what SCAD adds is the description of every cluster by its relevant features. The parameter study (fig. 6.32) varies the number of noise features. FCM degrades steadily (ARI 0.96 with four, 0.81 with eight), the weighted FCM stays at 1.00 and SCAD-1 at 0.98 or above up to eight, and from ten noise features on all methods

fail, because the partition of the first iterations is already dominated by noise and the weights are learned from it. The exponent study shows the concentration trap: with $t = 1.5$ or 2 the weighted FCM puts 0.98 or 0.80 of the weight on the informative features but cuts the data along a single one of them (ARI 0.47), while every $t \geq 2.5$ finds the groups.

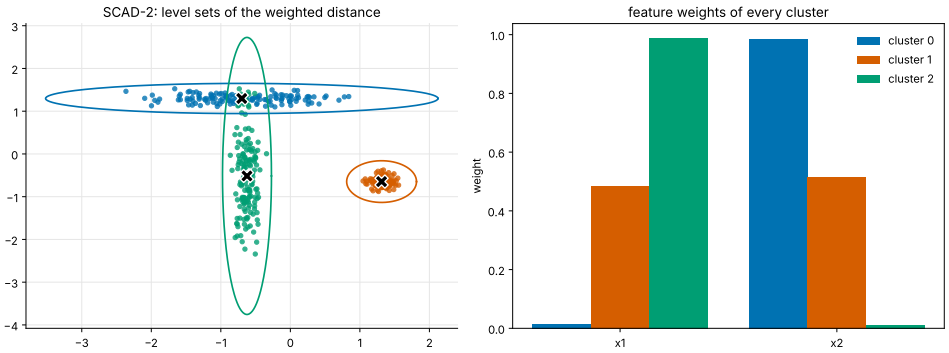


Figure 6.31. SCAD-2 on clusters in different subspaces: level sets of the weighted distance of every cluster (left) and the weights (right); each band weights the feature along which it is tight.

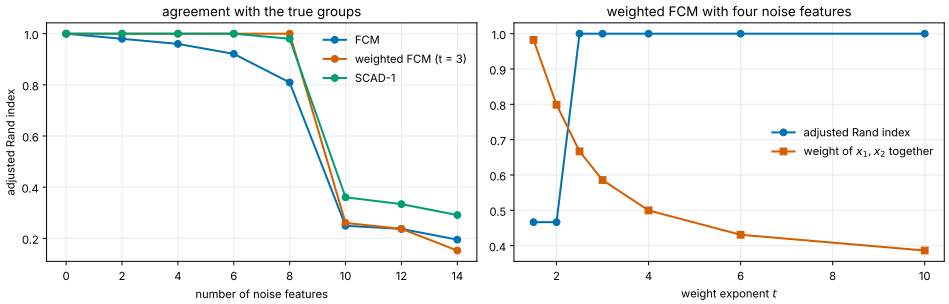


Figure 6.32. Parameter study: agreement with the true groups against the number of noise features for FCM, the weighted FCM, and SCAD-1 (left), and agreement and total weight of the informative features against the weight exponent t of the weighted FCM with four noise features (right).

6.5 Further reading

The possibilistic c-means was introduced in [14], and its insights and recommendations (the exponent, the fixed scales, the coincident clusters) are in [65]; the possibilistic fuzzy c-means is from [63]. Noise clustering is from [64], and the unified view of PCM, noise clustering, and robust statistics [67] is given in [66]. Kernel fuzzy c-means with input-space prototypes is from [71], and its application to incomplete data from [79]; the feature-space variant is the fuzzy counterpart of

the kernel k-means of [70]; the comparative study of both against FCM is [69]; and kernels in general are treated in [68]. Fuzzy c-lines and fuzzy c-varieties with their convex combinations are introduced in the two papers [72, 73], within the framework of Bezdek's book [3]. Fuzzy c-shells is from [74] and the fuzzy c-spherical shells algorithm from [75]; the family of shell algorithms for ellipses and quadrics, together with their possibilistic versions, is surveyed in [80]. The Gustafson–Kessel algorithm [18], with which the c-varieties are compared, is treated in section 3.2.4, and the graph-based methods that find arbitrary shapes without a prototype follow in chapter 7. Volume prototypes and adaptive cluster merging are due to Kaymak and Setnes [43]; the similarity by which the toolbox merges is that of [42]. The weighted fuzzy c-means with global attribute weights is the fuzzy form of the variable-weighted k-means of Huang et al. [76]; cluster-specific weights with an exponent are due to Keller and Klawonn [77], and simultaneous clustering and attribute discrimination, with its regularized and exponent forms, to Frigui and Nasraoui [78].

CHAPTER

7



Graph-based fuzzy clustering

Every algorithm so far described a cluster by a prototype (a point, an ellipsoid, a shell, a linear variety) and measured the distance of the samples to it. This chapter starts instead from a *graph* over the samples and presents two methods of the Abonyi group that find clusters of arbitrary shape and return to a prototype description where one is useful. The first builds the minimal spanning tree of the data, removes inconsistent edges by Zahn's criteria, splits the subtrees further as long as the total fuzzy hypervolume decreases, and hands the crisp partition to Gath–Geva clustering, which turns it into a mixture of Gaussians with memberships, covariances and a similarity structure among the clusters. The second keeps the c-means functional but replaces the Euclidean distance by the geodesic distance along a nearest-neighbor graph and the centers by medoids, so that the clusters follow the manifold the data lie on. Both are in `fuzzyclust.graph`, need only SciPy's sparse-graph routines, and reuse the Gath–Geva clusterer and the volume-based validity indices of earlier chapters.

7.1 Problem and idea

Prototype-based clustering answers the question "which prototype is this sample closest to?" and therefore inherits the geometry of the prototype: fuzzy c-means finds spheres, Gustafson–Kessel and Gath–Geva find ellipsoids, and the shell and variety algorithms find shells and subspaces. When the groups have none of these shapes (a chain of points, a bent arm, two interleaved spirals, rings), the prototypes can only tile the groups into pieces, and the pieces are not the clusters a human sees. Hierarchical and graph-theoretic clustering take the opposite view: a cluster is a set of samples that are connected to each other through short steps and separated from the rest by long ones, whatever the shape of the set. Zahn's classical formulation [81] uses the *minimal spanning tree* (MST) of the data: the

tree connects all N samples with $N - 1$ edges of minimal total length, so that removing k edges leaves $k + 1$ subtrees that are candidate clusters (fig. 7.1, top). The tree keeps only $N - 1$ of the $N(N - 1)/2$ pairwise distances, which makes the method cheap, and single-linkage clustering amounts to removing its edges in decreasing order of length [82].

Two problems arise. The first is deciding which edges are "inconsistent". The longest edges separate clusters only when all clusters have the same density: a global threshold either leaves tight groups merged or shatters diffuse ones, and the local comparison with neighboring edges that Zahn proposed requires the user to choose a factor and a neighborhood depth. The second is that no length criterion can detect a *chain*: a thin sequence of points between two groups keeps every edge short, so the groups remain connected (fig. 7.1, bottom); this is the chaining effect of single linkage. Vathy-Fogarassy, Feil and Abonyi [83] solve both problems by combining the tree with fuzzy clustering. The tree is first cut by Zahn's criteria; the subtrees are then split further at the edge that minimizes the total *fuzzy hypervolume* of the resulting clusters, which is Gath and Geva's validity measure [21] used as a splitting criterion. A union of two compact groups, with or without a chain between them, has a far larger volume than its parts, whereas halving a Gaussian group increases the summed volume, so chains and bends are cut and Gaussian groups are left alone. Finally, the crisp tree partition initializes Gath–Geva clustering, which is too sensitive to its start to be run from a random partition on such data but turns a good start into a fuzzy, parametric description: centers, covariance matrices, memberships, and a similarity among the clusters that organizes them into a dendrogram.

The second method addresses a different failure of the prototype. Even where the groups are clear, the *distance* may be wrong: when the data lie on a curved low-dimensional manifold, such as a spiral arm (fig. 7.2) or an S-shaped sheet (fig. 7.18), two samples that are close in the Euclidean sense may be far apart *along* the manifold, and a center that is the mean of samples on both sides of a fold lies in empty space. Feil and Abonyi [84] replace the Euclidean distance by the *geodesic* distance: the length of the shortest path in a graph that links every sample with its k nearest neighbors, with the edges weighted by their Euclidean lengths. This is the distance Isomap [85] uses to unfold manifolds, and it converges to the true geodesic as the sampling becomes dense [86]. Because geodesic distances exist only between samples, the prototypes are restricted to samples, which makes the algorithm a fuzzy c-medoid method in the spirit of Kaufman and Rousseeuw [5] and places the medoids on the manifold automatically. The only structural parameter is k: if it is too small, the manifold falls apart into several graph components; if it is too large, edges "short-circuit" arms that come close to each other.

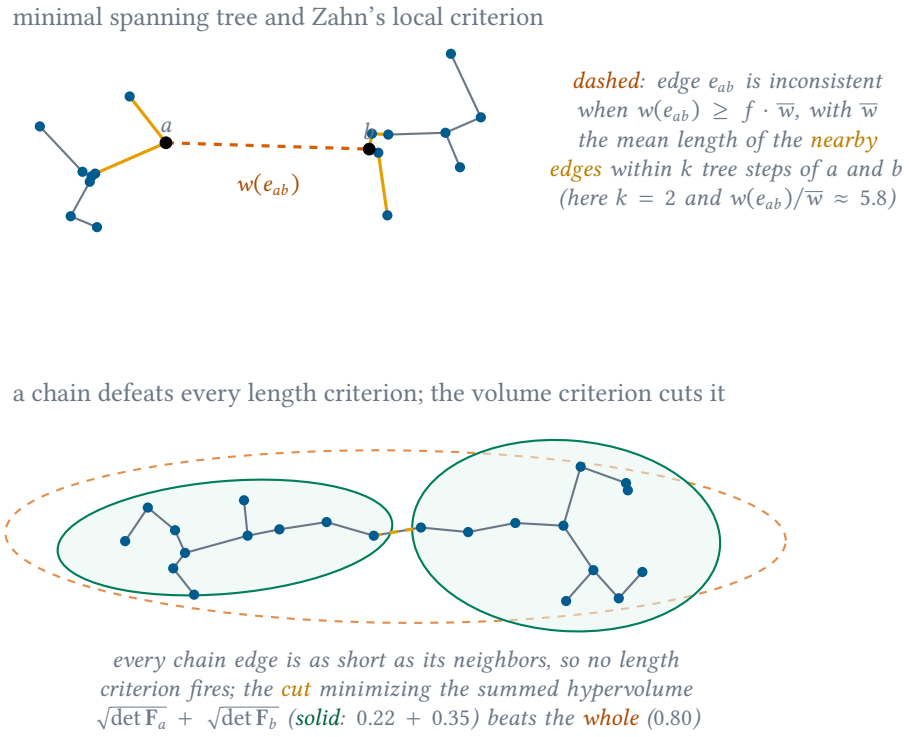


Figure 7.1. Clustering with the minimal spanning tree. Top: the tree of two groups; the long edge e_{ab} between them is inconsistent because it is much longer than the edges in its tree neighborhood (Zahn's local criterion), which also works when the groups have different densities. Bottom: a chain of points between two groups keeps every edge short and defeats all length criteria; the hybrid algorithm cuts the edge whose removal minimizes the summed fuzzy hypervolume of the two parts, because two compact groups have a much smaller volume than their elongated union.

7.2 Theory

The notation is that of the rest of the book: $\mathbf{X} \in \mathbb{R}^{N \times n}$ with rows $\mathbf{x}_k$, memberships μ_{ik} in the matrix $\mathbf{U} \in \mathbb{R}^{c \times N}$ (stored transposed in the code), prototypes $\mathbf{v}_i$ with index i (or j), samples with index k. We treat the hybrid MST–GG algorithm first (sections 7.2.1 to 7.2.3), then the geodesic distance (section 7.2.4) and the two clustering algorithms built on it (sections 7.2.5 and 7.2.6).

7.2.1 The minimal spanning tree and Zahn's cutting criteria

Let $G = (V, E)$ be the complete weighted graph on the samples, with edge weights $w(e_{kl}) = d(\mathbf{x}_k, \mathbf{x}_l)$ (Euclidean here; any metric would do). A spanning tree is a connected subgraph without cycles that reaches every vertex; the minimal

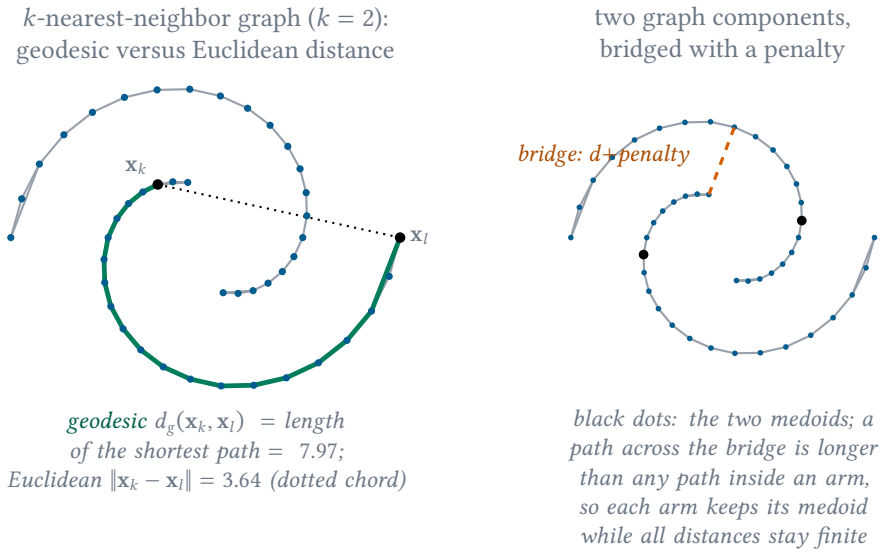


Figure 7.2. Geodesic distance on a neighborhood graph. Left: two spiral arms sampled by 22 points each, linked to their two nearest neighbors. The samples $\mathbf{x}_k$ and $\mathbf{x}_l$ are 3.6 apart in the plane but 8.0 along the arm; a Euclidean prototype between them would sit in empty space, whereas a medoid with geodesic distances stays on the arm. Right: with $k = 2$ the arms are separate components of the graph. They are bridged through their closest pair of points with a penalty equal to the largest within-component distance, so that all distances are finite but the two arms remain geodesically far apart.

spanning tree $T = (V, E_T)$ is the one with the smallest total weight,

$$\sum_{e \in E_T} w(e) \;=\; \min, \qquad |E_T| = N - 1, \tag{7.1}$$

computable by Prim's algorithm in $O(N^2)$ time or by Kruskal's in $O(N^2 \log N)$ [87, 88] (`fuzzyclust` uses SciPy's implementation on the dense distance matrix). Removing k edges from T leaves $k + 1$ subtrees, and the sets of their vertices are the clusters; if δ is the length of a removed edge, no point on one side of it is closer than δ to a point on the other side. Clustering with the MST is therefore divisive hierarchical clustering, and the whole question is *which* edges to remove.

Criterion 1 (global threshold). Remove every edge longer than a multiple of the mean edge length,

$$w(e) > \delta = \lambda \, \frac{1}{N-1} \sum_{e' \in E_T} w(e'), \tag{7.2}$$

or remove the longest edges one by one until the desired number of clusters is reached. Both forms presuppose a single density: the mean is dominated by the largest group, and a tight pair of groups sits below the threshold while the tails of a diffuse group are cut off (fig. 7.7).

Criterion 2 (local inconsistency). Zahn [81] compares an edge with its surroundings instead. Let $\mathcal{N}_k(e_{ab})$ be the set of tree edges within k steps of either endpoint a or b of the edge (not counting e_{ab} itself), with mean length $\overline{w}$ and standard deviation σ_w. The edge is *inconsistent* when

$$w(e_{ab}) \geq f\,\overline{w} \quad \text{(ratio form)} \qquad \text{or} \qquad w(e_{ab}) > \overline{w} + s\,\sigma_w \quad \text{(deviation form)}, \tag{7.3}$$

with a factor f (typically 2–3) or a number of standard deviations s. The hybrid algorithm uses the ratio form; the deviation form, Zahn's second proposal, needs a deeper neighborhood, because the standard deviation of a handful of edges is a poor reference. The reference length adapts to the surroundings of every edge, so groups of widely different density are separated (fig. 7.1, top), at the price of two parameters that interact: the deeper the neighborhood, the more stable the reference and the wider the range of factors that works. Both criteria cut edges in one pass; `fuzzyclust` applies the flagged cuts longest first and refuses a cut that would create a component smaller than `min_cluster_size`, which keeps single outliers from becoming clusters.

7.2.2 Criterion 3: splitting by the fuzzy hypervolume

Length criteria compare distances and are blind to chains. The hybrid algorithm therefore judges a candidate partition by the *quality of the clusters it produces*, using a validity measure as the splitting criterion. Varma and Simon [89] used the Fukuyama–Sugeno measure [33] in this role; Vathy-Fogarassy, Feil and Abonyi take the fuzzy hypervolume of Gath and Geva [21], because the clusters will be refined by Gath–Geva clustering, to which the hypervolume belongs. For a partition $\mathbf{U}$ with centers $\mathbf{v}_j$, the fuzzy covariance matrix, the hypervolume of cluster j and the total hypervolume are

$$\mathbf{F}_j = \frac{\sum_{k=1}^{N} \mu_{jk}^{m} (\mathbf{x}_k - \mathbf{v}_j)(\mathbf{x}_k - \mathbf{v}_j)^{\mathsf{T}}}{\sum_{k=1}^{N} \mu_{jk}^{m}}, \qquad V_j = \sqrt{\det \mathbf{F}_j}, \qquad FHV = \sum_{j=1}^{c} V_j, \tag{7.4}$$

evaluated during the tree phase with the crisp memberships of the tree partition. The alternative criterion of Varma and Simon, in its crisp form,

$$FS = \sum_{j=1}^{c} \sum_{\mathbf{x}_k \in S_j} \left(\left\| \mathbf{x}_k - \mathbf{v}_j \right\|^2 - \left\| \mathbf{v}_j - \bar{\mathbf{v}} \right\|^2 \right), \tag{7.5}$$

with $\bar{\mathbf{v}}$ the global mean, rewards compact clusters with distant centers and is available as `criterion="fukuyama_sugeno"`.

The splitting starts with the subcluster of largest volume V_i. Each of its $N_i - 1$ tree edges is removed in turn, which produces a binary split into a and b and a candidate partition with total volume $FHV - V_i + V_a + V_b$; the cut with the smallest

total volume is the best split of this subcluster. `fuzzyclust` accepts the best split when its *relative* gain,

$$\frac{V_i - V_a - V_b}{V_i} > \texttt{min_gain}, \tag{7.6}$$

exceeds a threshold (0.15 by default) and each part has at least `min_cluster_size` samples; otherwise the next largest subcluster is tried, and the procedure stops when no subcluster can be improved or `max_clusters` is reached. The threshold is the toolbox's reading of the paper's "special parameters that control the generalization level of the hierarchical clustering tree": cutting a few tail points off a compact group always lowers the volume a little, and the relative gain separates such cosmetic cuts from real structure. The criterion spares Gaussian groups and breaks chains. Halving a Gaussian group along its principal axis reduces the variance along that axis to $(1 - 2/\pi)\sigma^2 \approx 0.36\,\sigma^2$, so each half keeps about 0.6 of the volume and the two halves together have 1.2 times the volume of the whole; the split brings no gain. For two compact groups joined by a chain, the determinant of the union is inflated by the distance between the groups, and the parts are much smaller (fig. 7.1, bottom). A curved group is split into arcs for the same reason, and this is intended: every final cluster is then well approximated by one Gaussian, which is the representation the refinement needs. Evaluating all $N_i - 1$ cuts is cheap: with the subtree rooted at any vertex, the count, the sum of $\mathbf{x}_k$ and the sum of $\mathbf{x}_k\mathbf{x}_k^\mathsf{T}$ below every vertex are accumulated in one pass from the leaves to the root, and the statistics of both sides of any edge follow by subtraction, so the statistics of all candidate cuts cost $O(N_i n^2)$, plus two $n \times n$ determinants (one for each side) per cut.

When the number of clusters is *given* (`n_clusters=c`), the same machinery is steered toward it: coarse cuts of Criteria 1–2 beyond c are undone shortest first, and the volume splitting continues with the best available cut, gain or no gain, until c clusters exist. This makes the estimator usable in the validity scans of the toolbox; the scan of the volume indices over c shows a knee at the number where the automatic rule stops (fig. 7.10).

7.2.3 Gath–Geva refinement and the analysis of the result

The tree partition is crisp and carries no model. In Step 4 of the algorithm it initializes Gath–Geva clustering with as many clusters as the tree partition has: the crisp membership matrix is the starting $\mathbf{U}$, and the GG iterations (prior α_i, center $\mathbf{v}_i$, fuzzy covariance $\mathbf{F}_i$, exponential distance, membership update) run to convergence as in section 3.2.5. GG is a maximum-likelihood estimator of a Gaussian mixture and is notoriously sensitive to its initialization: started from FCM or at random on a chained or curved data set, it merges a group with a chain or splits a group. Here, however, it only has to *polish* the tree partition, and it does so without leaving it (fig. 7.5). The result is a fuzzy partition with centers, covariance matrices and priors: a compact parametric description of clusters whose shapes

the tree found, and the basis of the membership contours and covariance ellipses in the figures. (FCM as refinement, `refine="fcm"`, is pulled off the tree partition by the chain points; `refine=None` keeps the crisp partition with the covariances of the tree clusters.)

Two measures evaluate the result. The adequacy of a cut e_{ab} that produced clusters $C_a \ni a$ and $C_b \ni b$ is read off the refined memberships,

$$CP(e_{ab}) = \sum_{k\in\{a,b\}} \left|\mu_{ik} - \mu_{jk}\right|, \qquad i = \text{cluster of } a,\ j = \text{cluster of } b, \tag{7.7}$$

which is 2 when the two endpoints belong firmly to different clusters and 0 when the refinement has re-united them (`cut_quality_`). The clusters themselves are compared as fuzzy sets over the samples by the ratio of fuzzy intersection and union,

$$S_{ij} = \frac{\sum_{k=1}^{N} \min(\mu_{ik}, \mu_{jk})}{\sum_{k=1}^{N} \max(\mu_{ik}, \mu_{jk})} = \frac{|A_i \cap A_j|}{|A_i \cup A_j|}, \tag{7.8}$$

a symmetric matrix with unit diagonal (`cluster_similarity_`). Used as the distance $1 - S_{ij}$ in single-linkage clustering, it yields a dendrogram of the clusters: pieces of one nonconvex group share samples with intermediate memberships and join at a low level, while pieces of different groups do not, so the dendrogram shows which clusters may be merged and how they are related in the original space (fig. 7.12). GG memberships are nearly crisp, so the similarities are small (the dendrogram in the paper also spans only the range from 0.96 to 1), but their order carries the structure.

7.2.4 Geodesic distances on a neighborhood graph

Let the samples lie on (or near) a smooth manifold $M \subset \mathbb{R}^n$ of lower dimension. The intrinsic distance between two samples is the length of the shortest curve *on* M connecting them, the geodesic distance d_M, which can differ arbitrarily from the Euclidean chord (fig. 7.2, left). Isomap [85] approximates it in two steps. A neighborhood graph links each sample with its k nearest neighbors (k-Isomap) or with all samples within a radius ε (ε-Isomap), every edge weighted by its Euclidean length; the graph distance $d_g(\mathbf{x}_k, \mathbf{x}_l)$, the length of the shortest path found by Dijkstra's or Floyd's algorithm [90, 91], then stands in for d_M. Bernstein et al. [86] prove that $d_g \to d_M$ as the sampling density grows: short edges follow the manifold, and the shortest path cannot cut through empty space because there are no edges there. The approximation is good when k (or ε) is large enough for the graph to be connected along the manifold and small enough that no edge jumps across a gap between folds or arms (a "short-circuit"). The scale-free k is easier to set than ε, and the number of connected components of the graph is the first diagnostic.

Components are the practical difficulty: if the graph is disconnected, the geodesic distance between components is undefined (infinite). The paper treats the

Algorithm 7.1 Hybrid MST–GG clustering

Input: data $\mathbf{X}$; threshold λ and/or factor f (or s) with depth k; minimum gain and minimum cluster size; optional number of clusters c; exponent m for the refinement

build the minimal spanning tree T of $\mathbf{X}$ (7.1);
`// Step 0: Criteria 1 and 2`
flag the edges that satisfy (7.2) or (7.3); remove them longest first, skipping cuts that would create a component below the minimum size;
if *c is given and there are more components* **then** restore the shortest removed edges until c remain;
`// Steps 1–3: volume-based binary splitting`
repeat
  for *the subtrees in decreasing order of* $V_i = \sqrt{\det \mathbf{F}_i}$ **do**
    for every edge of the subtree compute $V_a + V_b$ of the two sides from the subtree sums (7.4);
    take the edge with the smallest $V_a + V_b$ whose parts are large enough;
    if $(V_i - V_a - V_b)/V_i >$ `min_gain` *(or c is given and not yet reached)* **then** cut it and restart the loop;
until *no split is accepted (or c clusters exist)*;
`// Step 4: refinement and analysis`
initialize Gath–Geva with the crisp tree partition and iterate to convergence;
cut adequacies $CP(e)$ (7.7), cluster similarities S_{ij} (7.8);

Output: tree partition, fuzzy partition $\mathbf{U}$, centers $\mathbf{v}_i$, covariances $\mathbf{F}_i$, removed edges with phase and adequacy, similarity matrix

components of the spiral as distinct manifolds that the clustering should keep apart. `fuzzyclust` follows this and makes it operational: components are linked through their closest pair of points by a *bridge* whose length is the Euclidean gap plus a penalty, by default equal to the largest finite geodesic distance of the unbridged graph (fig. 7.2, right). Every distance is then finite, and a path across a bridge is longer than any path inside a component, so clusters do not span components, while a component without a medoid still receives memberships through its bridge instead of being lost. The penalty can be set to zero when the components are known to be fragments of one manifold (k too small), in which case the plain bridge re-joins them at their natural junctions (fig. 7.20).

In a nutshell Hybrid MST–GG clustering

- Divisive clustering on the minimal spanning tree: Zahn's global and local length criteria separate distinct groups (also of different density); the fuzzy hypervolume criterion cuts chains and bends that length criteria cannot see and spares Gaussian groups.
- The tree partition initializes Gath–Geva clustering, which turns it into a mixture of Gaussians with memberships and covariances without destroying it; nonconvex groups come out as Gaussian pieces.
- Parameters: factor f and depth k of the local criterion ($f = 3$–4 with $k \geq 3$ in the examples), or a global λ; `min_gain` and `min_cluster_size` for the splitting; `n_clusters` to steer toward a given number.
- Analysis: cut adequacy $CP(e)$, similarity S_{ij} of the fuzzy clusters and its dendrogram for merging and visualization; $O(N^2)$ memory for the distance matrix.

7.2.5 Algorithm II: geodesic distance based fuzzy c-medoids

The clustering minimizes the c-means functional of Bezdek [3] with geodesic distances and prototypes restricted to samples,

$$J = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^{m} \, d_g(\mathbf{x}_k, \mathbf{v}_i)^2, \qquad \mathbf{v}_i \in \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}, \tag{7.9}$$

under the usual constraints $\sum_i \mu_{ik} = 1$. The restriction is forced by the distance, since d_g exists only between samples, and it is welcome: a medoid lies on the manifold by construction, whereas a mean of samples on both sides of a fold does not (fig. 7.18). The alternating optimization has the two familiar steps. For fixed medoids, the Lagrange conditions give the FCM membership formula with d_g,

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{c} \left(d_g(\mathbf{x}_k, \mathbf{v}_i) / d_g(\mathbf{x}_k, \mathbf{v}_j) \right)^{2/(m-1)}}, \tag{7.10}$$

and for fixed memberships every sample is tried as the medoid of every cluster, and the one with the smallest weighted sum of squared geodesic distances is taken,

$$\mathbf{v}_i = \mathbf{x}_j, \qquad j = \arg\min_l \sum_{k=1}^{N} \mu_{ik}^{m} \, d_g(\mathbf{x}_k, \mathbf{x}_l)^2, \tag{7.11}$$

which is one matrix product $\mathbf{D}_g^{(2)} (\mathbf{U}^m)^\mathsf{T}$ of the squared distance matrix with the powered memberships, $O(cN^2)$ per iteration. The iterations stop when the medoids no longer change (or $\max |\Delta \mu_{ik}| < \epsilon$). The paper starts from c arbitrary samples. `fuzzyclust` defaults to a D^2-weighted seeding on geodesic distances (`init="kmeans++"`), because a medoid rarely migrates across a penalized bridge

Algorithm 7.2 Geodesic distance based fuzzy c-medoid clustering (Algorithm II)

Input: data $\mathbf{X}$, number of clusters c, neighbors k (or radius ε), exponent m, tolerance ϵ

build the k-nearest-neighbor graph with Euclidean weights; find its connected components;
if *several components* **then** bridge them through their closest pairs with the penalized length;
geodesic distances $d_g(\mathbf{x}_k, \mathbf{x}_l)$ for all pairs by shortest paths;
choose c samples as initial medoids (D^2 seeding or arbitrary);
repeat
    memberships μ_{ik} by (7.10) from the geodesic distances to the medoids;
    for *every cluster* i **do**
        new medoid by (7.11): the sample minimizing $\sum_k \mu_{ik}^m d_g(\mathbf{x}_k, \cdot)^2$
until *the medoids do not change or* $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: medoids $\mathbf{v}_i$ (indices into $\mathbf{X}$), partition $\mathbf{U}$, J, the graph and the geodesic distance matrix

during the iterations: a random start that puts every medoid into one graph component leaves the other components without a prototype. Random starts with restarts are available as well. The dense geodesic distance matrix and the exhaustive medoid search limit the method to a few thousand samples; the paper points to sampling schemes such as CLARA [5] for larger data sets. A new sample has no geodesic distance of its own; `fuzzyclust` enters the graph at the nearest training sample and adds the Euclidean distance to it, which is exact for training points and adequate close to the manifold.

7.2.6 Algorithm I: clustering of the Isomap projection

The paper's first route leaves the manifold entirely to Isomap and clusters its output. Classical multidimensional scaling [92] of the geodesic distance matrix $\mathbf{D}_g$ gives coordinates $\mathbf{y}_k \in \mathbb{R}^q$ that reproduce the geodesic distances as well as possible: with $\mathbf{H} = \mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^\top$, the matrix $\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}_g^{(2)}\mathbf{H}$ is decomposed, and $\mathbf{Y} = \mathbf{Q}_q\mathbf{\Lambda}_q^{1/2}$ is formed from its q largest eigenpairs. The discarded share of the eigenvalue mass is the residual variance, and its decrease with q indicates the intrinsic dimension of the manifold. Fuzzy c-means is run on $\mathbf{Y}$, and each center is "re-projected" into the feature space as the sample whose projection is closest to it, so that the prototypes lie on the manifold here too; the cost (7.9) evaluated with geodesic distances to the re-projected centers compares the two routes. The weakness is the projection: Isomap can only embed a connected graph (hence the bridges), and a projection

In a nutshell **Geodesic fuzzy c-medoids**

- The c-means functional with the geodesic (shortest-path) distance of a k-nearest-neighbor graph and medoids as prototypes: clusters follow the manifold, and memberships fade along it, not across it.
- One structural parameter, k: large enough to keep each manifold connected, small enough not to connect different ones; the component count and the plateau of the result over k show the usable range.
- Disconnected components are bridged with a penalty (kept apart, all distances finite) or, when they are fragments of one manifold, with plain bridges; m plays its usual role.
- $O(N^2)$ memory and $O(cN^2)$ time per iteration, which suits small and medium data sets; new samples enter through their nearest training sample.

Algorithm 7.3 **Clustering of the Isomap projection (Algorithm I)**

1. Neighborhood graph, bridges and geodesic distances $\mathbf{D}_g$ as in Algorithm II.
2. Classical MDS of $\mathbf{D}_g$: $\mathbf{B} = -\frac{1}{2}\mathbf{H}\mathbf{D}_g^{(2)}\mathbf{H}$, $\mathbf{Y} = \mathbf{Q}_q\mathbf{\Lambda}_q^{1/2}$; residual variance from the discarded eigenvalues.
3. Fuzzy c-means on $\mathbf{Y}$ with c clusters and exponent m (random restarts allowed).
4. Re-project every center $\mathbf{z}_i$ as the sample whose $\mathbf{y}_k$ is closest to it; report $\mathbf{U}$, the projected and re-projected centers and J with geodesic distances.

that folds the manifold spoils the clustering. The paper found Algorithm II slightly better on the S-curve and the spiral; in `fuzzyclust` the two reach nearly the same geodesic cost (fig. 7.18).

In a nutshell **Clustering of the Isomap projection**

- Isomap unfolds the manifold, FCM clusters the flat picture, and the centers return to the data space as the closest samples.
- Needs a connected (or bridged) graph and a good projection; the residual variance tells how much of the geodesic geometry q dimensions keep.
- The graph parameters of Algorithm II (components are always bridged) plus the embedding dimension q; the FCM step inherits `init` and `n_init`.

7.3 Using it in fuzzyclust

Everything in this chapter is in `fuzzyclust.graph`. The three estimators derive from `BaseFuzzyClusterer` and have the usual methods `fit`, `predict`, `predict_proba`, `transform` and `score`. The graph utilities are functions built on the sparse-graph routines of SciPy, so no optional dependency is needed; `networkx` is accepted only as an alternative backend for the tree.

`fuzzyclust.graph.MSTClustering`

`n_clusters=None`, `threshold=None`, `inconsistency=2.5`, `inconsistency_std=None`, `n_hops=2`, `criterion="hypervolume"|"fukuyama_sugeno"`, `min_gain=0.15`, `min_cluster_size=5`, `max_clusters=20`, `refine="gg"|"fcm"|None`, `m=2.0`, `gamma=0.0`, `tol`, `max_iter`, `backend="scipy"|"networkx"`, `random_state`, `verbose`.
Attributes `n_clusters_`, `tree_labels_` (crisp tree partition), `labels_`, `membership_`, `cluster_centers_`, `covariances_`, `mst_edges_`, `mst_weights_`, `cut_edges_`, `cut_phases_` (0: length criteria, 1: volume), `cut_quality_` (CP), `hypervolumes_`, `cluster_similarity_` (S_{ij}), `refiner_` (the fitted GG or FCM). `min_gain=1.0` disables the splitting phase (unless `n_clusters` is given); `threshold=None` and `inconsistency=None`, with the default `inconsistency_std=None`, disable the length criteria.

`fuzzyclust.graph.GeodesicFCM`

`n_clusters=3`, `n_neighbors=10`, `radius=None`, `connect_components=True`, `bridge_penalty=1.0`, `m=2.0`, `tol`, `max_iter`, `init="kmeans++"|"random"|index array`, `n_init=1`, `random_state`, `verbose`.
Attributes `medoid_indices_`, `cluster_centers_` (the medoids), `membership_`, `labels_`, `geodesic_distances_`, `neighbors_graph_`, `n_components_`, `component_labels_`, `bridge_edges_`, `objective_`, `objective_history_`, `n_iter_`. New samples enter the graph through their nearest training sample.

`fuzzyclust.graph.IsomapFCM`

`n_clusters=3`, `n_components=2`, `n_neighbors=10`, `radius=None`, `bridge_penalty=1.0`, `m`, `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`.
Attributes `embedding_` (Y), `embedded_centers_`, `medoid_indices_`, `cluster_centers_` (re-projected), `membership_`, `labels_`, `geodesic_`, `objective_`, `residual_variance_`, `fcm_` and the graph attributes of `GeodesicFCM`.

graph utilities, data generators and plot helpers

`minimum_spanning_tree(X, backend)` → edges, lengths; `neighbors_graph(X, n_neighbors, radius)` → sparse adjacency; `geodesic_distances(X, n_neighbors, radius, connect_components, bridge_penalty)` → `Bunch(distances, graph, n_components, component_labels, bridges)`; `isomap_embedding(D, n_components)` → coordinates, eigenvalues; `fuzzy_cluster_similarity(U)` → S_{ij}. Generators `make_chained_blobs`, `make_density_blobs`, `make_s_curve` (with `make_rings`, `make_two_moons`, `make_curved_manifold` of `fuzzyclust.datasets`). Plots `plot_mst(X, model)`, `plot_neighbors_graph(X, graph_or_model)`, `plot_geodesic_path(X, graph, source, target)`, `plot_similarity_dendrogram(S)`.

The classical tree clusterer and the hybrid one differ in two parameters:

```
tree_only = MSTClustering(threshold=None, inconsistency=None,
    min_gain=1.0, refine=None).fit(X)
zahn = MSTClustering(inconsistency=2.5, n_hops=2, min_gain=1.0,
    refine=None).fit(X)
print("tree only  :", tree_only.n_clusters_, "cluster")
print("Zahn (f=2.5):", zahn.n_clusters_, "clusters, cut edges:",
    len(zahn.cut_edges_))
```

```
mst = MSTClustering(random_state=0).fit(X)
print("clusters found:", mst.n_clusters_)
print("cut phases    :", mst.cut_phases_, " (0 = length criteria, 1 =
    volume criterion)")
print("hypervolumes  :", mst.hypervolumes_.round(3))
```

The refinement and its adequacy measures are part of the fit, and the number of clusters can be prescribed and scanned with the validity functions of the toolbox:

```
gg_random = GG(n_clusters=mst.n_clusters_, random_state=0).fit(X)
ari_tree = adjusted_rand_score(y[blob], mst.tree_labels_[blob])
ari_gg = adjusted_rand_score(y[blob], mst.labels_[blob])
ari_random = adjusted_rand_score(y[blob], gg_random.labels_[blob])
print(f"ARI of the blobs: tree partition {ari_tree:.2f}, MST-GG
    {ari_gg:.2f}, GG from random start {ari_random:.2f}")
print("cut adequacy CP(e):", mst.cut_quality_.round(2))
```

```
cs = range(2, 9)
rows = []
for c in cs:
    model = MSTClustering(n_clusters=c, random_state=0).fit(X)
    U, V = model.membership_, model.cluster_centers_
    rows.append((
        validity.fuzzy_hypervolume(X, U, V),
        validity.partition_density(X, U, V),
        validity.average_partition_density(X, U, V),
```

```
        adjusted_rand_score(y, model.labels_),
    ))
rows = np.array(rows)
auto = MSTClustering(random_state=0).fit(X)
print("volume criterion alone picks", auto.n_clusters_, "clusters")
```

The geodesic clusterer is a drop-in replacement for FCM, and the geodesic distances and paths are available on their own:

```
k = 8
results = {}
for name, (X, y) in datasets.items():
    fcm = FCM(n_clusters=2, random_state=0).fit(X)
    geo = GeodesicFCM(n_clusters=2, n_neighbors=k, random_state=0).fit(X)
    results[name] = (fcm, geo)
    print(f"{name:16s}: FCM ARI = {adjusted_rand_score(y,
    fcm.labels_):.2f}, "
          f"geodesic c-medoids ARI = {adjusted_rand_score(y,
    geo.labels_):.2f} "
          f"({geo.n_components_} graph component(s), {geo.n_iter_}
    iterations)")
```

```
geo = geodesic_distances(X, n_neighbors=8)
D, G = geo.distances, geo.graph
# two samples of the first arm, far apart along the arm
arm = np.flatnonzero(y == 0)
radius = np.linalg.norm(X[arm], axis=1)
source, target = arm[radius.argmin()], arm[radius.argmax()]
print(f"Euclidean distance {np.linalg.norm(X[source] - X[target]):.2f},
    geodesic distance {D[source, target]:.2f}")
```

```
fcm = FCM(n_clusters=8, random_state=0).fit(Xs)
geo = GeodesicFCM(n_clusters=8, n_neighbors=10, random_state=0).fit(Xs)
iso = IsomapFCM(n_clusters=8, n_components=2, n_neighbors=10,
    random_state=0).fit(Xs)
# geodesic cost of the FCM partition, with the samples closest to its
    centers as prototypes
Dg = geo.geodesic_distances_
fcm_proxies = np.linalg.norm(Xs[:, None] - fcm.cluster_centers_[None],
    axis=2).argmin(axis=0)
J_fcm = float(np.sum(fcm.membership_**2 * Dg[:, fcm_proxies] ** 2))
print(f"geodesic cost J: FCM {J_fcm:.0f}, Isomap + FCM
    {iso.geodesic_objective_:.0f}, "
      f"geodesic c-medoids {geo.objective_:.0f}")
print(f"Isomap residual variance with 2 components:
    {iso.residual_variance_:.3f}")
```

7.4 Examples

7.4.1 Cutting, splitting and refining the tree

Two Gaussian groups of 120 points are joined by a chain of 25 points (fig. 7.3). Without any criterion the tree is one cluster. Zahn's local criterion with $f = 2.5$ and $k = 2$ flags one edge, inside the chain close to the left group, so most of the chain stays attached to the right group: the length criteria cannot separate the groups from the chain. The volume criterion (fig. 7.4) makes two further cuts, one where the chain meets the right group and one in the middle of the chain. The result is four tree clusters, the two groups with hypervolumes 0.29 and 0.25 (the left one with the six chain points left of the length cut) and two chain pieces with 0.02 and 0.05, and the points of the two groups are separated exactly. Started from this partition, Gath–Geva keeps it (agreement 0.98 with the tree partition) and delivers the membership contours and covariance ellipses of fig. 7.5; the same GG started from a random partition with four clusters spends one Gaussian on the whole chain and collapses another onto a few points at the edge of the left group. The cut adequacies are 0 for the cut of the length criterion, whose endpoints the refinement re-united in the neighboring chain piece, and 1.5 and 2.0 for the two volume cuts. The parameter study (fig. 7.6) shows the number of tree clusters falling as the required relative gain rises from 0.05 to 0.6: from twenty to six with `min_cluster_size=2`, from five to two with the default size of five, and from three to two with sizes of ten and twenty, which remove the small fragments altogether. The printed comparison of the refinements shows FCM pulled off the tree partition (agreement 0.63) while GG stays on it.

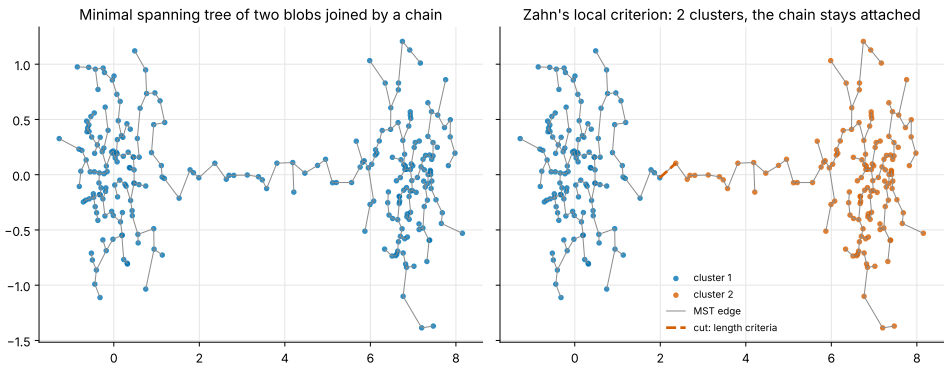


Figure 7.3. The minimal spanning tree of two groups joined by a chain (left) and the single edge that Zahn's local criterion removes (right, dashed): the chain stays attached to one group.

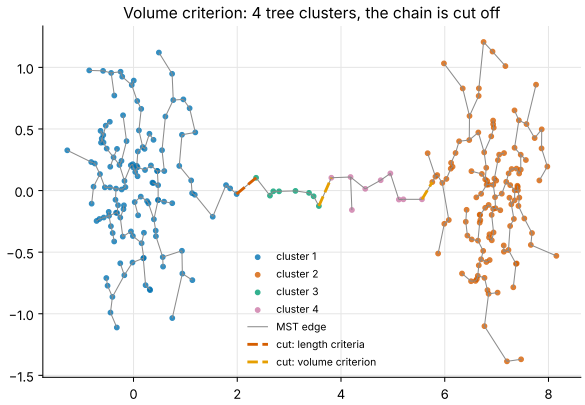


Figure 7.4. The volume criterion cuts the chain (orange dashed edges) and separates it from the groups; the length-criterion cut is drawn in vermilion.

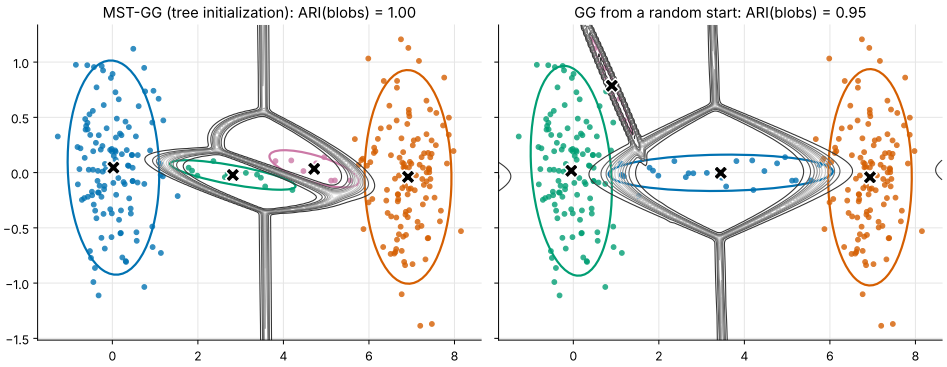


Figure 7.5. Gath–Geva clustering initialized with the tree partition (left) polishes it into a fuzzy partition with covariance ellipses; the same GG from a random start (right) spends one Gaussian on the chain and collapses another onto a few points of the left group.

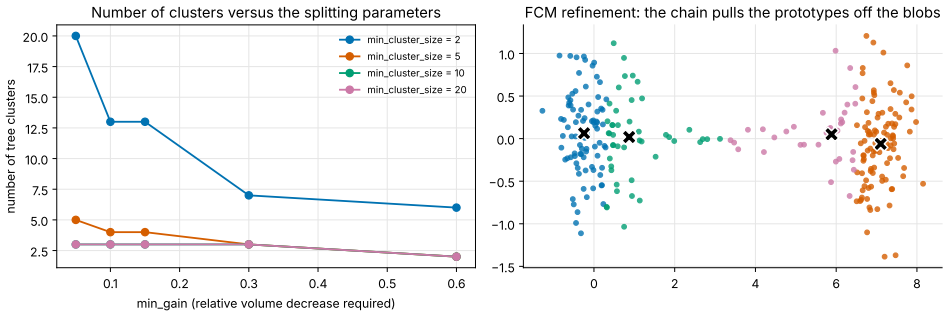


Figure 7.6. Left: the number of tree clusters against the required relative volume gain for several minimum cluster sizes. Right: FCM as refinement is pulled off the tree partition by the chain points.

7.4.2 Which edges are inconsistent?

The data are a diffuse group of 200 points next to two tight groups of 40 points (fig. 7.7). A global threshold of twice the mean edge length cuts eight clusters out of the tree, mostly tails of the diffuse group; four times the mean leaves the two tight groups merged, and the printed scan shows that no threshold in between returns the three groups: the edge between the tight groups is no longer than the longest edges inside the diffuse group (fig. 7.8). The local criterion with $f = 2.5$ and $k = 3$ finds the three groups exactly, because each separating edge is far longer than its own neighbors. Figure 7.9 varies the factor and the depth: with $k = 1$ no factor returns the three groups ($f \leq 3$ fragments the diffuse group, $f = 4$ already misses a separation), with $k = 3$ or 5 every f between 2.5 and 4 works (with $k = 8$ from $f = 3$ on), and $f = 6$ misses a separation for every depth; the deviation form needs $k \geq 5$ at three standard deviations and $k \geq 2$ at four. Raising `min_cluster_size` tames an eager criterion ($f = 2$, $k = 2$: from ten clusters at size 5 to four at size 20).

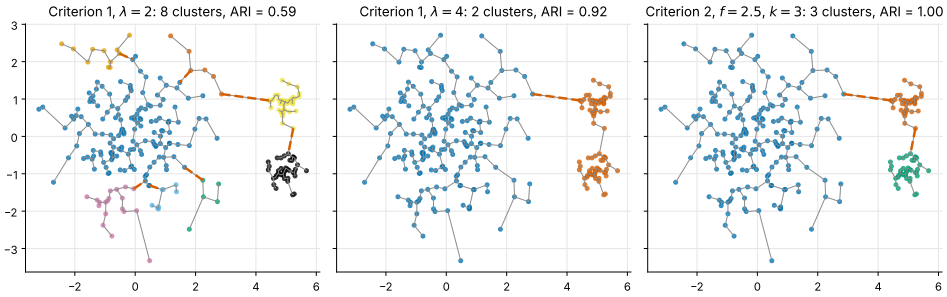


Figure 7.7. Length criteria on a diffuse group and two tight groups. A global threshold of twice the mean edge length shatters the diffuse group (left), four times the mean merges the tight groups (middle), and Zahn's local criterion separates the three groups (right).

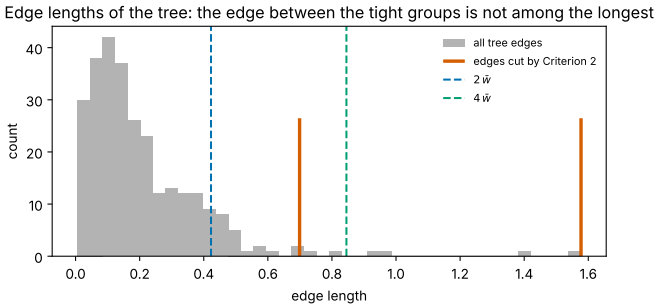


Figure 7.8. Edge lengths of the tree with the two global thresholds (dashed) and the two edges removed by the local criterion (vermilion): the edge between the tight groups lies among the long edges of the diffuse group.

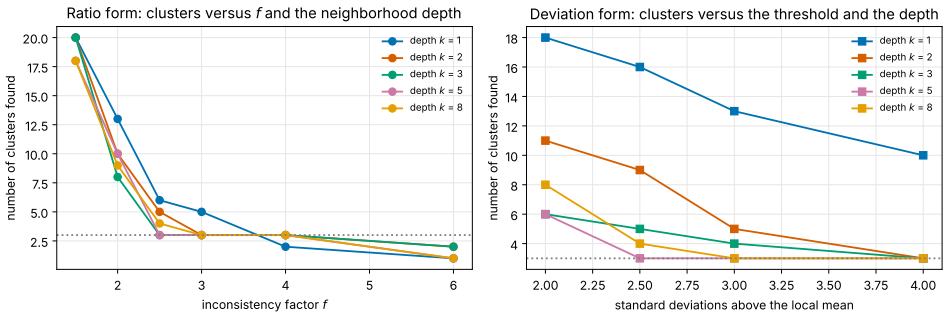


Figure 7.9. Number of clusters found by the ratio form (left) and the deviation form (right) of Criterion 2 as a function of the factor and the neighborhood depth; the dotted line is the true number.

7.4.3 How many clusters?

Four Gaussian groups of different sizes are clustered with `n_clusters` from 2 to 8, and the volume indices of Gath and Geva are computed from the refined partitions (fig. 7.10). The fuzzy hypervolume falls by 38 % from three to four clusters and by less than 3 % for every further split; the partition density rises steeply up to four and then stays flat; the average partition density peaks at four but is overtaken at eight, where small dense fragments appear. The knee, not the extremum, carries the information, and the relative decrease of FHV drops below the default `min_gain` of 0.15 exactly where the automatic run stops, at four. Two concentric rings (fig. 7.11) show the limits of the refinement: the tree partition with two clusters separates the rings perfectly, but a Gath–Geva refinement of two clusters fails (ARI 0.09), because one Gaussian cannot describe a hollow ring and the inner ring is absorbed by the broad outer Gaussian. Without a prescribed number, the volume criterion keeps splitting the rings into arcs that are approximately Gaussian until `max_clusters` is reached; with `max_clusters=12` it produces twelve arcs, and the refinement of the arcs is faithful. Their fuzzy similarity matrix (fig. 7.12) has its large entries between neighboring arcs of the same ring, the single-linkage dendrogram of $1 - S_{ij}$ has two main branches, and cutting it into two groups merges the arcs back into the rings (ARI 1.0). The similarities grow with the exponent m of the refinement (the largest same-ring entry rises from 0.007 at $m = 1.5$ to 0.05 at $m = 4$) while the merge stays correct, and the number of arcs follows `max_clusters`.

7.4.4 Geodesic c-medoids versus fuzzy c-means

The data sets are two spiral arms, two moons and two rings, each with two clusters (fig. 7.13). FCM reaches adjusted Rand indices of only 0.02, 0.24 and 0.00, because its centers cut straight across the manifolds, while the geodesic c-medoids with $k = 8$ recover all three groupings exactly in three to five iterations. The spiral forms one graph component in which the path between the arms runs through

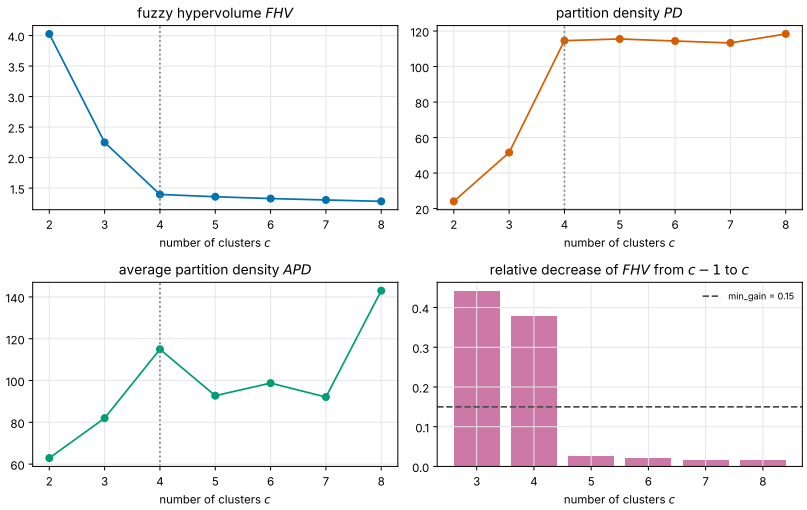


Figure 7.10. Volume-based validity indices of the MST–GG partitions of four groups against the number of clusters: fuzzy hypervolume, partition density, average partition density and the relative decrease of the hypervolume with the default minimum gain (dashed); the dotted line is the number the automatic run picks.

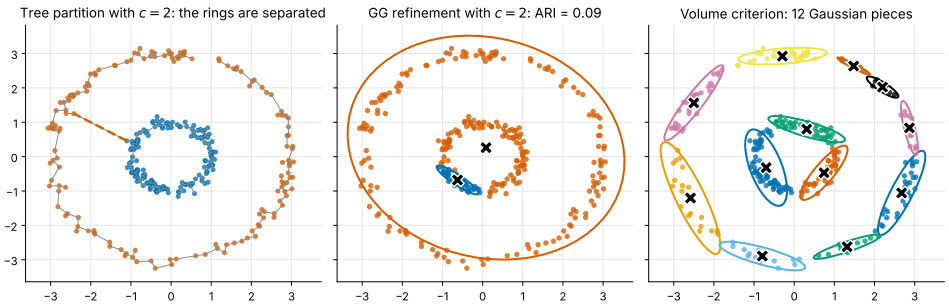


Figure 7.11. Two rings: the tree partition with two clusters (left), the failed Gath–Geva refinement of two clusters (middle) and the twelve Gaussian arcs the volume criterion produces (right).

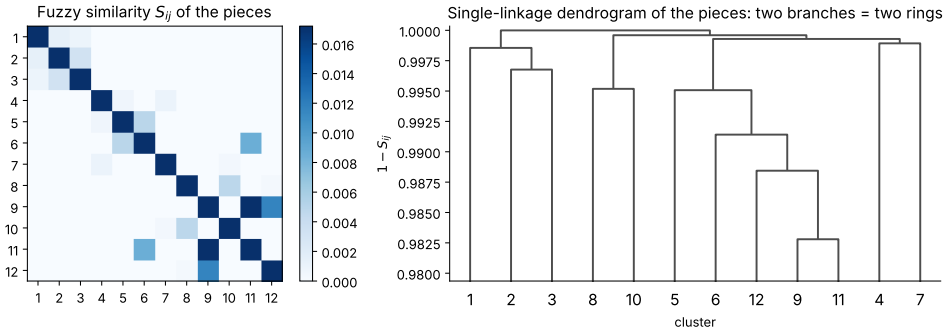


Figure 7.12. Fuzzy similarity of the twelve arcs (left) and its single-linkage dendrogram (right): the two branches are the two rings.

the center; the moons and the rings fall into two components that are bridged with a penalty. The memberships (fig. 7.14) fade along the arm from the medoid toward its ends, while a point of the other arm that is close in the Euclidean sense keeps a membership near zero. The scan over k (fig. 7.15) shows the two failure modes. A small k fragments the sparsely sampled outer parts of the arms into components that have to be bridged, and the result degrades; a large k connects the arms where they come close, and the geodesic distance loses its meaning. On the spiral the plateau spans $k = 8$ to 15, on the moons 5 to 8. Moons whose gap is narrower than the sample spacing form one component already at $k = 3$ (fig. 7.15, right) and cannot be separated for any k (fig. 7.16): the graph cannot resolve a gap the sampling does not resolve. The printed study shows the usual effect of m on the softness of the partition. It also shows that on the rings a plain Euclidean bridge (`bridge_penalty=0`) lets the far side of the outer ring switch to the inner medoid through the bridge (ARI 0.28), whereas both the penalized bridge and infinite distances (`connect_components=False`) keep the rings apart.

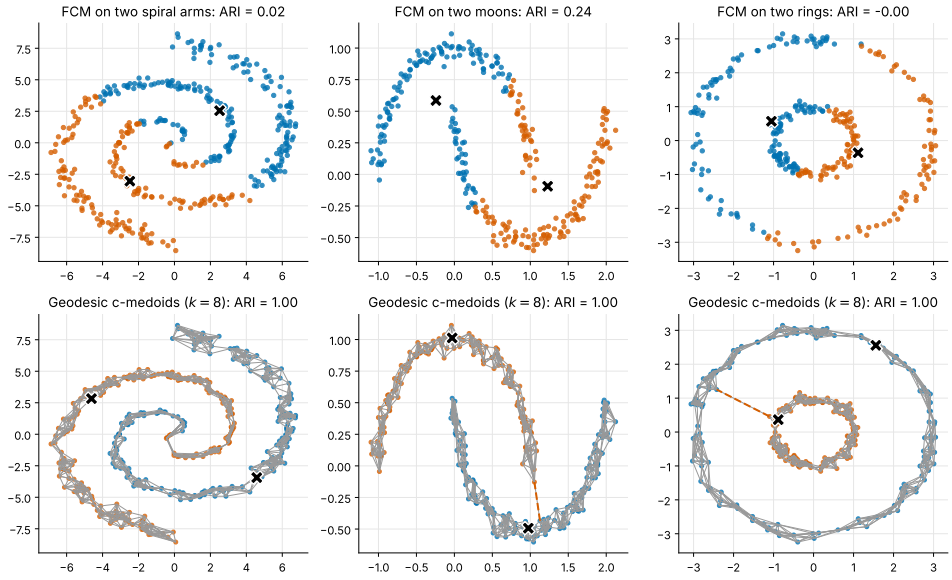


Figure 7.13. Fuzzy c-means (top) and geodesic c-medoids with $k = 8$ (bottom, with the neighborhood graph and the bridge of the two-component cases) on two spiral arms, two moons and two rings: only the geodesic clusters follow the manifolds.

7.4.5 Geodesic paths, the S-curve and the Isomap route

Figure 7.17 traces shortest paths through the $k = 8$ graph of the spiral: between the inner and the outer end of one arm the geodesic distance is 32 against a chord of 8.4, and the closest outer pair of samples of different arms, 1.7 apart in the plane, is 38 apart along the graph. On the three-dimensional S-curve with eight clusters (fig. 7.18) the FCM centers float between the layers of the S (up to 0.19

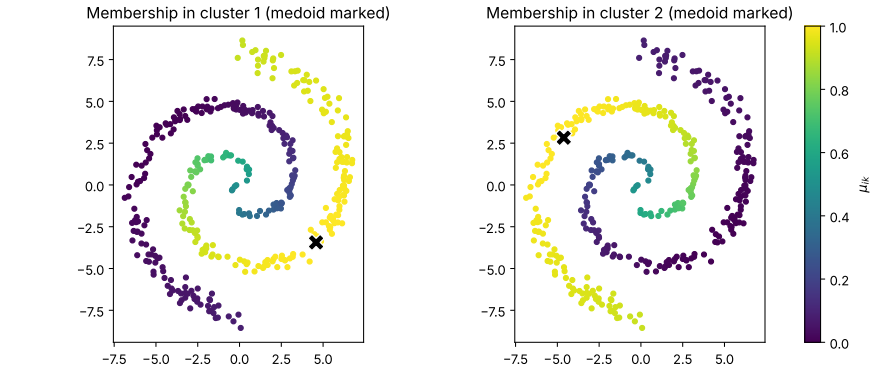


Figure 7.14. Memberships of the spiral samples in the two geodesic clusters: they fade along the arm, not across it.

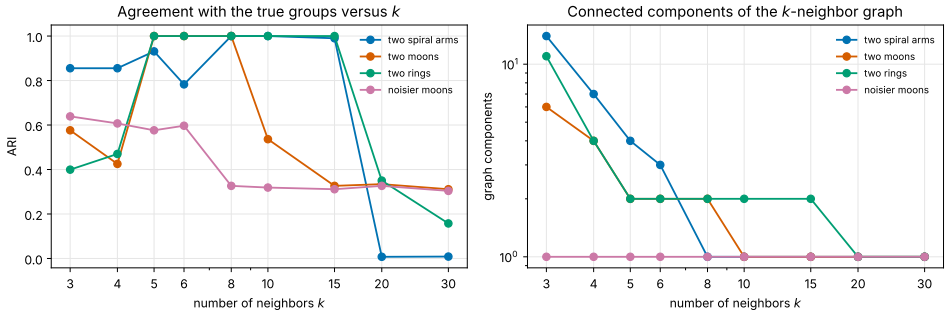


Figure 7.15. Agreement with the true groups (left) and number of graph components (right) against the number of neighbors k for the four data sets: good results need a k between fragmentation and short-circuits.

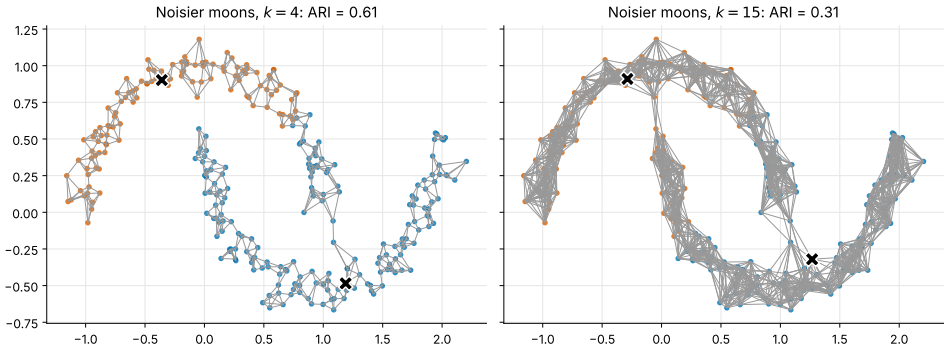


Figure 7.16. Noisier moons with $k = 4$ and $k = 15$: the gap is bridged by the graph itself, and the tips of the moons follow the other medoid.

from the nearest sample), while the medoids of Algorithm II and the re-projected centers of Algorithm I lie on the sheet. The geodesic cost of the partitions is 334 for FCM against 269 and 273 for the two geodesic routes, which thus reach nearly the same solution by different means. The Isomap projection unrolls the S into a rectangle that FCM tiles regularly, and the geodesic clusters cover the sheet in the same way (fig. 7.19). The study of Algorithm I on the spiral (fig. 7.20) shows the role of the bridges when the graph is fragmented: with $k = 4$ the arms break into seven components. The penalized bridges keep every fragment apart in the projection, and two clusters cannot collect them (ARI 0.86 for both algorithms), whereas plain bridges re-join the fragments of an arm at their natural junctions, and both algorithms recover the arms. With $k = 10$ the graph is connected and the bridges play no role. On the S-curve, two and three Isomap dimensions give nearly the same geodesic cost (269 and 272), whereas a single dimension raises it to 665; the residual variance drops from 0.09 with one component to 0.04 with two.

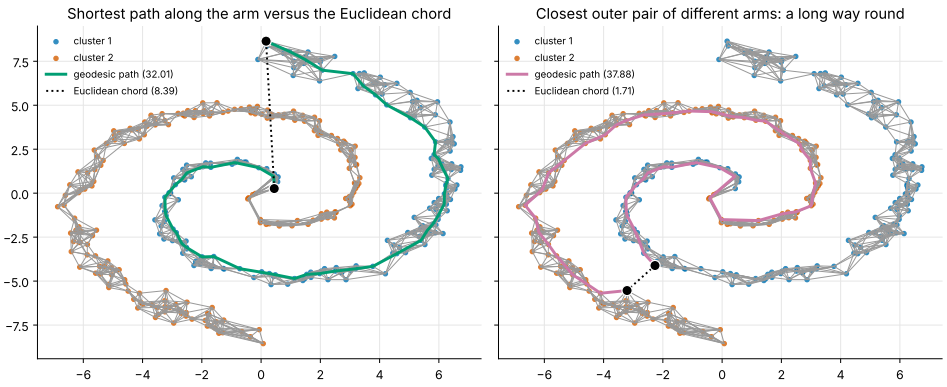


Figure 7.17. Shortest paths in the neighborhood graph of the spiral: along one arm from its inner to its outer end (left) and between the closest outer pair of samples of different arms (right); the dotted lines are the Euclidean chords.

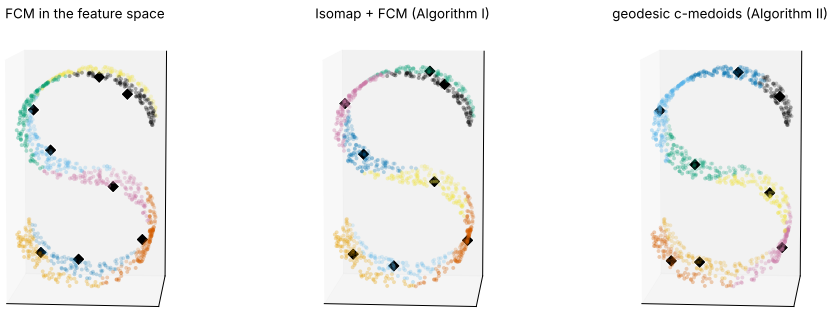


Figure 7.18. Eight clusters on the S-curve: FCM centers float between the layers (left), the re-projected centers of the Isomap route (middle) and the geodesic medoids (right) lie on the sheet.

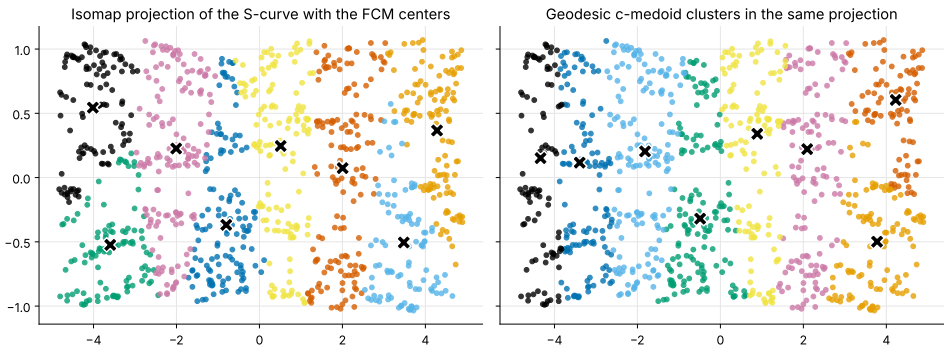


Figure 7.19. The Isomap projection of the S-curve with the FCM centers (left) and the geodesic c-medoid clusters in the same projection (right): both tile the unrolled sheet regularly.

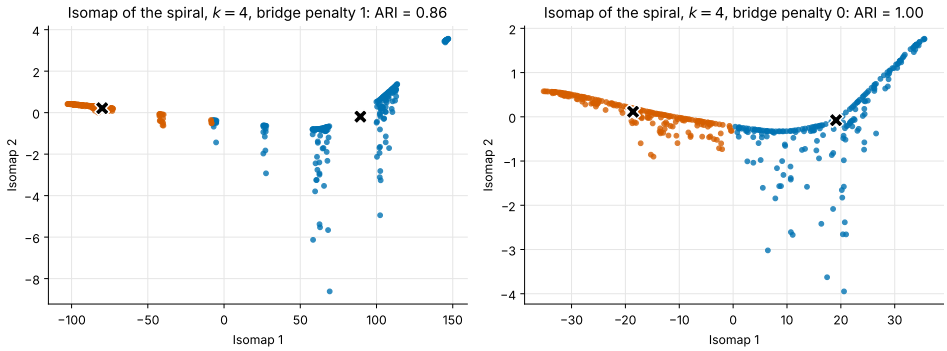


Figure 7.20. Isomap projection of the spiral with $k = 4$, where the arms are fragmented into seven components: penalized bridges keep the fragments apart (left), plain bridges re-join the fragments of each arm (right); colors are the true arms, crosses the FCM centers.

7.5 Further reading

The hybrid MST–GG algorithm was published in [83]; it builds on Zahn's graph-theoretical clustering [81], on the relation between the minimal spanning tree and single linkage [82], on the tree algorithms of Prim and Kruskal [87, 88], on the hypervolume and density indices of Gath and Geva [21], on the Fukuyama–Sugeno measure [33] as used by Varma and Simon [89], and on compatible cluster merging [40] for the idea of merging similar fuzzy clusters; the Gath–Geva refinement may use the covariance regularization of [19]. Graph-theoretic and hierarchical clustering in general are covered by [4]. The geodesic clustering algorithms are from [84], which rests on Isomap [85] and the convergence of graph distances to geodesics [86], on shortest paths [90, 91], on classical multidimensional scaling [92], on the c-means functional [3] and on the medoid methods of Kaufman and Rousseeuw [5]; locally linear embedding [93] is the other classical manifold method the paper mentions. The fuzzy Sammon mapping of chapter 5 [51] is the natural tool for inspecting the clusters found here when the data have more than two dimensions.

CHAPTER

8



Clustering for model identification

So far the clusters were the result. In this chapter they are a means: a fuzzy cluster in the joint space of inputs and outputs is a region of the data in which one simple model holds, so every cluster can be turned into an *if–then rule* with a local linear model as consequent. We start from the Takagi–Sugeno fuzzy model and show how product-space clustering with Gath–Geva or Gustafson–Kessel prototypes yields its antecedents and consequents, why the projection of oblique clusters onto the input axes costs accuracy, and how the modified Gath–Geva algorithm avoids this by clustering with a prototype that *is* a rule and that carries over to multi-output dynamic systems. Three further algorithms exploit the cluster-as-model view: fuzzy c-regression models (clusters are hyperplanes), supervised fuzzy clustering (clusters are rules with class probabilities, and their projections give fuzzy partitions for decision trees), and the hinging-hyperplane regression tree, which grows a binary tree by clustering with two regression prototypes at every node; all of them are in `fuzzyclust.models`.

8.1 Problem and idea

A nonlinear function $y = f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$, is rarely known in closed form, but it is often approximately linear in small regions. *Operating regime based modeling* builds a global model from local linear models $\hat{y}_i = \mathbf{a}_i^\mathsf{T}\mathbf{x} + b_i$, each valid in a region of the input space described by a validity function, and interpolates between them. When the regions are fuzzy sets, the model is the Takagi–Sugeno (TS) fuzzy model [94]

$$\begin{aligned} R_i &: \text{If } x_1 \text{ is } A_{i1}(x_1) \text{ and } \ldots \text{ and } x_n \text{ is } A_{in}(x_n) \\ &\quad \text{then } \hat{y}_i = \mathbf{a}_i^\mathsf{T}\mathbf{x} + b_i, \quad [w_i], \qquad i = 1, \ldots, c, \end{aligned} \tag{8.1}$$

whose rules fire with the strength $\beta_i(\mathbf{x}) = w_i \prod_{j=1}^{n} A_{ij}(x_j)$ and whose output is the fuzzy mean

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{c} \beta_i(\mathbf{x})\,(\mathbf{a}_i^{\mathsf{T}}\mathbf{x} + b_i)}{\sum_{i=1}^{c} \beta_i(\mathbf{x})}. \tag{8.2}$$

The model output is linear in the consequent parameters $\boldsymbol{\theta}_i = [\mathbf{a}_i^{\mathsf{T}}\ b_i]^{\mathsf{T}}$, which are therefore estimated by least squares once the antecedents are known. The bottleneck is the antecedent part, that is, the location and the width of the operating regions. Gradient-based neuro-fuzzy tuning and evolutionary search find them at a high computational price and tend to destroy the local meaning of the rules.

Fuzzy clustering in the *product space* of inputs and outputs offers a cheaper and more transparent solution [7, 95]. Collect every sample as $\mathbf{z}_k = [\mathbf{x}_k^{\mathsf{T}}\ y_k]^{\mathsf{T}} \in \mathbb{R}^{n+1}$ and cluster the $\mathbf{z}_k$. Where the function is approximately linear, the samples lie close to a hyperplane; a cluster that adapts its shape (Gustafson–Kessel, Gath–Geva) becomes a flat ellipsoid stretched along that hyperplane. The hyperplane is the local linear model, and the extent of the cluster along the inputs is its operating region. Figure 8.1 shows the idea: three clusters along a curve, each projected onto the input axis to give a membership function $A_i(x)$ and each carrying a line $a_i x + b_i$. The fuzzy partition matrix of the clustering then serves two purposes: it defines the antecedents and weights the least-squares problems of the consequents. The fuzziness also regularizes the local models, because neighboring rules share data.

Two difficulties remain, and they motivate the algorithms of this chapter. First, a Gaussian cluster found by Gath–Geva clustering is an *oblique* ellipsoid in $\mathbb{R}^{n+1}$ (the input block $\mathbf{F}_i^{xx}$ of its covariance matrix has off-diagonal elements), whereas rule (8.1) needs univariate sets $A_{ij}(x_j)$; projecting the ellipsoid onto the axes loses its orientation and introduces a *decomposition error*, which can be compensated only by re-estimating the consequents globally, at the price of their local meaning. The exact alternative, a linear transformation of the inputs (eigenvector projection), yields rules on transformed variables that are hard to interpret. The modified Gath–Geva algorithm of Abonyi, Babuška and Szeifert [96] resolves this by changing the prototype: a cluster is an *axis-parallel* Gaussian in the inputs together with a local linear model and the variance of its error, so the clusters are rules by construction. Second, point-like prototypes cannot cope with data in which the local laws cross each other; for such *switching regression* problems the prototype must be the hyperplane itself, which is the fuzzy c-regression model of Hathaway and Bezdek [97]. Both ideas, a prototype that includes a model and a distance that includes the class label or the output, recur in the supervised clustering of classifiers [98], in the supervised partitions for decision trees [99] and in the hinging-hyperplane trees [100].

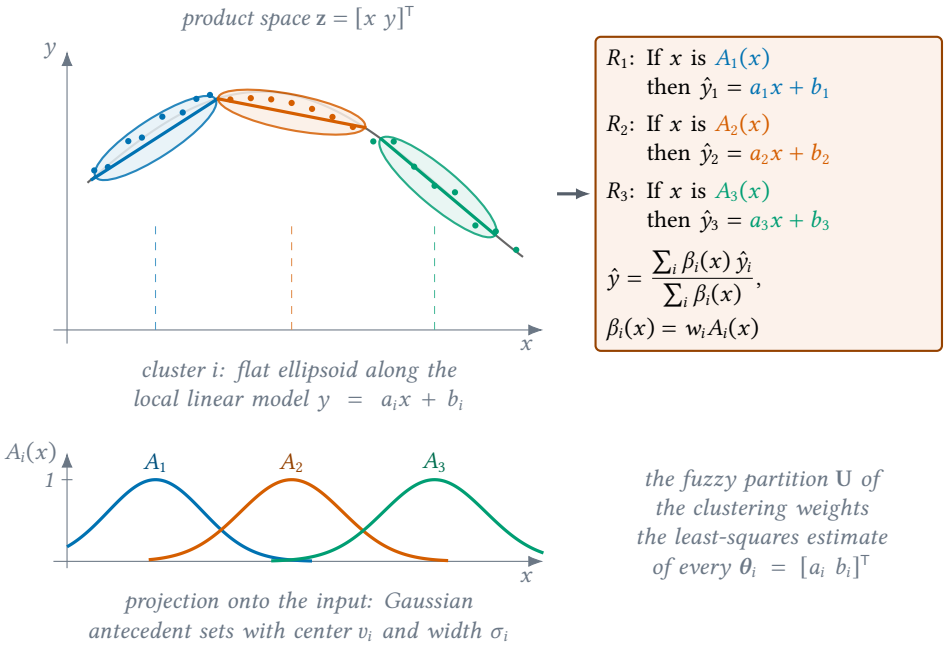


Figure 8.1. Product-space clustering for Takagi–Sugeno model identification. Top left: the samples $\mathbf{z}_k = [x_k\ y_k]^\mathsf{T}$ of an unknown function are clustered; where the function is nearly linear, the clusters are flat ellipsoids whose main axis is the local linear model. Bottom left: the projection of each cluster onto the input gives a Gaussian antecedent set. Right: cluster i becomes rule R_i, and the memberships weight the least-squares estimation of the consequents.

8.2 Theory

Throughout the chapter $\mathbf{X} \in \mathbb{R}^{N\times n}$ collects the inputs (regressors) $\mathbf{x}_k$, $\mathbf{y} \in \mathbb{R}^N$ the outputs y_k (or $\mathbf{Y} \in \mathbb{R}^{N\times n_y}$ for several outputs), $\mathbf{z}_k = [\mathbf{x}_k^\mathsf{T}\ y_k]^\mathsf{T}$ the joint samples, $\mathbf{U} = [\mu_{ik}] \in \mathbb{R}^{c\times N}$ the fuzzy partition (stored transposed in the code, as everywhere in `fuzzyclust`), $\mathbf{X}_e = [\mathbf{X}\ \mathbf{1}]$ the regressor matrix extended by a column of ones and $\boldsymbol{\theta}_i = [\mathbf{a}_i^\mathsf{T}\ b_i]^\mathsf{T}$ the parameters of the local linear models. We first recall how TS models follow from Gath–Geva clusters and derive the modified algorithm (sections 8.2.1 and 8.2.2), extend it to multi-output dynamic systems (section 8.2.3), and then treat fuzzy c-regression (section 8.2.4), supervised clustering for classifiers and partitions (sections 8.2.5 and 8.2.6) and the hinging-hyperplane tree (section 8.2.7).

8.2.1 From Gath–Geva clusters to Takagi–Sugeno rules

Gath–Geva clustering (section 3.2.5) has a probabilistic interpretation [21, 96]. Let $p(\eta_i)$ be the prior probability of cluster i and $p(\mathbf{z} \mid \eta_i) = \mathcal{N}(\mathbf{v}_i, \mathbf{F}_i)$ its domain of influence; the algorithm identifies the mixture $p(\mathbf{z}) = \sum_i p(\mathbf{z} \mid \eta_i)p(\eta_i)$, with

$\alpha_i = p(\eta_i)$ and the distance $1/D_{ik}^2 = p(\mathbf{z}_k \mid \eta_i)p(\eta_i)$. Partition the prototype and the covariance into input and output blocks,

$$\mathbf{v}_i = \begin{bmatrix} \mathbf{v}_i^x \\ v_i^y \end{bmatrix}, \qquad \mathbf{F}_i = \begin{bmatrix} \mathbf{F}_i^{xx} & \mathbf{F}_i^{xy} \\ \mathbf{F}_i^{yx} & F_i^{yy} \end{bmatrix}. \tag{8.3}$$

The regression function is the conditional expectation $E[y \mid \mathbf{x}]$ of the mixture, and because every component is Gaussian it is a weighted sum of linear functions,

$$\begin{aligned} E[y \mid \mathbf{x}] &= \sum_{i=1}^{c} p(\eta_i \mid \mathbf{x})\, [\mathbf{x}^\mathsf{T}\ 1]\, \boldsymbol{\theta}_i, \qquad p(\eta_i \mid \mathbf{x}) = \frac{g_i(\mathbf{x})}{\sum_{j=1}^{c} g_j(\mathbf{x})}, \\ g_i(\mathbf{x}) &= \frac{p(\eta_i) \exp\!\big(-\tfrac{1}{2}(\mathbf{x}-\mathbf{v}_i^x)^\mathsf{T}(\mathbf{F}_i^{xx})^{-1}(\mathbf{x}-\mathbf{v}_i^x)\big)}{(2\pi)^{n/2}\sqrt{\det \mathbf{F}_i^{xx}}}. \end{aligned} \tag{8.4}$$

This is exactly the TS model (8.2) with multivariate Gaussian antecedents $A_i(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x}-\mathbf{v}_i^x)^\mathsf{T}(\mathbf{F}_i^{xx})^{-1}(\mathbf{x}-\mathbf{v}_i^x))$ and rule weights

$$w_i = \frac{p(\eta_i)}{(2\pi)^{n/2}\sqrt{\det \mathbf{F}_i^{xx}}}, \tag{8.5}$$

the *Gaussian mixture of regressors* [101]. The consequents can be estimated in two ways [96, 102]:

- *Local weighted least squares*: one problem per rule with the memberships as weights, $\boldsymbol{\Phi}_i = \operatorname{diag}(\mu_{i1}, \ldots, \mu_{iN})$,

$$\boldsymbol{\theta}_i = \left(\mathbf{X}_e^\mathsf{T}\boldsymbol{\Phi}_i\mathbf{X}_e\right)^{-1}\mathbf{X}_e^\mathsf{T}\boldsymbol{\Phi}_i\mathbf{y}, \tag{8.6}$$

  which can also be read off the cluster, $\mathbf{a}_i = (\mathbf{F}_i^{xx})^{-1}\mathbf{F}_i^{xy}$, $b_i = v_i^y - \mathbf{a}_i^\mathsf{T}\mathbf{v}_i^x$, when the center and the covariance of the cluster are computed with the same weights μ_{ik} (with the μ_{ik}^m-weighted $\mathbf{F}_i$ of Gath–Geva clustering the two estimates differ slightly). The local models are local linearizations of the function; they can be read and used one by one (the model is a lookup table of linearizations), but the overall prediction error is not minimal.
- *Global least squares*: all consequents in one problem with the normalized firing strengths as regressor weights, which minimizes the prediction error of (8.2) but lets the local models compensate each other where rules overlap, so that they lose their meaning.

A third estimate, *total least squares*, takes the hyperplane the cluster lies on: with $\mathbf{t}_i = [(\mathbf{t}_i^x)^\mathsf{T}\ t_i^y]^\mathsf{T}$ the eigenvector of the smallest eigenvalue of $\mathbf{F}_i$, $\mathbf{a}_i = -\mathbf{t}_i^x/t_i^y$ and $b_i = \mathbf{t}_i^\mathsf{T}\mathbf{v}_i/t_i^y$. It is appropriate when the inputs are noisy, but it does not minimize the prediction error, and it fails when the inputs are locally collinear, because the eigenvector of the smallest eigenvalue then reflects the dependency among the inputs rather than the regression.

The antecedents of (8.4) are multivariate. For the conjunctive rule (8.1) they have to be decomposed into univariate sets, and this is where the difficulty lies

(fig. 8.2). The product of the univariate Gaussians $A_{ij}(x_j) = \exp(-\frac{1}{2}(x_j - v_{ij})^2/\sigma_{ij}^2)$ equals $A_i(\mathbf{x})$ only if $\mathbf{F}_i^{xx}$ is diagonal. An oblique cluster can be projected onto the axes ($\sigma_{ij}^2 = [\mathbf{F}_i^{xx}]_{jj}$), which widens its footprint and loses the orientation (the *decomposition error*), or the inputs can be rotated into the eigenvector basis of $\mathbf{F}_i^{xx}$ (the transformed input-domain approach [103]), which is exact but yields rules on linear combinations of the inputs.

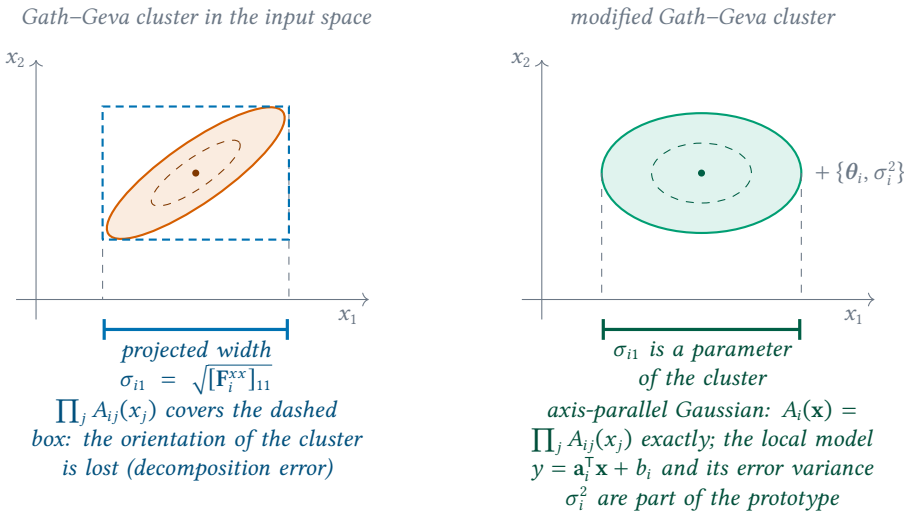


Figure 8.2. Projected Gath–Geva cluster and modified Gath–Geva prototype. Left: a Gath–Geva cluster is an oblique Gaussian in the input space; the univariate sets obtained by projection cover the dashed box and no longer describe the cluster. Right: the modified Gath–Geva prototype is an axis-parallel Gaussian whose widths σ_{ij} are cluster parameters, completed by a local linear model and the variance of its error, so the cluster is a conjunctive rule without any projection.

8.2.2 Modified Gath–Geva clustering

The modified algorithm [96] describes each cluster by an input distribution, a local model and an output distribution,

$$p(\mathbf{x}, y) = \sum_{i=1}^{c} p(y \mid \mathbf{x}, \eta_i)\, p(\mathbf{x} \mid \eta_i)\, p(\eta_i), \tag{8.7}$$

with the axis-parallel Gaussian input density and the Gaussian output error

$$p(\mathbf{x} \mid \eta_i) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} \exp\Big(-\frac{1}{2}\frac{(x_j - v_{ij})^2}{\sigma_{ij}^2} \Big), \tag{8.8}$$

$$p(y \mid \mathbf{x}, \eta_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\Big(-\frac{(y - \mathbf{x}_e^\mathsf{T}\boldsymbol{\theta}_i)^2}{2\sigma_i^2} \Big). \tag{8.9}$$

The parameters $\{\alpha_i, \mathbf{v}_i, \sigma_{ij}^2, \boldsymbol{\theta}_i, \sigma_i^2\}$ of cluster η_i are those of the TS rule (8.1) plus the two variances. The expectation-maximization (EM) algorithm for the mixture alternates the E-step, in which the posterior probability $p(\eta_i \mid \mathbf{x}_k, y_k)$ of every sample is computed by Bayes' rule, and the M-step, in which the parameters are re-estimated with these posteriors as weights: the priors $\alpha_i = \frac{1}{N}\sum_k p(\eta_i \mid \mathbf{x}_k, y_k)$, the centers and variances as weighted means, the local models by weighted least squares (8.6) with the posteriors on the diagonal of $\boldsymbol{\Phi}_i$, and the error variances as weighted mean squared residuals. Written as a fuzzy clustering algorithm with the memberships μ_{ik} in the role of the posteriors and the distance

$$\frac{1}{D_{ik}^2} = \alpha_i \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} \exp\Big(-\frac{1}{2}\frac{(x_{jk}-v_{ij})^2}{\sigma_{ij}^2}\Big) \cdot \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\Big(-\frac{(y_k - \mathbf{x}_{e,k}^{\mathsf{T}}\boldsymbol{\theta}_i)^2}{2\sigma_i^2}\Big), \tag{8.10}$$

the algorithm takes the form of Gath–Geva clustering with the functional $J = \sum_i \sum_k \mu_{ik}^m D_{ik}^2$. For $m = 2$ the membership update $\mu_{ik} = 1/\sum_j (D_{ik}/D_{jk})^{2/(m-1)}$ is the E-step, $\mu_{ik} = p(\eta_i \mid \mathbf{x}_k, y_k)$, and because the parameters are re-estimated with the weights μ_{ik} rather than μ_{ik}^m, as in the M-step above, the alternating optimization is the EM algorithm: it increases the likelihood of the mixture and does not, strictly, minimize J. The first factor of (8.10) measures how far the input is from the cluster center in units of the cluster widths, the second how well the local model predicts the output; a sample belongs to a cluster if its input lies in the cluster *and* the local model fits its output. The rule weights of the resulting TS model are $w_i = \alpha_i \prod_j (2\pi\sigma_{ij}^2)^{-1/2}$, and the firing strengths $\beta_i(\mathbf{x}) = w_i \prod_j A_{ij}(x_j)$, once normalized, are the posteriors $p(\eta_i \mid \mathbf{x})$ given the input alone, which is what prediction needs. `fuzzyclust` evaluates all densities in the log domain.

With the exponential distance, the algorithm converges fast to a nearby local optimum and can shrink a cluster onto a few samples (a tiny σ_{ij} makes $1/D^2$ huge). `fuzzyclust` therefore offers the variance regularization $\sigma_{ij}^2 \leftarrow (1-\gamma)\sigma_{ij}^2 + \gamma\,\mathrm{var}(x_j)$, the diagonal analog of the covariance regularization of section 3.2.4, a variance floor, and random restarts. Because J is not comparable between restarts of an exponential-distance algorithm, the restart is selected (`restart_criterion`) either by the mixture log-likelihood $\sum_k \log \sum_i 1/D_{ik}^2$, the EM criterion, or by the training error of the TS model. The two can disagree, because a rule with a small error variance on part of the data raises the likelihood without improving the global fit.

8.2.3 Multi-output dynamic systems

A nonlinear dynamic process is identified in the same framework by choosing lagged outputs and inputs as regressors. For a process with n_y outputs and n_u inputs, the regression vector of the nonlinear ARX (NARX) model is

$$\boldsymbol{\varphi}(k) = \big[\mathbf{y}(k-1)^{\mathsf{T}}, \ldots, \mathbf{y}(k-n_a)^{\mathsf{T}},\ \mathbf{u}(k-n_k)^{\mathsf{T}}, \ldots, \mathbf{u}(k-n_k-n_b+1)^{\mathsf{T}}\big]^{\mathsf{T}}, \tag{8.11}$$

Algorithm 8.1 Modified Gath–Geva clustering for TS model identification

Input: regressors $\mathbf{X}$, outputs $\mathbf{y}$, number of rules c, exponent m (usually 2), tolerance ϵ

initialize the partition $\mathbf{U}^{(0)}$ (random, k-means++ or from an FCM/GK run);

repeat

  for *every cluster* i **do**

    centers and widths: $v_{ij} = \sum_k \mu_{ik} x_{jk} / \sum_k \mu_{ik}$, $\sigma_{ij}^2 = \sum_k \mu_{ik}(x_{jk} - v_{ij})^2 / \sum_k \mu_{ik}$;

    local model: $\boldsymbol{\theta}_i = (\mathbf{X}_e^\mathsf{T} \boldsymbol{\Phi}_i \mathbf{X}_e)^{-1} \mathbf{X}_e^\mathsf{T} \boldsymbol{\Phi}_i \mathbf{y}$ with $\boldsymbol{\Phi}_i = \operatorname{diag}(\mu_{i1}, \dots, \mu_{iN})$;

    error variance: $\sigma_i^2 = \sum_k \mu_{ik}(y_k - \mathbf{x}_{e,k}^\mathsf{T} \boldsymbol{\theta}_i)^2 / \sum_k \mu_{ik}$;

    prior and rule weight: $\alpha_i = \frac{1}{N} \sum_k \mu_{ik}$, $w_i = \alpha_i \prod_j (2\pi\sigma_{ij}^2)^{-1/2}$;

  distances: D_{ik}^2 by (8.10) for all i, k;

  partition: $\mu_{ik} = 1 / \sum_{j=1}^{c} (D_{ik}/D_{jk})^{2/(m-1)}$;

until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: rules R_i: Gaussian sets (v_{ij}, σ_{ij}), local models $\boldsymbol{\theta}_i$, weights w_i; TS output by (8.2)

In a nutshell Modified Gath–Geva clustering

- A cluster is a TS rule: axis-parallel Gaussian antecedent sets, a local linear model, its error variance and a prior; the distance is the inverse joint density of input and output.
- For $m = 2$ the alternating optimization is the EM algorithm for a mixture of local linear regressions; the memberships are posterior probabilities.
- No projection, no decomposition error, rules on the original variables; the local models are local linearizations (weighted least squares), and a global re-estimation trades interpretability for accuracy.
- Sensitive to initialization like all exponential-distance algorithms: use restarts, the variance regularization γ and a sensible number of rules.

and the rule consequents are local linear *MIMO ARX* models $\mathbf{y}^i(k) = \boldsymbol{\theta}_i^\mathsf{T}\boldsymbol{\varphi}_e(k)$ with a parameter matrix $\boldsymbol{\theta}_i$ that has n_y columns [104, 105]. The antecedent acts on a *scheduling vector* $\mathbf{x}(k)$, a subset of $\boldsymbol{\varphi}(k)$ that determines the operating point (typically a few lagged outputs). Only the output density changes in (8.10): the scalar error variance becomes the weighted error covariance matrix

$$\mathbf{F}_i^{yy} = \frac{\sum_k \mu_{ik}(\mathbf{y}_k - \boldsymbol{\theta}_i^\mathsf{T}\boldsymbol{\varphi}_{e,k})(\mathbf{y}_k - \boldsymbol{\theta}_i^\mathsf{T}\boldsymbol{\varphi}_{e,k})^\mathsf{T}}{\sum_k \mu_{ik}}, \tag{8.12}$$

$$p(\mathbf{y} \mid \boldsymbol{\varphi}, \eta_i) = \frac{\exp\!\big(-\tfrac{1}{2}(\mathbf{y} - \boldsymbol{\theta}_i^\mathsf{T}\boldsymbol{\varphi}_e)^\mathsf{T}(\mathbf{F}_i^{yy})^{-1}(\mathbf{y} - \boldsymbol{\theta}_i^\mathsf{T}\boldsymbol{\varphi}_e)\big)}{(2\pi)^{n_y/2}\sqrt{\det \mathbf{F}_i^{yy}}}, \tag{8.13}$$

and the local models are estimated column by column with the same weights (8.6). With local estimation, the model is a linear parameter-varying (LPV) system: at the operating point $\mathbf{x}$ its matrices are the sums of the local matrices weighted by the normalized firing strengths, $\mathbf{A}_j(\mathbf{x}) = \sum_i \beta_i(\mathbf{x})\mathbf{A}_j^i$, which is the interpretation model-based control needs [106]. A single rule base replaces n_y MISO models and shows the cross-couplings explicitly.

Dynamic models must be validated in *free run* (parallel simulation): the model is driven by the measured inputs only, and its own past outputs are fed back as regressors, $\hat{\mathbf{y}}(k) = f(\hat{\mathbf{y}}(k-1), \dots, \mathbf{u}(k-n_k), \dots)$. Errors can accumulate, so a model that predicts one step ahead well may still drift; the free-run error is the relevant measure when the model is used for simulation or control.

8.2.4 Fuzzy c-regression models

In switching regression the data are generated by c unknown linear laws $y = \mathbf{x}_e^\mathsf{T}\boldsymbol{\theta}_i$ that share the input range and may cross each other. A point prototype with a fixed norm cannot separate such data, and an adaptive norm (Gustafson–Kessel) describes each law only indirectly, by an elongated ellipsoid, so Hathaway and Bezdek [97] let the prototype be the regression model itself and measure the distance of a sample by its squared residual,

$$E_{ik}(\boldsymbol{\theta}_i) = \big(y_k - \mathbf{x}_{e,k}^\mathsf{T}\boldsymbol{\theta}_i\big)^2, \qquad J = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^m E_{ik}(\boldsymbol{\theta}_i). \tag{8.14}$$

Any error measure that keeps the minimizer property of the c-means functional may replace the squared residual. For models linear in the parameters, the minimizing $\boldsymbol{\theta}_i$ for fixed $\mathbf{U}$ is the weighted least-squares estimate (8.6) with $\boldsymbol{\Phi}_i = \operatorname{diag}(\mu_{i1}^m, \dots, \mu_{iN}^m)$, and the minimizing $\mathbf{U}$ for fixed models is the FCM update with E_{ik} in place of D_{ik}^2; the alternating optimization is a grouped coordinate descent and converges to a local minimum or saddle point, so random restarts are advisable. The memberships of a sample depend on the output, so they exist only for input–output *pairs*; for a new input the model can return the outputs of all c local models but cannot say which one applies. Prediction from the input alone needs

Algorithm 8.2 Fuzzy c-regression models (FCRM)

Input: regressors $\mathbf{X}$, outputs $\mathbf{y}$, number of models c, exponent $m > 1$, tolerance ϵ

initialize the partition $\mathbf{U}^{(0)}$ at random;
repeat
  models: $\boldsymbol{\theta}_i = (\mathbf{X}_e^\mathsf{T}\boldsymbol{\Phi}_i\mathbf{X}_e)^{-1}\mathbf{X}_e^\mathsf{T}\boldsymbol{\Phi}_i\mathbf{y}$, $\boldsymbol{\Phi}_i = \operatorname{diag}(\mu_{i1}^m, \dots, \mu_{iN}^m)$, $i = 1, \dots, c$;
  residuals: $E_{ik} = (y_k - \mathbf{x}_{e,k}^\mathsf{T}\boldsymbol{\theta}_i)^2$;
  partition: $\mu_{ik} = 1/\sum_{j=1}^{c}(E_{ik}/E_{jk})^{1/(m-1)}$;
until $\max_{ik}\left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: c regression models $\boldsymbol{\theta}_i$, partition $\mathbf{U}$ of the pairs $(\mathbf{x}_k, y_k)$

In a nutshell Fuzzy c-regression models

- Prototypes are regression models; the distance of a sample is its squared residual, so crossing linear laws are separated.
- Alternating weighted least squares and FCM membership update; the functional has local minima, so use restarts.
- Memberships are defined for input–output pairs only; a new input yields the c candidate outputs, not a single prediction.
- Building block of the hinging-hyperplane tree and a diagnostic tool for switching behavior in data.

an input-space partition, as in a TS model, or the hinging-hyperplane construction of section 8.2.7; both can be built on FCRM.

8.2.5 Supervised fuzzy clustering for fuzzy classifiers

A fuzzy rule-based classifier with one rule per class and a crisp class as consequent can represent only classes that are single compact regions. Abonyi and Szeifert [98] generalize the consequent to a vector of class probabilities,

$$\begin{aligned} r_i &: \text{If } x_1 \text{ is } A_{i1}(x_1) \text{ and } \dots \text{ and } x_n \text{ is } A_{in}(x_n) \\ &\quad \text{then } \hat{y} = c_1 \text{ with } P(c_1 \mid r_i), \dots, \ \hat{y} = c_C \text{ with } P(c_C \mid r_i) \quad [w_i], \end{aligned} \tag{8.15}$$

so that R rules, possibly more than the C classes, describe the data, and a class may be covered by several rules. The rules are aggregated with the normalized firing strengths, and the class with the largest posterior is chosen,

$$P(c_q \mid \mathbf{x}) = \frac{\sum_{i=1}^{R}\beta_i(\mathbf{x})P(c_q \mid r_i)}{\sum_{i=1}^{R}\beta_i(\mathbf{x})}, \qquad \hat{y} = \arg\max_q P(c_q \mid \mathbf{x}). \tag{8.16}$$

With Gaussian antecedents and the weights $w_i = P(r_i)/\,|2\pi\mathbf{F}_i|^{1/2}$ this is the Bayes decision rule of a classifier whose class-conditional densities are mixtures of (axis-parallel) Gaussians, an extension of the quadratic Bayes classifier, so the rule base inherits its probabilistic interpretation.

Unsupervised clustering estimates the density of the inputs without looking at the labels and therefore places clusters that ignore the class boundaries. The *supervised* algorithm includes the label in the sample, $\mathbf{z}_k = [\mathbf{x}_k^{\mathsf{T}}\ y_k]^{\mathsf{T}}$, and in the distance: the inverse Gath–Geva distance of the input is multiplied by the probability that the rule describes the class of the sample,

$$\frac{1}{D_{ik}^2} = P(r_i)\prod_{j=1}^{n}\frac{1}{\sqrt{2\pi\sigma_{ij}^2}}\exp\!\Big(-\frac{1}{2}\frac{(x_{jk}-v_{ij})^2}{\sigma_{ij}^2}\Big)\,P(y_k \mid r_i). \tag{8.17}$$

(The paper writes this distance without the normalizing constants of the Gaussians; with them $1/D_{ik}^2$ is the joint probability $p(\mathbf{x}_k, y_k, r_i)$, consistent with the weights w_i of the classifier. `fuzzyclust` implements both forms, selected by `normalize_density`.) The updates follow from the Lagrangian of $J = \sum_i \sum_k \mu_{ik}^m D_{ik}^2$ as for Gath–Geva clustering: centers and variances are μ^m-weighted means, the class probabilities are the membership mass of each class in the rule, $P(c_q \mid r_i) = \sum_{k:\,y_k=c_q}\mu_{ik}^m / \sum_k \mu_{ik}^m$, and $P(r_i) = \frac{1}{N}\sum_k \mu_{ik}^m$. A sample is thus attracted by a rule when it lies in the rule's region *and* the rule's consequent favors its class; the clustering pulls the rules apart along class boundaries that pass through a dense region of the input space.

Feature selection follows the same probabilistic view. With the within-rule and between-rule scatter matrices

$$\mathbf{F}_W = \sum_{i=1}^{R} P(r_i)\mathbf{F}_i, \quad \mathbf{F}_B = \sum_{i=1}^{R} P(r_i)(\mathbf{v}_i-\mathbf{v}_0)(\mathbf{v}_i-\mathbf{v}_0)^{\mathsf{T}}, \quad \mathbf{v}_0 = \sum_{i=1}^{R} P(r_i)\mathbf{v}_i, \tag{8.18}$$

the Fisher interclass separability $J = \det\mathbf{F}_B / \det\mathbf{F}_W$ measures how well separated and how compact the rules are. The importance of a feature is the value of J computed *without* it; the feature whose removal lowers J least is deleted, the classifier is re-identified on the reduced data, and the procedure stops when the accuracy starts to drop. (Since $\mathbf{F}_B$ has rank at most $R-1$, `fuzzyclust` uses the product of its $\min(R-1, n)$ largest eigenvalues in place of a vanishing determinant, which keeps the values comparable between feature subsets of equal size; the trace criterion $\mathrm{tr}(\mathbf{F}_W^{-1}\mathbf{F}_B)$ is available as well.)

8.2.6 Supervised fuzzy partitions for decision-tree induction

Decision-tree and rule-induction methods of the ID3 family [107] operate on discrete attributes, so continuous features are partitioned *a priori* and the partition fixes what the tree can express. Fuzzy sets make the partition soft and the tree a fuzzy decision tree [108], but the common a priori partitions (equal-width bins,

Algorithm 8.3 Supervised fuzzy clustering

Input: inputs $\mathbf{X}$, class labels $y_k \in \{c_1, \dots, c_C\}$, number of rules R, exponent m, tolerance ϵ

initialize the partition $\mathbf{U}^{(0)}$;
repeat
  for *every rule* i **do**
    $v_{ij} = \sum_k \mu_{ik}^m x_{jk} / \sum_k \mu_{ik}^m$, $\sigma_{ij}^2 = \sum_k \mu_{ik}^m (x_{jk} - v_{ij})^2 / \sum_k \mu_{ik}^m$;
    $P(c_q \mid r_i) = \sum_{k:\, y_k = c_q} \mu_{ik}^m / \sum_k \mu_{ik}^m$ for $q = 1, \dots, C$;
    $P(r_i) = \frac{1}{N} \sum_k \mu_{ik}^m$, $w_i = P(r_i) \prod_j (2\pi\sigma_{ij}^2)^{-1/2}$;
  distances D_{ik}^2 by (8.17); partition $\mu_{ik} = 1 / \sum_j (D_{ik}/D_{jk})^{2/(m-1)}$;
until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: rules r_i with Gaussian sets, class probabilities and weights; classification by (8.16)

In a nutshell Supervised fuzzy clustering

- Rules carry a probability for every class, so the number of rules is free and classes made of several groups are handled; the classifier is a Gaussian-mixture Bayes classifier.
- The class label enters the distance; the clustering is the supervised counterpart of Gath–Geva clustering and identifies the classifier in one pass.
- Choose R by cross-validation, regularize the variances for large R, and prune the features with the interclass separability criterion.
- The projections of the rules onto the features provide the supervised fuzzy partitions of section 8.2.6.

equal-frequency bins or a Ruspini partition of triangles [1]) ignore the class labels; Dougherty, Kohavi and Sahami [109] survey these unsupervised discretizations and their supervised counterparts. An unsupervised clustering of each feature ignores the labels as well: like equal-frequency binning, it adapts the sets to the density of the samples rather than to the class boundaries. Abonyi [99] derives the partition from the supervised clustering of section 8.2.5 in four steps (fig. 8.3):

1. *Supervised clustering* with R rules; every rule projected onto feature j gives a Gaussian set (v_{ij}, σ_{ij}), so each feature starts with R sets whose positions follow the class structure.
2. *Trapezoids*: since the tree learner needs triangular or trapezoidal sets, every Gaussian is replaced by the trapezoid with the corner points

$$a_{ij} = v_{ij} - 3\sigma_{ij}, \quad b_{ij} = v_{ij} - \tfrac{1}{2}\sigma_{ij}, \quad c_{ij} = v_{ij} + \tfrac{1}{2}\sigma_{ij}, \quad d_{ij} = v_{ij} + 3\sigma_{ij}. \tag{8.19}$$

3. *Merging (pre-pruning)*: sets of the same feature that overlap strongly are merged on the basis of the similarity measure of Setnes et al. [42],

$$S(A_{ij}, A_{lj}) = \frac{|A_{ij} \cap A_{lj}|}{|A_{ij} \cup A_{lj}|}, \tag{8.20}$$

the ratio of the cardinalities of the intersection (min) and the union (max) of the two membership functions; pairs with $S > \theta$ (the paper uses $\theta = 0.8$) are merged, most similar first, into a set whose center and width are the prior-weighted combination of the two. A feature left with a single set carries no information about the classes and is a *don't care* variable that can be dropped.

4. *Tree induction and post-pruning*: the sets are the a priori partition of the tree learner (the fuzzy ID3 variant FID in the paper); the leaves of the resulting tree store the class frequencies and are read as rules of the form (8.15), and rules with a prior below a cut level α_c are removed.

The cut points of the partition therefore sit where the classes change, and the tree is small because every attribute test is informative.

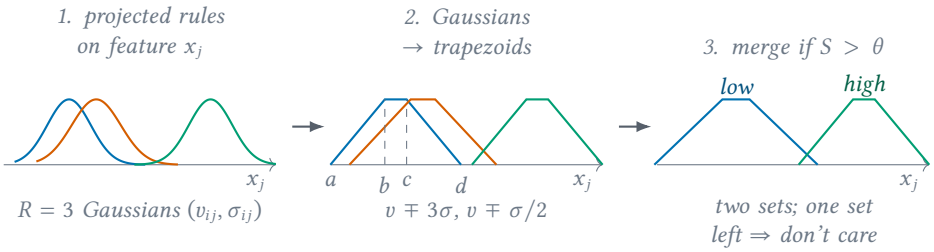


Figure 8.3. Supervised fuzzy partition of one feature. The Gaussian antecedent sets of the R rules are projected onto the feature (left), converted into trapezoids with corner points $v \mp 3\sigma$ and $v \mp \sigma/2$ (middle), and sets whose Jaccard similarity exceeds the threshold θ are merged (right). A feature whose sets all merge into one does not separate the classes and is dropped.

8.2.7 Hinging-hyperplane regression trees

A *hinging hyperplane* [110] is the continuous piecewise-linear function of two hyperplanes joined along their intersection,

$$\begin{aligned} h(\mathbf{x}) &= \max\left(\mathbf{x}_e^\top \boldsymbol{\theta}_1,\ \mathbf{x}_e^\top \boldsymbol{\theta}_2\right) \quad \text{or} \quad \min\left(\mathbf{x}_e^\top \boldsymbol{\theta}_1,\ \mathbf{x}_e^\top \boldsymbol{\theta}_2\right), \\ \text{hinge: } &\{\mathbf{x} \ :\ \mathbf{x}_e^\top(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) = 0\}, \end{aligned} \tag{8.21}$$

and sums of hinge functions approximate smooth functions arbitrarily well. Finding one hinge is a nonsmooth least-squares problem with local minima, for which Breiman's iterative algorithm has convergence and range problems. Kenesei and Abonyi [100] observed that fuzzy c-regression with $c = 2$ (section 8.2.4) identifies

Algorithm 8.4 Supervised fuzzy partitioning of continuous features

Input: inputs $\mathbf{X}$, labels $\mathbf{y}$, number of rules R, similarity threshold θ

identify R rules by supervised fuzzy clustering (algorithm 8.3);
for *every feature* j **do**
  sets A_{ij}: trapezoids (8.19) from (v_{ij}, σ_{ij}), $i = 1, \dots, R$;
  while *the most similar pair* (A_{ij}, A_{lj}) *has* $S > \theta$ *by* (8.20) **do**
    merge the pair into one set (prior-weighted center and width);
  if *one set is left* **then** mark feature j as don't care;
encode every sample by the index of its most active set (or by its memberships) and induce the tree;

Output: fuzzy partition of every feature, list of don't care features, decision tree / rule base

In a nutshell Supervised fuzzy partitions

- A priori discretization whose cut points come from a supervised clustering and therefore sit where the classes change.
- Gaussian sets become trapezoids; similar sets are merged; features with one set are discarded.
- Gives small trees with linguistically meaningful tests; a tree that optimizes its own thresholds (CART) may still be more accurate on axis-parallel problems.
- Parameters: initial number of sets R (3–5) and the merging threshold $\theta \approx 0.8$.

two linear models and their fuzzy operating regions at once, which is a hinge if the two planes cross inside the data (fig. 8.4, top). With $\mathbf{v}_i = \sum_k \mu_{ik}^m \mathbf{x}_k / \sum_k \mu_{ik}^m$ as the weighted center of the regressors of cluster i, the crossing is enforced by the *relative constraint*

$$\begin{aligned} &\mathbf{v}_{e,1}^\mathsf{T}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) < 0 \text{ and } \mathbf{v}_{e,2}^\mathsf{T}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) > 0 \\ \text{or} \quad &\mathbf{v}_{e,1}^\mathsf{T}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) > 0 \text{ and } \mathbf{v}_{e,2}^\mathsf{T}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) < 0, \end{aligned} \tag{8.22}$$

that is, the two cluster centers lie on opposite sides of the hinge. When the unconstrained weighted least-squares solution violates it, the two models are re-estimated jointly by a quadratic program $\min_{\boldsymbol{\theta}} \frac{1}{2}\boldsymbol{\theta}^\mathsf{T}\mathbf{H}\boldsymbol{\theta} + \mathbf{c}^\mathsf{T}\boldsymbol{\theta}$ with the linear constraints (8.22) (`fuzzyclust` solves it with SLSQP for both sign patterns and keeps the better one); local or global constraints on the parameters can be added in the same way. The hinge type (max or min) is the one with the smaller squared error on the node data.

A single hinge is a coarse model, and a sum of hinges fitted to all data is hard to interpret. The paper therefore builds a *binary tree*: the hinge of the root splits the data into the two operating regions (the samples where plane 1 is active go to the left child), each region is modeled by a hinge again, and so on, until a stopping rule holds: the depth limit is reached, a child would be too small for a two-plane fit, the error does not decrease ($E_1 + E_2 \geq E$ of the parent), or the node error is below a tolerance. The tree is grown either *balanced* (every branch to the same depth) or *greedy* (the leaf with the largest error is split next, until a budget of leaves is spent). Internal nodes split along oblique hinges, so no splitting variable has to be chosen, and the leaves are hinge functions (or single planes). The whole model is therefore piecewise linear, with a transparent structure that can be used directly for instantaneous linearization in predictive control. Each node reports its RMSE and the *condition*

$$\varrho = 1 - \frac{|m_1 - \sum_k \mu_{1k}|}{\sum_{i=1}^{2} \sum_k \mu_{ik}}, \tag{8.23}$$

where m_1 is the number of samples the crisp hinge sends to the side of plane 1 and $\sum_k \mu_{1k}$ the membership mass of model 1. A value of ϱ close to one means that the hinge does not override the fuzzy partition found by FCRM, that is, the hinge identification at that node is reliable.

8.3 Using it in fuzzyclust

The estimators of this chapter are in `fuzzyclust.models`. All of them are supervised in the scikit-learn sense: `fit(X, y)` takes the regressors and the outputs (or the class labels), the clusterers inherit the alternating-optimization loop of chapter 2, and the regressors and the classifier follow the `RegressorMixin` / `ClassifierMixin` conventions (`score` is R^2 or the accuracy). Regression matrices of dynamic systems are built with `fuzzyclust.preprocessing.make_regressors`.

`fuzzyclust.models.TakagiSugeno`

`n_clusters=3, m=2.0, antecedent="gaussian_axis_parallel"|"gg_projected"|"gk_projected", consequent="local"|"global"|"tls", antecedent_features=None, gamma=0.0, beta=1e15, tol, max_iter, init="kmeans++", n_init=1, restart_criterion="likelihood"|"error", random_state, verbose.`
`fit(X, y)` with `y` of shape `(N,)` or `(N, n_outputs)`; `predict(X)` is the TS output (8.2); `firing_strengths(X)`, `antecedent_membership(X)`, `local_predict(X)`; `rule_base()` (DataFrame) and `describe()` (text). Attributes `centers_`, `sigmas_`, `rule_weights_`, `local_models_` `(c, n+1, n_outputs)`, `coef_`, `intercept_`, `membership_`, `clusterer_` (the engine).

Algorithm 8.5 Hinging-hyperplane regression tree

Input: regressors $\mathbf{X}$, outputs $\mathbf{y}$, maximum depth, minimum leaf size, growth strategy (balanced or greedy), tolerance

Function `FitHinge`(*node data*)**:**
  run FCRM with $c = 2$ (algorithm 8.2, several restarts, keep the lowest J);
  if *the centers* $\mathbf{v}_1, \mathbf{v}_2$ *are on the same side of the hinge* **then**
    re-estimate $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$ by the constrained quadratic program (8.22) and update $\mathbf{U}$
  choose max or min by the smaller squared error; compute RMSE and ϱ by (8.23);

Function `Grow`(*node*)**:**
  if *depth limit, RMSE below tolerance or a child would be smaller than the minimum* **then**
    return
  split the node data by the sign of $\mathbf{x}_e^{\mathsf{T}}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)$; `FitHinge` on both parts;
  if $E_1 + E_2 \geq E$ **then return** (the split does not help);
  attach the children; `Grow` both (balanced), or push them to the list of candidate leaves (greedy);

root ← `FitHinge`(*all data*); `Grow`(*root*) (greedy: repeatedly split the leaf with the largest error until the leaf budget is spent);

Output: binary tree of hinge nodes; prediction by descending to a leaf and evaluating its hinge

In a nutshell Hinging-hyperplane regression tree

- FCRM with two models identifies a hinge; the crossing constraint keeps the hinge inside the data.
- Recursive hinge identification grows a binary tree with oblique splits (no splitting variable) and piecewise-linear leaves.
- Complexity is controlled by the depth, the leaf size and the leaf budget of greedy growth; the node RMSE and the condition ϱ guide the construction.
- The local linear models of the leaves make the model directly usable for linearization-based predictive control.

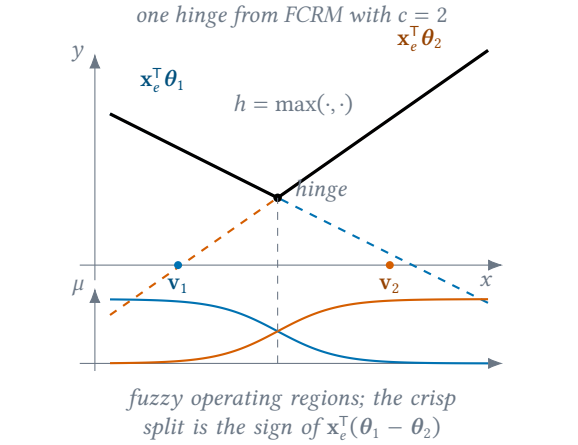


Figure 8.4. Hinging-hyperplane regression tree. Top: fuzzy c-regression with two models yields two lines and their fuzzy operating regions; the lines cross between the cluster centers $\mathbf{v}_1$, $\mathbf{v}_2$ (the crossing constraint), and the hinge function is their maximum or minimum. Bottom: the sign of $\mathbf{x}_e^\top(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)$ splits the data; each part is modeled by a new hinge, which gives a binary tree with oblique splits and piecewise-linear leaves.

`antecedent_features` selects the scheduling variables; the consequents always use all regressors.

`fuzzyclust.models.ModifiedGG`

A `BaseFuzzyClusterer`, the clustering engine of the default `TakagiSugeno`, with `fit(X, y)`, the parameters of `TakagiSugeno` without `antecedent`, `consequent` and `beta`, plus `var_floor=1e-6` (and `n_init=3` by default),

and the fitted rule parameters `cluster_centers_`, `input_variances_`, `local_models_`, `error_covariances_`, `priors_`, `rule_weights_`, `log_likelihood_`, `restart_scores_`. `predict(X)` returns the TS output, `predict_proba(X[, y])` the memberships (posteriors given the input, or given input and output), `transform(X[, y])` the distances, `describe_rules()` the rule table.

`fuzzyclust.models.simulate_free_run(model, u, y_init, n_a, n_b=0, n_k=1)`

Free-run simulation of any regressor trained on the regressors of `make_regressors`: the predicted outputs are fed back while the measured inputs `u` (`None` for autonomous models) drive the model, and `y_init` supplies the initial outputs up to the largest lag. Works for single- and multi-output models.

`fuzzyclust.models.FuzzyCRegression`

`n_clusters=2`, `m=2.0`, `fit_intercept=True`, `tol`, `max_iter`, `init="random"|"kmeans++"|array`, `n_init=1`, `random_state`, `verbose`.
`fit(X, y)`; `coef_`, `intercept_`, `local_models_`, `membership_`, `labels_`. `predict(X)` returns the outputs of *all* local models `(N, c)`; `predict_proba(X, y)`, `predict_labels(X, y)` and `transform(X, y)` (absolute residuals) need the pairs.

`fuzzyclust.models.SupervisedFuzzyClassifier`

`n_clusters=3`, `m=2.0`, `gamma=0.0`, `var_floor=1e-6`, `normalize_density=True`, `tol`, `max_iter`, `init="kmeans++"`, `n_init=3`, `random_state`, `verbose`.
A classifier: `predict_proba(X)` are class posteriors (8.16), `predict(X)` the labels, `cluster_membership(X[, y])` the rule memberships; `classes_`, `cluster_centers_`, `input_variances_`, `class_probs_` `(R, C)`, `priors_`, `rule_weights_`; `rule_base()`, `interclass_separability(features, criterion)`, `feature_importance()`. The function `select_features_by_separability(X, y, n_features_to_select, **params)` runs the backward elimination and returns the kept features, the elimination order and a history of separability and accuracy.

fuzzyclust.models.FuzzyPartitionDiscretizer

`n_sets=3`, `shape="trapezoid"|"gaussian"|"triangle"`, `merge_threshold=0.8`, `encode="crisp"|"membership"`, `method="supervised"|"ruspini"`, `drop_single=False`, `m`, `tol`, `max_iter`, `init`, `random_state`.
A transformer: `fit(X, y)`, `transform(X)` returns one set index per feature (floats, like `KBinsDiscretizer`) or the memberships in all sets; `partitions_` (centers, widths, merged rule groups per feature), `n_sets_`, `active_features_`, `classifier_`; `membership(X)`, `evaluate(feature, grid)`, `get_feature_names_out()`. The functions `supervised_fuzzy_partition`, `ruspini_partition`, `membership_1d` and `set_similarity` expose the building blocks.

fuzzyclust.models.HingingHyperplaneTree

`max_depth=3`, `min_samples_leaf=10`, `growth="balanced"|"greedy"`, `max_leaves=None`, `tol_rmse=0.0`, `leaf_model="hinge"|"linear"`, `constrain_hinge=True`, `m=2.0`, `n_init=3`, `fcrm_max_iter=100`, `fcrm_tol=1e-4`, `random_state`.
A regressor: `predict(X)`, `apply(X)` (leaf index), `tree_` (root `HingeNode` with `theta`, `kind`, `rmse`, `condition`, children), `leaves_`, `n_leaves_`, `depth_`; `node_table()` and `describe()`.

data generators and plot helpers

`make_sinc`, `make_second_order_plant` (the benchmark plant, random or sinusoidal excitation), `make_mimo_plant`, `make_switching_lines`, `make_two_class_overlap`, `make_banded_classes`, `make_piecewise_linear`; `plot_membership_functions(model, X)`, `plot_local_models(model, X, y)`, `plot_decision_regions(clf, X, y, proba_class)`, `plot_fuzzy_partition(disc, feature, X, y)`, `plot_hinge_tree_surface(tree, X)`.

A TS model of a static function is identified and inspected in three lines:

```
ts = TakagiSugeno(n_clusters=4, random_state=0).fit(X, y)
print(ts.describe(digits=3))
print(f"R^2 on noise-free test data: {ts.score(X_test, y_test):.3f}")
```

For a dynamic system, the regressors are built first and the model is validated in free run:

```
u_id, y_id = make_second_order_plant(n_samples=200, excitation="random",
    random_state=0)
u_val, y_val = make_second_order_plant(n_samples=200, excitation="sine",
    random_state=1)


def nonlinear_part(u, y):
    Phi, target, names = make_regressors(y, n_a=2)  # rows [y(k-1),
    y(k-2)], k = 2, ..., N-1
    return Phi, target - u[2:], names


X_id, f_id, names = nonlinear_part(u_id, y_id)
X_val, f_val, _ = nonlinear_part(u_val, y_val)
print("regressors:", names)
```

```
ts = TakagiSugeno(n_clusters=4, n_init=5, restart_criterion="error",
    random_state=0).fit(X_id, f_id)
y_pred = ts.predict(X_val) + u_val[2:]  # one-step-ahead prediction of y(k)
y_sim = free_run(ts, u_val, y_val[:2])  # free-run simulation
```

A MIMO model schedules on the lagged outputs and is simulated with `simulate_free_run`:

```
ts = TakagiSugeno(n_clusters=3, antecedent_features=[0, 1], n_init=5,
    restart_criterion="error",
                  random_state=0).fit(Phi_id, Y_id)
print(ts.rule_base(feature_names=names, output_names=["y1",
    "y2"]).round(3).to_string(index=False))
y_sim = simulate_free_run(ts, u_val, y_val[:1], n_a=1, n_b=1, n_k=1)
```

Switching regression, the fuzzy classifier and the hinge tree follow the same pattern:

```
fcm = FCM(n_clusters=2, random_state=0).fit(Z)
gk = GK(n_clusters=2, random_state=0).fit(Z)
fcrm = FuzzyCRegression(n_clusters=2, n_init=5, random_state=0).fit(X, y)
print("slopes    :", fcrm.coef_[:, 0].round(3))
print("intercepts:", fcrm.intercept_.round(3))
```

```
clf2 = SupervisedFuzzyClassifier(n_clusters=2, random_state=0).fit(X, y)
clf4 = SupervisedFuzzyClassifier(n_clusters=4, random_state=0).fit(X, y)
print(f"training accuracy: 2 rules {clf2.score(X, y):.3f}, 4 rules
    {clf4.score(X, y):.3f}")
print(clf4.rule_base().round(3).to_string(index=False))
```

```
tree = HingingHyperplaneTree(max_depth=2, min_samples_leaf=10,
    random_state=0).fit(X, y)
print(tree.describe(feature_names=["x1", "x2"], digits=3))
print(tree.node_table().round(3).to_string(index=False))
print(f"test RMSE: {np.sqrt(np.mean((tree.predict(X_test) - y_test) **
    2)):.4f}")
```

and the supervised partition is a transformer in a scikit-learn pipeline:

```
pipe = make_pipeline(
    FuzzyPartitionDiscretizer(n_sets=3, drop_single=True, random_state=0),
    DecisionTreeClassifier(max_depth=2, random_state=0),
).fit(X, y)
codes_names = list(pipe[0].get_feature_names_out(feature_names))
print(export_text(pipe[-1], feature_names=codes_names))
```

8.4 Examples

8.4.1 Takagi–Sugeno model of the sinc function

Three hundred noisy samples of $y = \sin x/x$ on $[-10, 10]$ are modeled with four rules (fig. 8.5). The local linear models (colored) are the linear laws of the clusters, drawn thick where the rule fires; the black curve is the TS output, a smooth interpolation whose weights are the shaded firing strengths, and the Gaussian sets below are the clusters themselves (nothing has been projected). Figure 8.6 compares the three antecedent constructions with four rules each: the modified GG model is the most accurate ($R^2 = 0.96$ on noise-free test data), the projected GK model follows, and the projected GG model, whose clusters have free volumes and lean most strongly along the local slopes, loses the central peak ($R^2 = 0.86$), which makes the decomposition error visible. Figure 8.7 shows two of them in the product space: GK clusters are oblique ellipses in the (x, y) plane, modified GG clusters are Gaussians in x with a line inside. The consequent estimators differ as predicted: global least squares reaches the smallest RMSE (0.067 against 0.074), but its slopes no longer match the local slopes of the sinc. Finally, fig. 8.8 scans the number of rules. Both errors fall from 0.24 with two rules to 0.04 with six, though not monotonically: the five-rule model is worse than the four-rule one (training RMSE 0.096 against 0.074), because the clustering, not the prediction error, places the rules. From six rules on, the training error approaches the noise level of 0.02, and the error on the noise-free test data keeps falling, to 0.016 with twelve rules, so up to twelve rules the model does not yet fit the noise.

8.4.2 Nonlinear dynamic plant with free-run validation

The second-order plant of [96] is identified from 200 samples with a random input and validated on 200 samples with a sinusoid (fig. 8.9). The TS model approximates the nonlinear component $f(y(k-1), y(k-2))$ with four rules; fed with its own outputs, it reproduces the validation output of an excitation it has never seen (fig. 8.10, free-run MSE $1.3 \cdot 10^{-2}$ against $5 \cdot 10^{-3}$ for one-step prediction). The antecedent sets on the two lagged outputs (fig. 8.11) and the identified surface (fig. 8.12) show the operating regimes: the model matches f where there are data and extrapolates linearly outside, and the input space is tiled by the four rules.

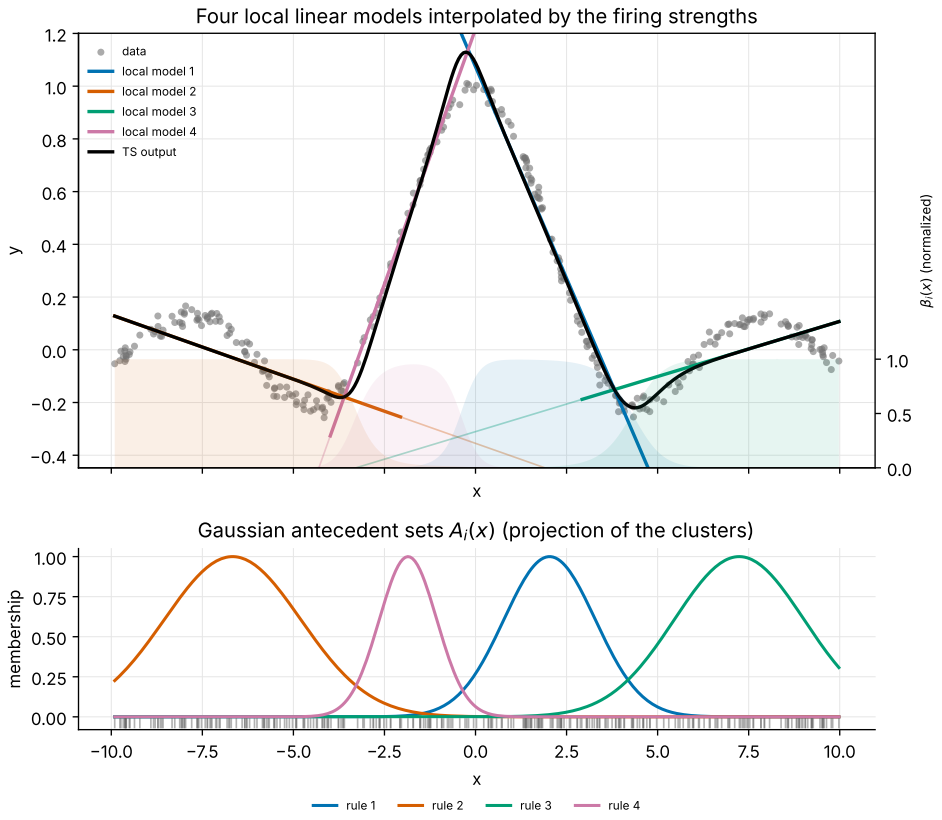


Figure 8.5. Four rules identified by modified Gath–Geva clustering on the sinc function. Top: local linear models (thick where the rule fires), TS output (black) and normalized firing strengths (shaded). Bottom: the Gaussian antecedent sets, which are the clusters themselves.

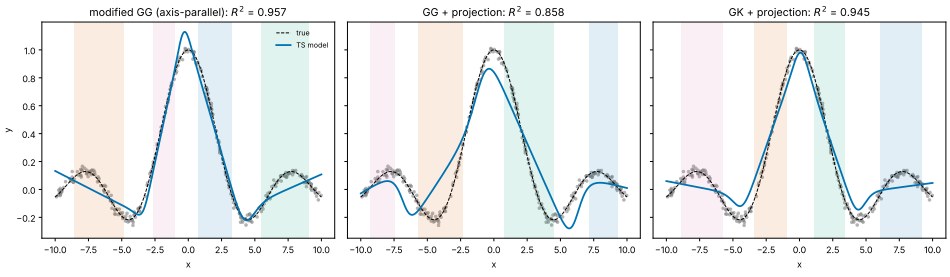


Figure 8.6. The same data with four rules from the modified GG algorithm, from Gath–Geva clustering with projected antecedents and from Gustafson–Kessel clustering with projected antecedents (shaded bands: $\pm\sigma_i$ of the sets). The projection of the oblique Gath–Geva clusters cuts off the central peak.

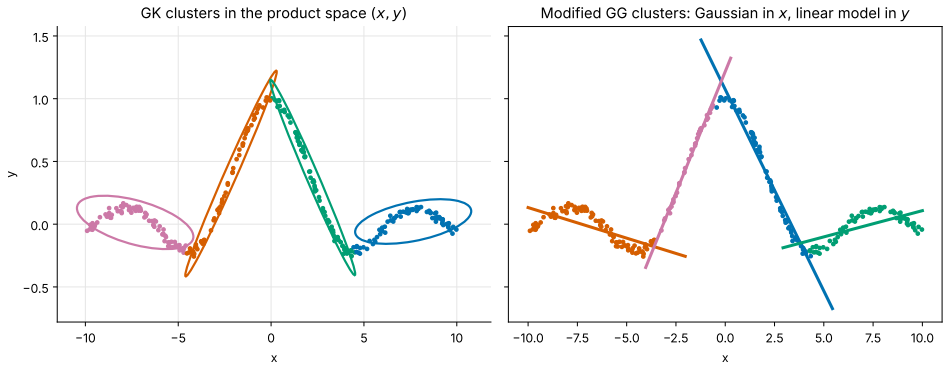


Figure 8.7. The product space (x, y): Gustafson–Kessel clusters are oblique ellipses whose width along x is not the width of their operating region (left); modified Gath–Geva clusters are Gaussians in x carrying a line (right).

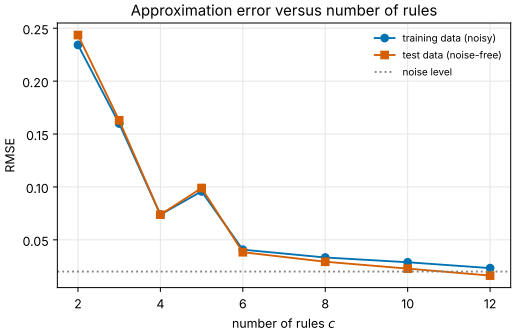


Figure 8.8. RMSE on the noisy training data and on noise-free test data as a function of the number of rules; apart from the poorer five-rule solution, both errors fall, and the test error keeps falling up to twelve rules.

The scan in fig. 8.13 repeats the paper's experiment over the number of rules and compares the two restart criteria: the training error decreases monotonically, the free-run error on the sinusoid no longer decreases systematically beyond three rules (it fluctuates between $5 \cdot 10^{-3}$ and $2 \cdot 10^{-2}$ up to eight rules) and grows tenfold for twelve rules (the sinusoid visits parts of the state space the random excitation has barely covered), and selecting the restart by the TS training error avoids the poor two-rule solution that the likelihood criterion picks.

8.4.3 MIMO model of a two-input two-output process

A process with two coupled states and operating-point-dependent input gains is identified with three rules whose antecedents act on the lagged outputs. Figure 8.14 shows the free-run simulation of both outputs on a validation sequence of random steps (RMSE 0.10 and 0.08); because the two outputs are fed back jointly, the cross-couplings are represented inside every local MIMO model. The antecedent

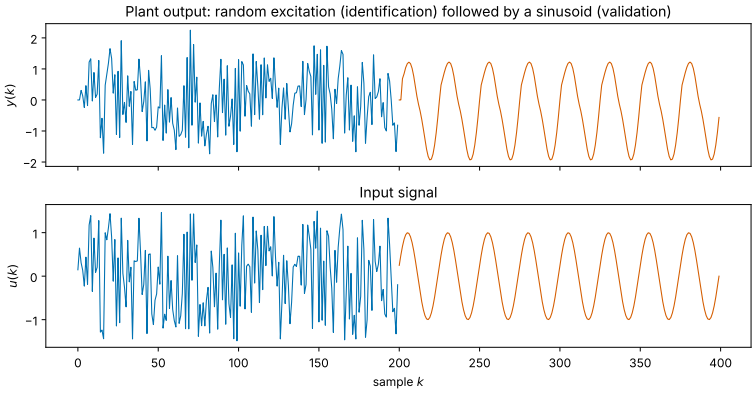


Figure 8.9. The second-order plant is identified on data with a random input (left part) and validated on data with a sinusoidal input (right part).

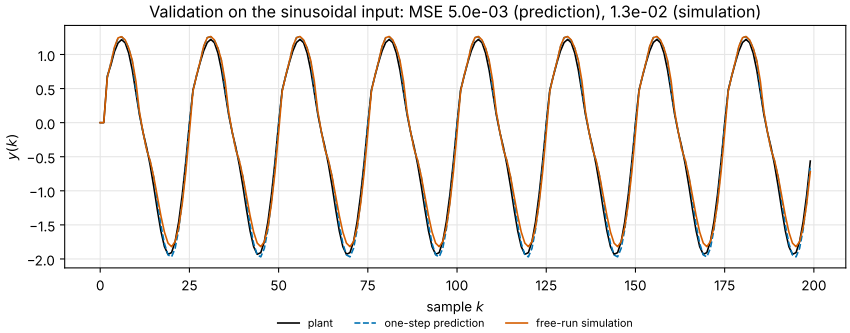


Figure 8.10. Validation on the sinusoidal input: one-step-ahead prediction and free-run simulation of the four-rule model follow the plant closely although the model was identified under random excitation.

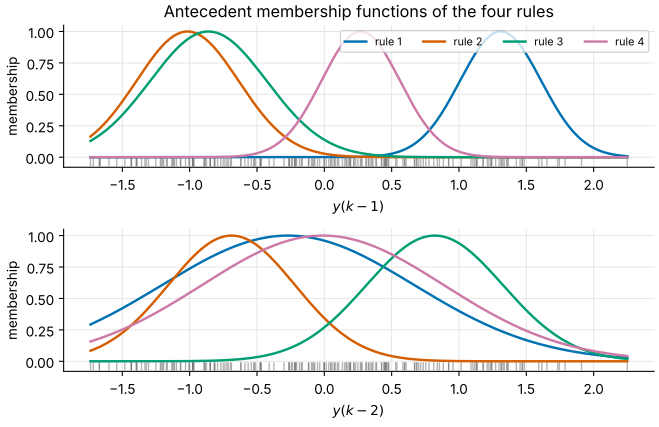


Figure 8.11. The antecedent membership functions of the four rules tile the ranges of $y(k-1)$ and $y(k-2)$.

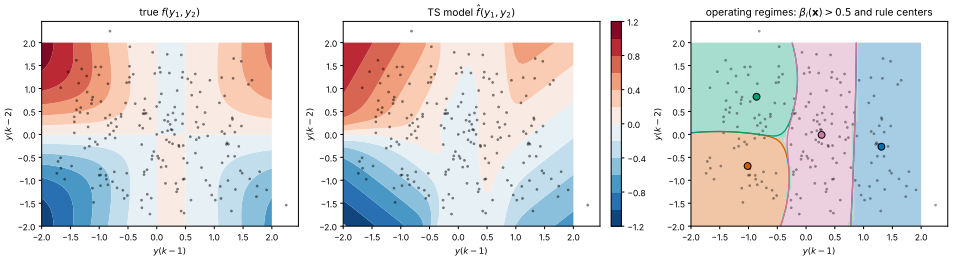


Figure 8.12. True nonlinearity $f(y_1, y_2)$ (left), TS model (middle) and operating regimes with the rule centers (right); dots are the identification samples. The model is accurate where there are data and extrapolates linearly.



Figure 8.13. Training and free-run error as a function of the number of rules for the two restart criteria. The training error decreases monotonically, the free-run error on the different excitation saturates and grows again for twelve rules.

sets (fig. 8.15) divide the operating range of the outputs into three regimes, and the local input gains, compared with the true gains at the rule centers, give the LPV interpretation. The rule at large outputs matches them closely (0.42 and 0.27 against 0.41 and 0.27); the central rule has the largest $u_2 \to y_2$ gain, as it should, but the membership-weighted fit over a wide region flattens the $u_1 \to y_1$ gains of the two rules near the origin (0.49 and 0.47 against 0.56 and 0.60). Figure 8.16 scans the number of rules for three choices of the scheduling vector: one rule is the linear MIMO ARX model, and adding rules lowers the free-run error on the whole, though not at every step (with scheduling on the outputs, from 0.114 for one rule to 0.063 for six, with three and four rules equally good); scheduling on the outputs (or on all regressors) works, whereas scheduling on the inputs brings little because the inputs do not define the operating point of this process.

8.4.4 Switching regression with fuzzy c-regression models

Three hundred samples of two crossing lines are clustered in fig. 8.17. FCM splits the plane into halves (ARI 0.28). GK finds two elongated ellipsoids that follow the lines and labels the points as well as FCRM does (ARI 0.96 for both, the same three points at the crossing being mislabeled), but it describes the laws only by

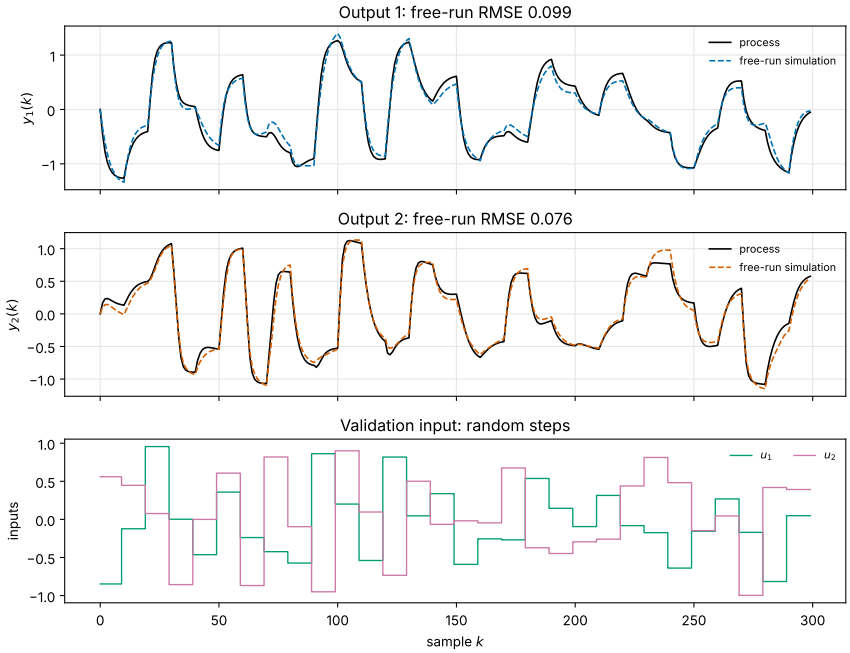


Figure 8.14. Free-run simulation of the three-rule MIMO TS model on validation data with random step inputs; both outputs are simulated jointly.

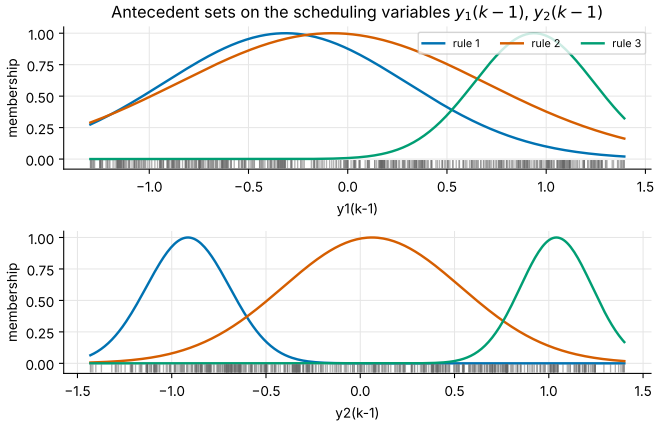


Figure 8.15. The antecedent sets on the two scheduling variables $y_1(k-1)$ and $y_2(k-1)$ divide the operating range into three regimes.

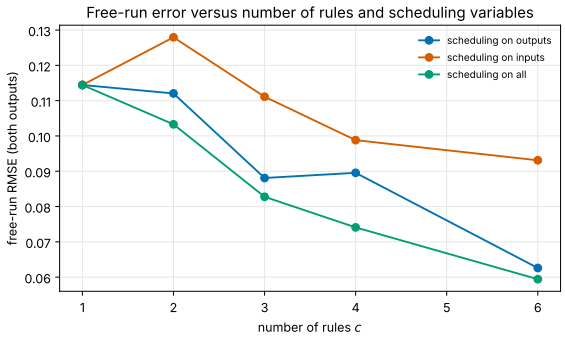


Figure 8.16. Free-run RMSE versus the number of rules for scheduling on the outputs, on the inputs and on all regressors: scheduling on the inputs brings little.

ellipsoids. FCRM recovers the slopes and intercepts to within 0.02 and assigns every point to the line with the smaller residual. The memberships (fig. 8.18) are fuzzy only around the crossing, and the objective decreases monotonically over eleven iterations. With three lines, two of them parallel, the functional has several minima (fig. 8.19): of twelve runs from random partitions, six reach $J = 3.9$ and the correct lines, and the other six end in solutions with $J = 14.7$ (four runs) or 46.3 (two runs) in which one model averages the parallel lines. With `n_init` restarts, the run with the lowest J is the right one. The exponent m (fig. 8.20) changes the crispness of the partition but hardly changes the lines: the weighted least-squares fits are insensitive to m as long as the residual ratios are sharp.

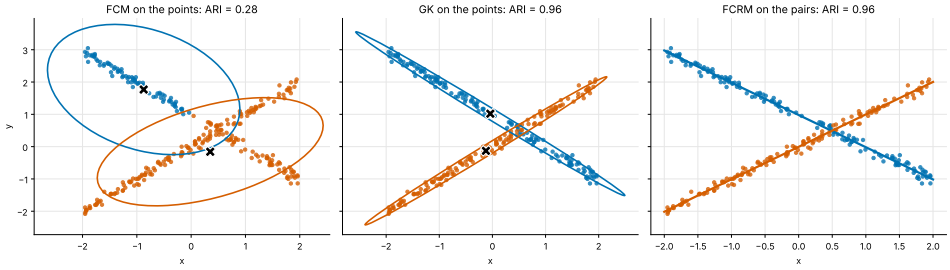


Figure 8.17. Two crossing lines: FCM cuts the points into two halves, GK follows the lines with two elongated ellipsoids, and FCRM clusters the pairs with regression prototypes and recovers the lines themselves.

8.4.5 Fuzzy classifier from supervised clustering

Each of two classes consists of two Gaussian blobs in an XOR layout. With two rules, the clustering can only place one rule on a blob and one broad rule in the middle whose consequent mixes the classes, and the accuracy is 0.70; with four rules, every blob gets a nearly pure rule, and the decision regions show the XOR pattern (fig. 8.21, accuracy 0.94). The posterior of class 1 (fig. 8.22) is smooth

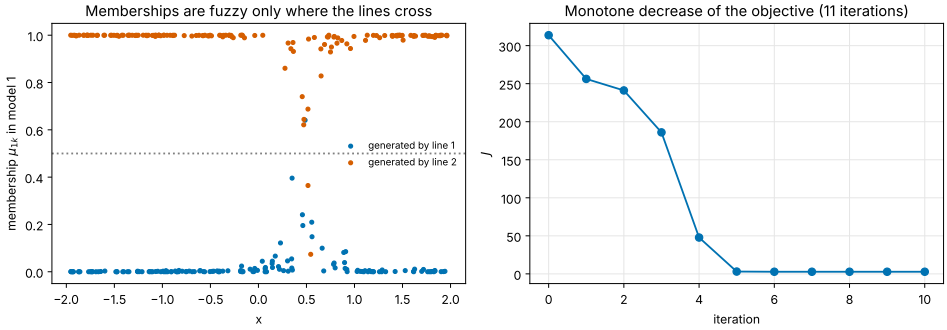


Figure 8.18. Left: membership in the first model along x, fuzzy only where the lines cross. Right: monotone decrease of the FCRM objective.

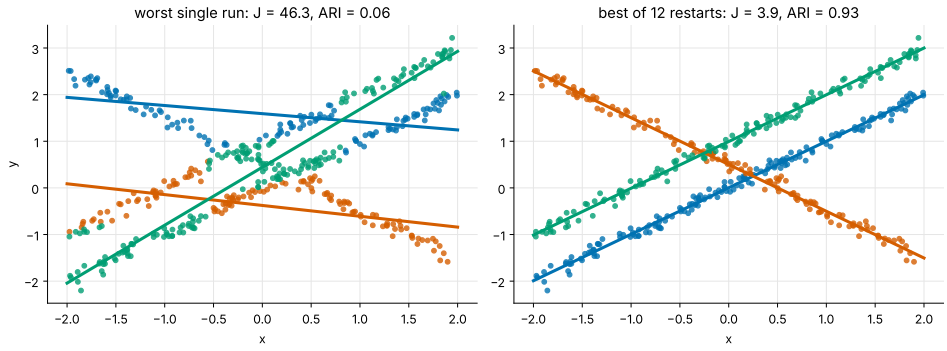


Figure 8.19. Three lines, two of them parallel: the worst of twelve random starts averages the parallel lines (left); the best restart recovers all three (right).

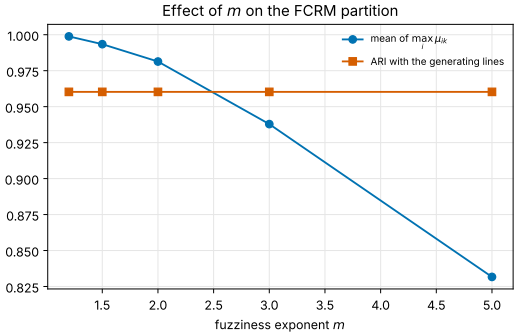


Figure 8.20. Effect of the exponent m: the partition becomes fuzzier, the agreement with the generating lines stays.

between the rules, and the antecedent sets come in pairs on each feature (fig. 8.23). Cross-validation over the number of rules (fig. 8.24) confirms that four rules are needed and that the variance regularization $\gamma = 0.1$ keeps the accuracy at 0.94 for twelve rules, where the unregularized rules start to shrink onto subsets. On the wine data (fig. 8.25), three rules on all thirteen attributes classify 98.9 % of the wines; the backward elimination by interclass separability removes magnesium, alcalinity of ash, malic acid and color intensity first and keeps 95–98 % accuracy down to four attributes (alcohol, flavanoids, nonflavanoid phenols and hue, 96.1 %); with three attributes the accuracy drops to 89 %.

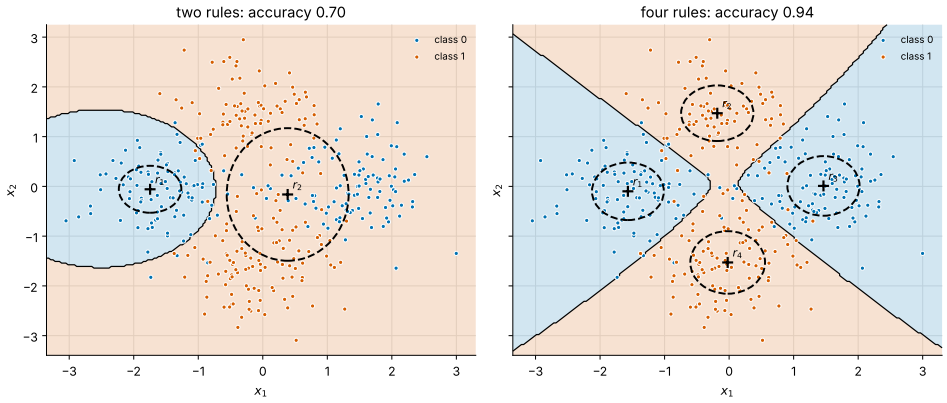


Figure 8.21. Decision regions of the fuzzy classifier with two rules (left) and four rules (right) on two classes made of two blobs each; dashed ellipses are the $\pm\sigma$ extents of the rules.

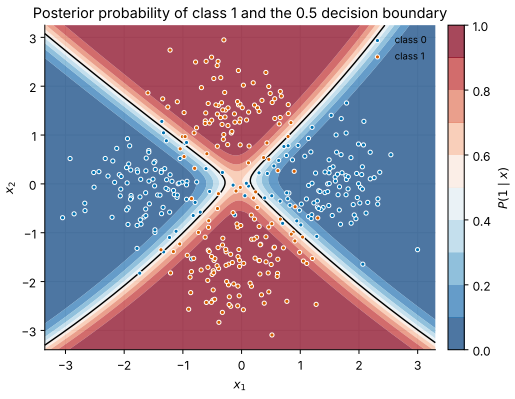


Figure 8.22. Posterior probability of class 1 and the 0.5 boundary of the four-rule classifier: the posterior changes smoothly between the rules.

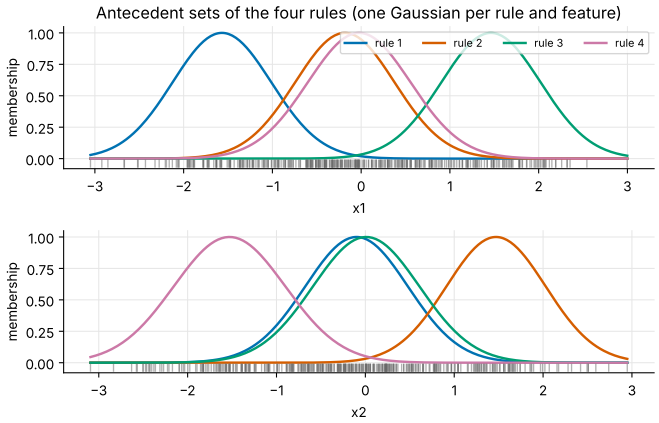


Figure 8.23. The antecedent sets of the four rules come in pairs on each of the two features.

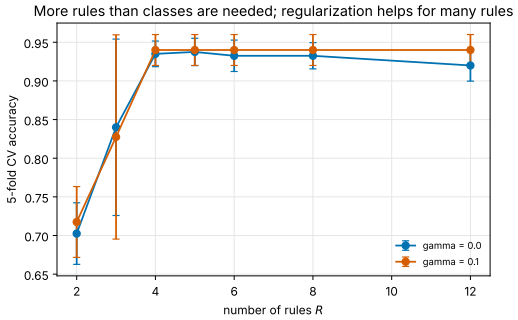


Figure 8.24. Cross-validated accuracy versus the number of rules without and with variance regularization: four rules are needed, and the regularized classifier keeps its accuracy for many rules.

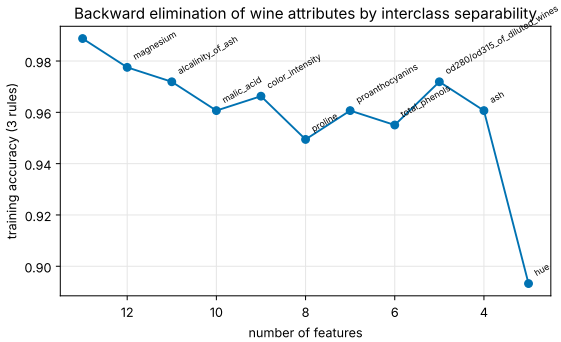


Figure 8.25. Backward elimination of the wine attributes by interclass separability: the training accuracy of the three-rule classifier drops only when fewer than four attributes remain.

8.4.6 Supervised fuzzy partitions and decision trees

Three classes are separated by thresholds on two features; a third feature is noise. The supervised partition (fig. 8.26, top) merges two of the three sets on x_1 (the rules of classes 1 and 2 coincide there), keeps three sets on x_2 with the transitions between the classes, and merges all sets of the noise feature into one, which marks it as a don't care variable; the Ruspini partition (bottom) places its sets at equal distances regardless of the classes. A decision tree of depth two on the crisp codes reads as three rules on linguistic terms. In cross-validation, unrestricted trees on the supervised codes reach 0.96 accuracy with five leaves, whereas equal-width, equal-frequency and Ruspini partitions need about fifteen leaves for 0.93, 0.91 and 0.79; the membership encoding reaches 0.996, and a tree on the raw features is perfect because the boundaries are axis-parallel thresholds that CART finds exactly. On iris (fig. 8.27) the supervised sets of the petal length switch between the species, and trees of depth three on the a priori partitions reach 0.91–0.98 accuracy against 0.93 for the raw tree: equal-width bins (0.98) and the supervised codes (0.94) do slightly better, equal-frequency bins, the Ruspini sets and the membership encoding (0.91–0.92) slightly worse, so on these well-separated species the benefit of the supervised sets is their meaning rather than accuracy. The scan in fig. 8.28 shows that three to five initial sets suffice, that two are too few, and that merging at $\theta \leq 0.8$ removes a fifth to a third of the sets in this range (six instead of nine for three initial sets) without loss of accuracy, whereas $\theta = 1$ keeps every set and grows larger trees (nine leaves instead of five for three initial sets).

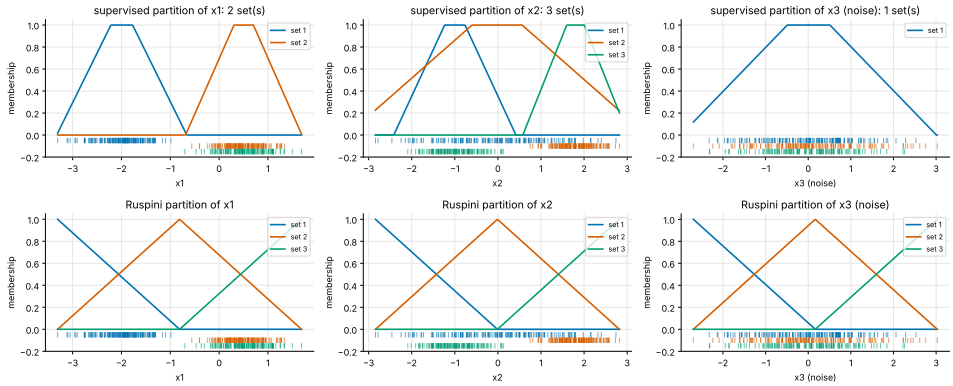


Figure 8.26. Supervised fuzzy partition (top) and Ruspini partition (bottom) of three features, with the class-conditional samples as rugs. The supervised sets switch where the classes change and collapse to one set on the noise feature.

8.4.7 Hinging-hyperplane tree on a piecewise-linear surface

Figure 8.29 shows the elementary step on one input: FCRM with two models fits the two lines of $y = \max(0.8x, -1.2x + 1)$, their crossing at $x = 0.49$ (true 0.5) is the hinge, and the memberships are the fuzzy version of the split; the condition of

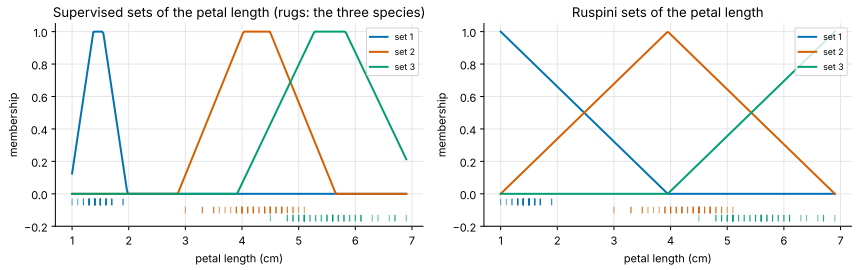


Figure 8.27. Supervised and Ruspini sets of the petal length of the iris data: the supervised sets switch between the species.

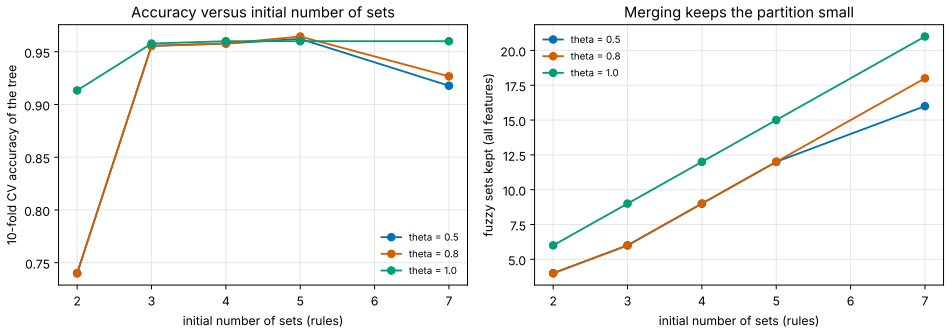


Figure 8.28. Cross-validated accuracy of the tree and total number of sets as functions of the initial number of sets, for three merging thresholds: three to five initial sets suffice.

the node is 0.999. On the two-input surface with two crossing creases (fig. 8.30) the root hinge of a depth-two tree splits the plane close to the main crease, the child hinges model the remaining kinks, and one child is split again; the root reports $\varrho = 1.00$ and the deepest node $\varrho = 0.66$, where the remaining error sits. The parameter study (fig. 8.31) shows the test RMSE falling from 0.30 (one hinge) to 0.06 with seven leaves, steeply for the first splits and slowly afterwards, because every hinge is fitted to data that are only approximately piecewise planar. On this surface greedy growth gives practically the same error as balanced growth for every number of leaves, a large `min_samples_leaf` stops the growth at four leaves, and linear leaves need more splits than hinge leaves (a depth-two tree with three planes has an RMSE of 0.26).

8.5 Further reading

The Takagi–Sugeno model was introduced in [94]; Babuška's book [7] and the chapter [95] are the standard references for product-space clustering, projection of clusters and the local versus global estimation of consequents, whose interpretation is analyzed in [102, 106]. The probabilistic interpretation of Gath–Geva clustering [21] as a Gaussian mixture of regressors connects it to cluster-weighted modeling

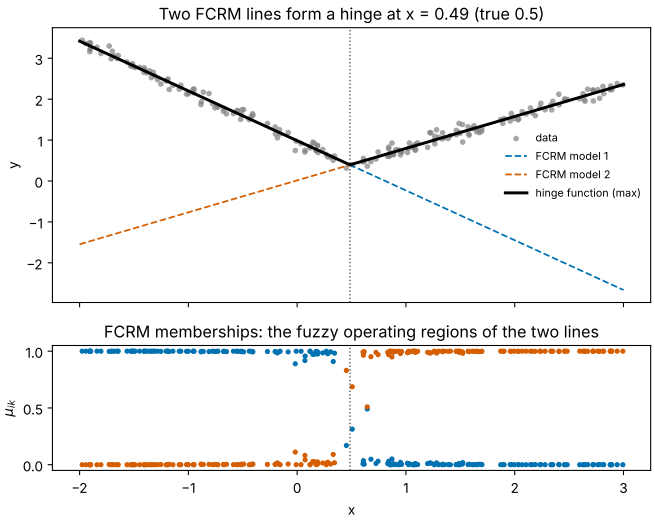


Figure 8.29. One hinge from FCRM with two models on one input: the two lines (dashed), the hinge function (black) and the memberships of the two models, which are the fuzzy operating regions.

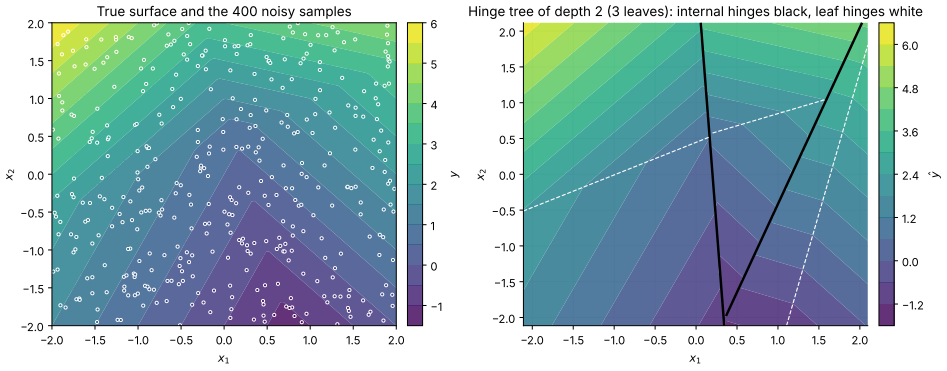


Figure 8.30. True piecewise-linear surface with the samples (left) and the prediction of a depth-two hinge tree with its hinges (right): the hinges of the internal nodes in black, those of the leaves dashed white.

[101]; the transformed input-domain approach [103] is the exact but opaque alternative to projection, and the modified Gath–Geva algorithm [96] the transparent one, extended to MIMO systems in [104, 105]. Gustafson–Kessel clustering was introduced in [18] and fuzzy c-regression models in [97]; the c-means functional with its convergence theory is Bezdek's [3]. The supervised fuzzy classifier and its feature selection are in [98], and the supervised partitions for decision trees in [99], which builds on the similarity-based rule-base simplification of [42], the fuzzy decision trees of Janikow [108], ID3 [107], CART [111] and the comparison of discretization methods in [109]. Hinging hyperplanes are Breiman's [110]; their

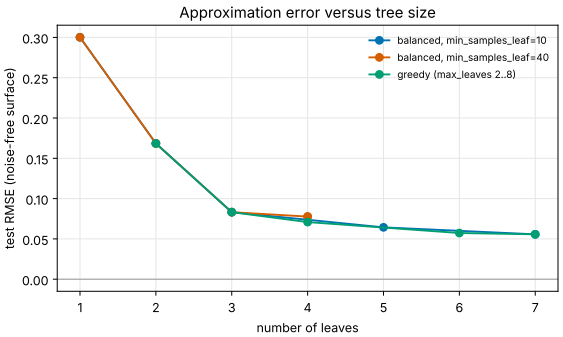


Figure 8.31. Test RMSE versus the number of leaves for balanced growth with two leaf sizes and for greedy growth. The greedy and balanced curves practically coincide; the large leaf size stops the growth at four leaves.

identification by constrained fuzzy c-regression, the tree construction and its use in predictive control are described in [100].

CHAPTER

9



Clustering of time series

The previous chapter turned clusters into local models of static and dynamic functions; this chapter adds the ingredient that ordered data bring, time. Four problems of time-series analysis are solved with one tool, a fuzzy clustering whose prototype is adapted to the question. For segmentation the prototype combines a Gaussian membership function in time with a local PCA model of the observations, and the number of segments follows from merging compatible neighbors; for model-order selection Gath–Geva clusters supply the threshold of the false-nearest-neighbor test as local slopes of the regression surface. Clustering the delay vectors of a chaotic series together with their successors gives the embedding dimension, the local dimension of the attractor and a predictor, and fuzzy c-means partitions let structure selection treat continuous inputs like categorical ones. All methods live in `fuzzyclust.timeseries`; the fuzzy Sammon mapping of chapter 5 displays the clusters of a reconstructed state space.

9.1 Problem and idea

A time series $T = \{\mathbf{x}_k \mid 1 \le k \le N\}$ is a sequence of observations $\mathbf{x}_k \in \mathbb{R}^n$ labeled by time points $t_1 < \cdots < t_N$. The observations are not exchangeable: their order carries information, and the questions asked of a time series are questions about that order. Four such questions recur in the analysis of process data, and each has a clustering-based answer in this chapter.

Segmentation. A c-segmentation splits T into c disjoint intervals that are internally homogeneous—stable periods of operation, product grades, regimes—separated by change points. For one variable the cost of a segment is the error of a constant or linear fit; for many correlated variables the right notion of homogeneity is that the samples lie on a common hyperplane, i.e. that one PCA model with a few components describes them. Changes of the mean, of the variance *and* of

the correlation structure then all become changes of the hyperplane, and the last kind is invisible to any univariate method. Minimizing the summed segment costs by dynamic programming is too expensive for long series, so greedy top-down, bottom-up and sliding-window heuristics are used instead [112, 113]. Abonyi, Feil, Németh and Árva [114] treat segmentation as clustering with a time-ordered structure (their conference paper [115], the precursor of the method, used the simpler Q and T^2 distances of section 9.2.1). They cluster the extended samples $\mathbf{z}_k = [t_k \ \mathbf{x}_k^\mathsf{T}]^\mathsf{T}$ with a prototype made of a Gaussian membership function in time, which keeps the cluster contiguous, and a probabilistic PCA model in the feature space, which measures homogeneity (fig. 9.1). The borders are fuzzy, as the transitions of real processes are, and the number of segments is found by merging compatible neighbors.

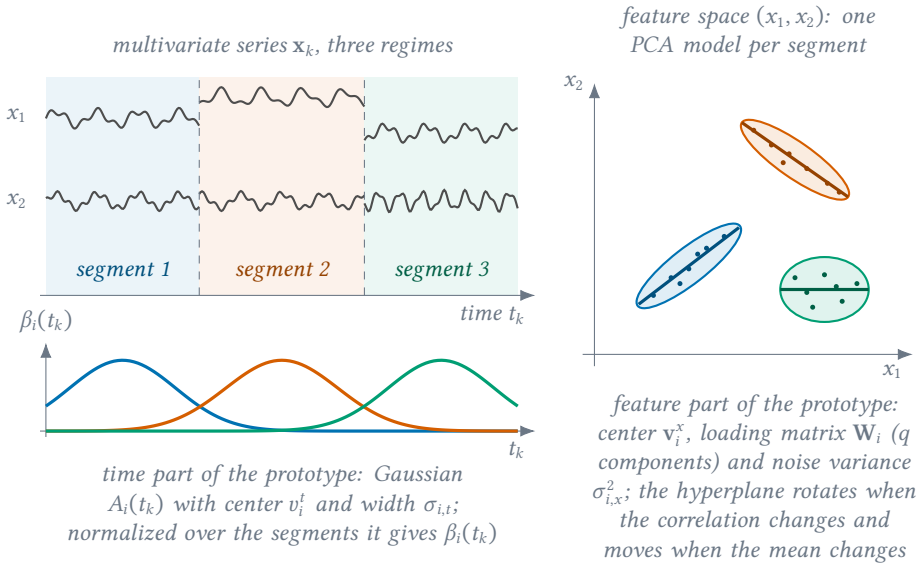
Model order. Before a nonlinear dynamic model $y_k = f(\mathbf{x}_k)$ can be identified, the regression vector $\mathbf{x}_k = [y_{k-1}, \dots, y_{k-n_a}, u_{k-1}, \dots, u_{k-n_b}]^\mathsf{T}$ has to be chosen. Information criteria (AIC, FPE, MDL) presuppose a model class; the false-nearest-neighbor (FNN) test [116, 117, 118] does not: it checks whether samples that are close in the regression space have close outputs. Its weak point is a threshold that depends on the slope of the unknown function, and Feil, Abonyi and Szeifert [119, 120] estimate that slope locally from Gath–Geva clusters, which are local linear models of the regression surface.

State-space reconstruction. The state space of a chaotic system observed through one variable y_k can be reconstructed from the delay vectors $\mathbf{y}_k = [y_k, y_{k-\tau}, \dots, y_{k-\tau(d_e-1)}]^\mathsf{T}$ [121, 122]. Choosing the embedding dimension d_e, estimating the intrinsic dimension of the attractor and building a predictor are usually three separate tasks; the clustering method of [123] solves them together, because the clusters of the pairs $(\mathbf{y}_k, \mathbf{y}_{k+1})$ are local linear models of the map whose prediction error decides d_e and whose flatness reveals the local dimension.

Structure selection. A nonlinear model often has many candidate inputs, some continuous and some categorical, of which only a few are needed. Girimonte, Babuška and Abonyi [124] rank all small subsets of inputs by the variance of the output inside the segments spanned by the values of the inputs; fuzzy c-means partitions turn continuous inputs into fuzzy "values", so that no model has to be fitted and the two kinds of input are treated alike.

9.2 Theory

The notation is that of the book: $\mathbf{X} \in \mathbb{R}^{N \times n}$ with rows $\mathbf{x}_k$, memberships μ_{ik} in the matrix $\mathbf{U} \in \mathbb{R}^{c \times N}$ (stored transposed in the code), prototypes with index i, samples with index k. Time stamps are t_k, the extended sample is $\mathbf{z}_k = [t_k \ \mathbf{x}_k^\mathsf{T}]^\mathsf{T}$, delay vectors are $\mathbf{y}_k$, and η_i denotes cluster i together with the parameters of its prototype. We treat segmentation first (sections 9.2.1 and 9.2.2), then FNN-based model-order



$$\frac{1}{D^2(\mathbf{z}_k, \eta_i)} = \underbrace{\alpha_i}_{p(\eta_i)} \underbrace{\frac{e^{-\frac{1}{2}(t_k - v_i^t)^2/\sigma_{i,t}^2}}{\sqrt{2\pi\sigma_{i,t}^2}}}_{p(t_k|\eta_i)} \underbrace{\frac{e^{-\frac{1}{2}(\mathbf{x}_k - \mathbf{v}_i^x)^\mathsf{T}\mathbf{A}_i^{-1}(\mathbf{x}_k - \mathbf{v}_i^x)}}{(2\pi)^{n/2}\sqrt{\det \mathbf{A}_i}}}_{p(\mathbf{x}_k|\eta_i)}$$

a sample belongs to a segment if it is observed *at the right time* and lies *on the right hyperplane*

Figure 9.1. Fuzzy segmentation as clustering with a time-aware prototype. Left: three regimes of a bivariate series; each differs from the previous one in the mean of x_1 and in the correlation of the two variables (reversed in the second regime, absent in the third), and the change of correlation is visible in neither variable alone. Below them: the Gaussian membership functions in time, normalized to $\beta_i(t_k)$, are the fuzzy segments. Right: every segment carries a probabilistic PCA model in the feature space whose hyperplane reacts to changes of mean, variance and correlation. Bottom: the distance of the clustering is the inverse of the joint probability density of the time stamp and the observation.

selection (section 9.2.3), the reconstruction of chaotic series (section 9.2.4) and structure selection (section 9.2.5).

9.2.1 Fuzzy segmentation by modified Gath–Geva clustering

A crisp c-segmentation with the variance as cost minimizes

$$\begin{aligned} \operatorname{cost}(S_T^c) &= \sum_{i=1}^{c} \sum_{k=s_{i-1}+1}^{s_i} \|\mathbf{x}_k - \mathbf{v}_i^x\|^2 = \sum_{i=1}^{c} \sum_{k=1}^{N} \beta_i(t_k)\, D^2(\mathbf{x}_k, \mathbf{v}_i^x), \\ \beta_i(t_k) &= \begin{cases} 1 & s_{i-1} < k \le s_i \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \tag{9.1}$$

which is the k-means functional with clusters restricted to time intervals instead of Voronoi cells. Fuzzy segments replace the indicator by normalized Gaussian membership functions in time,

$$A_i(t_k) = \exp\Big(-\frac{1}{2} \frac{(t_k - v_i^t)^2}{\sigma_{i,t}^2} \Big), \qquad \beta_i(t_k) = \frac{A_i(t_k)}{\sum_{j=1}^{c} A_j(t_k)}, \tag{9.2}$$

whose centers v_i^t and widths $\sigma_{i,t}$ are identified by the clustering. The algorithm minimizes the usual functional $J = \sum_i \sum_k \mu_{ik}^m D^2(\mathbf{z}_k, \eta_i)$ under the partition constraints, with a Gath–Geva-type distance that is the inverse of a probability density (fig. 9.1, bottom):

$$\begin{aligned} \frac{1}{D^2(\mathbf{z}_k, \eta_i)} = \alpha_i \, & \frac{1}{\sqrt{2\pi\sigma_{i,t}^2}} \exp\Big(-\frac{1}{2} \frac{(t_k - v_i^t)^2}{\sigma_{i,t}^2} \Big) \\ & \times \frac{1}{(2\pi)^{r/2} \sqrt{\det \mathbf{A}_i}} \exp\Big(-\frac{1}{2} (\mathbf{x}_k - \mathbf{v}_i^x)^{\mathsf{T}} \mathbf{A}_i^{-1} (\mathbf{x}_k - \mathbf{v}_i^x) \Big). \end{aligned} \tag{9.3}$$

Within a segment, time is treated as independent of the observations, so the density factorizes into the prior α_i, a Gaussian in time and a Gaussian in the feature space with covariance $\mathbf{A}_i$ of rank r. For $m = 2$ the memberships are the posterior probabilities $p(\eta_i \mid \mathbf{z}_k)$ of this mixture, and the alternating optimization is a fuzzy relative of the EM identification of the mixture, which would weight the prototype updates by μ_{ik} instead of μ_{ik}^m [21, 96].

The feature-space model. Choosing $\mathbf{A}_i = \mathbf{F}_i$, the fuzzy covariance matrix, gives Gath–Geva clustering of the extended samples; when the variables are strongly correlated, $\mathbf{F}_i$ is ill-conditioned, and regularizing it [19] hides exactly the low-dimensional structure one wants to detect. The journal version of the method therefore uses *probabilistic PCA* [125]: the observations of a segment are $\mathbf{x}_k = \mathbf{W}_i \mathbf{y}_{ik} + \mathbf{v}_i^x + \boldsymbol{\varepsilon}$ with q latent variables and isotropic noise, so that

$$\mathbf{A}_i = \sigma_{i,x}^2 \mathbf{I} + \mathbf{W}_i \mathbf{W}_i^{\mathsf{T}}, \tag{9.4}$$

a full-rank covariance ($r = n$) whose q large eigen-directions span the hyperplane of the segment and whose remaining $n-q$ eigenvalues all equal $\sigma_{i,x}^2$. The maximum-

likelihood solution is known in closed form: with $\mathbf{F}_i = \mathbf{U}_i \mathbf{\Lambda}_i \mathbf{U}_i^\mathsf{T}$ (eigenvalues in decreasing order),

$$\mathbf{W}_i = \mathbf{U}_{i,q} \left(\mathbf{\Lambda}_{i,q} - \sigma_{i,x}^2 \mathbf{I} \right)^{1/2} \mathbf{R}_i, \qquad \sigma_{i,x}^2 = \frac{1}{n-q} \sum_{j=q+1}^{n} \lambda_{i,j}, \tag{9.5}$$

with an arbitrary rotation $\mathbf{R}_i$. The journal version instead interleaves one EM step of PPCA with every clustering iteration: with $\mathbf{M}_i = \sigma_{i,x}^2 \mathbf{I} + \mathbf{W}_i^\mathsf{T} \mathbf{W}_i$ and the latent expectations $\langle \mathbf{y}_{ik} \rangle = \mathbf{M}_i^{-1} \mathbf{W}_i^\mathsf{T} (\mathbf{x}_k - \mathbf{v}_i^x)$,

$$\tilde{\mathbf{W}}_i = \mathbf{F}_i \mathbf{W}_i \left(\sigma_{i,x}^2 \mathbf{I} + \mathbf{M}_i^{-1} \mathbf{W}_i^\mathsf{T} \mathbf{F}_i \mathbf{W}_i \right)^{-1}, \quad \sigma_{i,x}^2 = \frac{1}{n} \operatorname{tr} \left(\mathbf{F}_i - \mathbf{F}_i \mathbf{W}_i \mathbf{M}_i^{-1} \tilde{\mathbf{W}}_i^\mathsf{T} \right), \tag{9.6}$$

where the center $\mathbf{v}_i^x = \sum_k \mu_{ik}^m (\mathbf{x}_k - \mathbf{W}_i \langle \mathbf{y}_{ik} \rangle) / \sum_k \mu_{ik}^m$ and the fuzzy covariance $\mathbf{F}_i$ around it are computed with the weights μ_{ik}^m. Both routes have the same fixed point; `fuzzyclust` offers the EM step (`pca_update="em"`) and the eigen-solution (`"eig"`). The noise variance in (9.6) carries the factor $1/n$ of the EM algorithm of Tipping and Bishop [125]; the segmentation paper prints $1/q$ there, which the book and the code correct. The prior and the time part are updated by

$$\alpha_i = \frac{1}{N} \sum_{k=1}^{N} \mu_{ik}, \qquad v_i^t = \frac{\sum_k \mu_{ik}^m t_k}{\sum_k \mu_{ik}^m}, \qquad \sigma_{i,t}^2 = \frac{\sum_k \mu_{ik}^m (t_k - v_i^t)^2}{\sum_k \mu_{ik}^m}, \tag{9.7}$$

and the partition by $\mu_{ik} = 1 / \sum_j (D_{ik}/D_{jk})^{2/(m-1)}$.

Two older variants. The conference version [115] restricted the feature-space Gaussian to a subspace of the PCA model and used a pseudo-inverse and pseudo-determinant in (9.3). The *fuzzy-PCA-Q* method keeps the $n - q$ smallest eigen-directions, so the exponent is the reconstruction error $Q_{ik} = \left\| (\mathbf{I} - \mathbf{U}_{i,q} \mathbf{U}_{i,q}^\mathsf{T})(\mathbf{x}_k - \mathbf{v}_i^x) \right\|^2$ with every residual direction divided by its eigenvalue: it detects changes of the correlation structure but is blind to displacements inside the hyperplane. The *fuzzy-PCA-T^2* method keeps the q largest, so the exponent is the Hotelling statistic $T_{ik}^2 = \mathbf{y}_{ik}^\mathsf{T} \mathbf{\Lambda}_{i,q}^{-1} \mathbf{y}_{ik}$ of the projection: it detects shifts of the operating point but not rotations of the hyperplane. Both are sensitive to q (with $q \to n$ the reconstruction error vanishes everywhere; the T^2 variant needs a q defined by a required accuracy so that all segments keep comparable models). The PPCA covariance (9.4) covers both subspaces and is insensitive to q; the two statistics remain available as diagnostics (`hotelling_t2`, `q_reconstruction_error`).

Number of components. The eigenvalues of the covariance matrices of the *initial* segments—a uniform partition into c intervals—are inspected in a scree plot, or q is chosen as the smallest number of components that explains a required fraction of the variance in every initial segment,

$$\sum_{j=1}^{q-1} \lambda_{i,j} \Big/ \sum_{j=1}^{n} \lambda_{i,j} < \text{accuracy} \le \sum_{j=1}^{q} \lambda_{i,j} \Big/ \sum_{j=1}^{n} \lambda_{i,j}. \tag{9.8}$$

Number of segments by compatible cluster merging. Clustering starts from a deliberately large number of segments (ten in the paper) and merges adjacent segments that are compatible, re-clustering after every merge, in the spirit of the bottom-up algorithm and of compatible cluster merging for fuzzy modeling [40]. Two criteria are evaluated for every pair: the similarity of the PCA models by Krzanowski's factor [126],

$$S_{PCA}^{ij} = \frac{1}{q}\sum_{a=1}^{q}\sum_{b=1}^{q}\cos^2\theta_{ab} = \frac{1}{q}\operatorname{tr}\left(\mathbf{U}_{i,q}^{\mathsf{T}}\mathbf{U}_{j,q}\mathbf{U}_{j,q}^{\mathsf{T}}\mathbf{U}_{i,q}\right), \tag{9.9}$$

the mean squared cosine between the principal directions of one segment and the principal subspace of the other, and the distance of the centers $D(\mathbf{v}_i^x, \mathbf{v}_j^x) = \left\|\mathbf{v}_i^x - \mathbf{v}_j^x\right\|$, because a change of the mean alone leaves $S_{PCA} = 1$. Each criterion is turned into a degree of fulfillment by a triangular membership function: *parallelism* μ_{ij}^1 rises from 0 at v^1 to 1 at $S_{PCA} = 1$, and *closeness* μ_{ij}^2 falls from 1 at distance 0 to 0 at v^2. The knots are the averages of the criteria over all pairs,

$$v^1 = \frac{1}{c(c-1)}\sum_i\sum_{j\neq i} S_{PCA}^{ij}, \qquad v^2 = \frac{1}{c(c-1)}\sum_i\sum_{j\neq i} D(\mathbf{v}_i^x, \mathbf{v}_j^x), \tag{9.10}$$

and the two degrees are aggregated to the compatibility

$$O_{ij} = \left[\tfrac{1}{2}\left((\mu_{ij}^1)^2 + (\mu_{ij}^2)^2\right)\right]^{1/2}. \tag{9.11}$$

The most compatible *adjacent* pair is merged whenever $O_{i,i+1}$ exceeds a threshold γ (0.3–0.75 in the paper, depending on the homogeneity of the series). Kelly's formulas [41] give the parameters of the merged cluster from those of its parts, with $N_i = \sum_k \mu_{ik}$ and $N_{i*} = N_i + N_{i+1}$:

$$\begin{aligned}
\mathbf{v}_{i*}^x &= \frac{N_i}{N_{i*}}\mathbf{v}_i^x + \frac{N_{i+1}}{N_{i*}}\mathbf{v}_{i+1}^x,\\
\mathbf{F}_{i*} &= \frac{N_i - 1}{N_{i*} - 1}\mathbf{F}_i + \frac{N_{i+1} - 1}{N_{i*} - 1}\mathbf{F}_{i+1} + \frac{N_i N_{i+1}}{N_{i*}(N_{i*} - 1)}(\mathbf{v}_i^x - \mathbf{v}_{i+1}^x)(\mathbf{v}_i^x - \mathbf{v}_{i+1}^x)^{\mathsf{T}},
\end{aligned} \tag{9.12}$$

and analogous formulas merge the time center and variance; the priors add, $\alpha_{i*} = \alpha_i + \alpha_{i+1}$, and the PPCA model of the new segment follows from the eigendecomposition of $\mathbf{F}_{i*}$ by (9.5). Two segments with identical PCA models but different means have $\mu^1 = 1$ and $O = \sqrt{(1 + (\mu^2)^2)/2} \geq 1/\sqrt{2} \approx 0.71$, so the threshold has to be raised above the compatibility of the smallest mean shift that should survive; this is how γ encodes how different two regimes must be. The compatibility matrix of all pairs, adjacent or not, is informative in itself: repeated regimes show up as compatible nonadjacent segments. (`fuzzyclust` adds a safeguard, `min_segment_weight`, that absorbs a segment with a negligible prior into its more compatible neighbor regardless of γ; such slivers arise between regimes when the initial number of segments is far above the true one.)

Algorithm 9.1 Fuzzy segmentation by modified Gath–Geva clustering

Input: series $\mathbf{X}$ with time stamps t_k, initial number of segments c, components q (or an accuracy), exponent m, threshold γ, tolerance ϵ

partition the series uniformly into c intervals (crisp $\mathbf{U}^{(0)}$); choose q from the scree plot or by (9.8);
repeat
  for *every segment* i **do**
    prior, time center and width by (9.7);
    center $\mathbf{v}_i^x$ with the latent expectations, fuzzy covariance $\mathbf{F}_i$;
    PPCA step (9.6) (or the eigen-solution (9.5));
    $\mathbf{A}_i = \sigma_{i,x}^2 \mathbf{I} + \tilde{\mathbf{W}}_i \tilde{\mathbf{W}}_i^{\mathsf{T}}$;
  distances $D^2(\mathbf{z}_k, \eta_i)$ by (9.3); partition $\mu_{ik} = 1/\sum_j (D_{ik}/D_{jk})^{2/(m-1)}$;
  if *the tolerance is reached or a fixed number of iterations has passed* **then**
    compatibilities $O_{i,i+1}$ by (9.9)–(9.11);
    if $\max_i O_{i,i+1} > \gamma$ **then** merge that pair by (9.12), $c \leftarrow c - 1$, recompute $\mathbf{U}$;
until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$ and *no merge was made*;

Output: fuzzy segments $\beta_i(t_k)$, posteriors $\mu_{ik} = p(\eta_i \mid \mathbf{z}_k)$, borders where the dominant β_i changes, one PCA model per segment

In a nutshell Fuzzy segmentation

- Segmentation is clustering of $\mathbf{z}_k = [t_k \ \mathbf{x}_k^{\mathsf{T}}]^{\mathsf{T}}$ with a prototype made of a Gaussian in time (contiguity) and a probabilistic PCA model in the feature space (homogeneity); the distance is the inverse joint density.
- Detects changes of the mean, the variance and the correlation structure of multivariate series; for $m = 2$ the memberships are the posteriors of a mixture of PPCA models, and the alternating optimization is a fuzzy relative of its EM algorithm.
- q from the scree plot or an accuracy on the initial segments; the number of segments by merging compatible adjacent segments (PCA similarity and center distance, threshold γ) from a large initial number.
- Fuzzy borders, posterior memberships and a PCA model per segment for process monitoring; Q and T^2 remain available as diagnostics.

Algorithm 9.2 Bottom-up segmentation with PCA models

Input: series $\mathbf{X}$, number of components p, cost (Q or T^2), stopping rule (maximal error or number of segments), initial piece length

cut the series into pieces of the initial length;
mergecost(i) = cost$(S(a_i, b_{i+1}))$ for all adjacent pairs;
while $\min_i$ mergecost$(i) <$ maxerror *(or more segments than wanted)* **do**
  $i = \arg\min_i$ mergecost(i); merge S_i and S_{i+1};
  recompute mergecost(i) = cost$(S(a_i, b_{i+1}))$ and mergecost$(i-1)$ = cost$(S(a_{i-1}, b_i))$;

Output: crisp segments, their borders and the sequence of merge costs

9.2.2 Bottom-up segmentation with PCA models

The classical competitor is the bottom-up algorithm [112], extended to multivariate data by using one PCA model per segment [114]. The series is cut into short pieces; for every pair of adjacent segments the cost of their union is computed; the cheapest pair is merged, and its merge costs with its two neighbors are updated; the loop stops when the cheapest merge would cost more than a tolerance or when the wanted number of segments is reached. For a segment $S_i = \{a_i, \dots, b_i\}$ and a PCA model with p components, the cost is either the mean reconstruction error or the mean Hotelling statistic,

$$\begin{aligned} \text{cost}_Q(S_i) &= \frac{1}{b_i - a_i + 1} \sum_{k=a_i}^{b_i} Q_{i,k} = \sum_{j>p} \lambda_{i,j}, \\ \text{cost}_{T^2}(S_i) &= \frac{1}{b_i - a_i + 1} \sum_{k=a_i}^{b_i} T^2_{i,k} = \sum_{j\le p} \lambda^2_{i,j}, \end{aligned} \tag{9.13}$$

where the right-hand sides hold for centered data, with T^2 computed as in the paper from the scores $\mathbf{y}_k = \mathbf{\Lambda}_p^{1/2}\mathbf{U}_p^\mathsf{T}\mathbf{x}_k$. The cost measures the level of the model error of the merged segment, not its increase, so the natural stopping rule is a maximal error. Each cost is blind to one kind of change, Q to displacements along the hyperplane and T^2 to rotations of it, and a greedy merge made early is never revised. This is the baseline against which the fuzzy segmentation is judged.

9.2.3 Model order by false nearest neighbors with cluster-based thresholds

Let $y_k = f(\mathbf{x}_k)$ with the NARX regression vector $\mathbf{x}_k = [y_{k-1}, \dots, y_{k-n_a}, u_{k-1}, \dots, u_{k-n_b}]^\mathsf{T}$. If the regressors carry all the information about y_k and f is smooth, close

In a nutshell Bottom-up segmentation

- Greedy merging of adjacent pieces by the PCA cost of the union; fast and simple, the multivariate counterpart of piecewise-linear approximation.
- The Q cost finds changes of the correlation structure, the T^2 cost shifts of the operating point; neither finds both, and the Q cost depends strongly on p.
- Stop by a maximal error (the jump of the merge-cost curve) or by a given number of segments; useful as a first look and to validate the fuzzy result.

regression vectors have close outputs: by a first-order Taylor expansion and the Cauchy–Schwarz inequality,

$$y_k - y_j = \mathrm{d}f(\mathbf{x}_k)^\mathsf{T}(\mathbf{x}_k - \mathbf{x}_j) + O(\|\mathbf{x}_k - \mathbf{x}_j\|^2) \quad \Longrightarrow \quad \frac{|y_k - y_j|}{\|\mathbf{x}_k - \mathbf{x}_j\|} \lesssim \|\mathrm{d}f(\mathbf{x}_k)\|, \tag{9.14}$$

where $\mathrm{d}f$ is the gradient (Jacobian) of f; the bound holds up to a term of order $\|\mathbf{x}_k - \mathbf{x}_j\|$, that is, approximately for close neighbors. The FNN algorithm [116, 117] finds for every $\mathbf{x}_k$ its nearest neighbor $\mathbf{x}_j$ in the regression space, declares the neighbor *false* when the ratio exceeds a threshold R, and tabulates the percentage $f(n_a, n_b)$ of false neighbors over the orders. The percentage is 100 for the empty regressor, stays high while a regressor is missing (points that are close in the incomplete space have different outputs because they differ in the missing coordinate; see fig. 9.2), and drops to zero, or to a small, flat level when the data are noisy, as soon as the regression vector is complete. The smallest order in the corner of the table where this happens is the model order. For a scalar time series (u absent, regressors $y_{k-\tau}, \ldots, y_{k-n_a\tau}$) the same test gives the embedding dimension, the problem for which it was introduced [116].

The difficulty is the threshold. Heuristic values ($10 \le R \le 50$, after Kennel et al. [116]) work for some systems and not for others [118], and the exact bound $R = \max_k \|\mathrm{d}f(\mathbf{x}_k)\|$ requires a nonlinear model of the unknown function [117]—the very thing the order selection is supposed to precede. Feil, Abonyi and Szeifert [119, 120] observe that when the regression vector is adequate, Gath–Geva clusters of the joint data $\mathbf{z}_k = [\mathbf{x}_k^\mathsf{T}\ y_k]^\mathsf{T}$ approximate the regression surface piecewise linearly (fig. 9.2, bottom): the smallest eigenvalue of $\mathbf{F}_i$ is much smaller than the others and its eigenvector $\mathbf{t}_i = [(\mathbf{t}_i^x)^\mathsf{T}\ t_i^y]^\mathsf{T}$ is the normal of the local hyperplane, so that

$$(\mathbf{t}_i)^\mathsf{T}(\mathbf{z}_k - \mathbf{v}_i) = 0 \quad \Longrightarrow \quad y_k = \underbrace{\frac{-1}{t_i^y}(\mathbf{t}_i^x)^\mathsf{T}}_{\mathbf{a}_i^\mathsf{T}}\mathbf{x}_k + \underbrace{\frac{1}{t_i^y}\mathbf{t}_i^\mathsf{T}\mathbf{v}_i}_{b_i}, \tag{9.15}$$

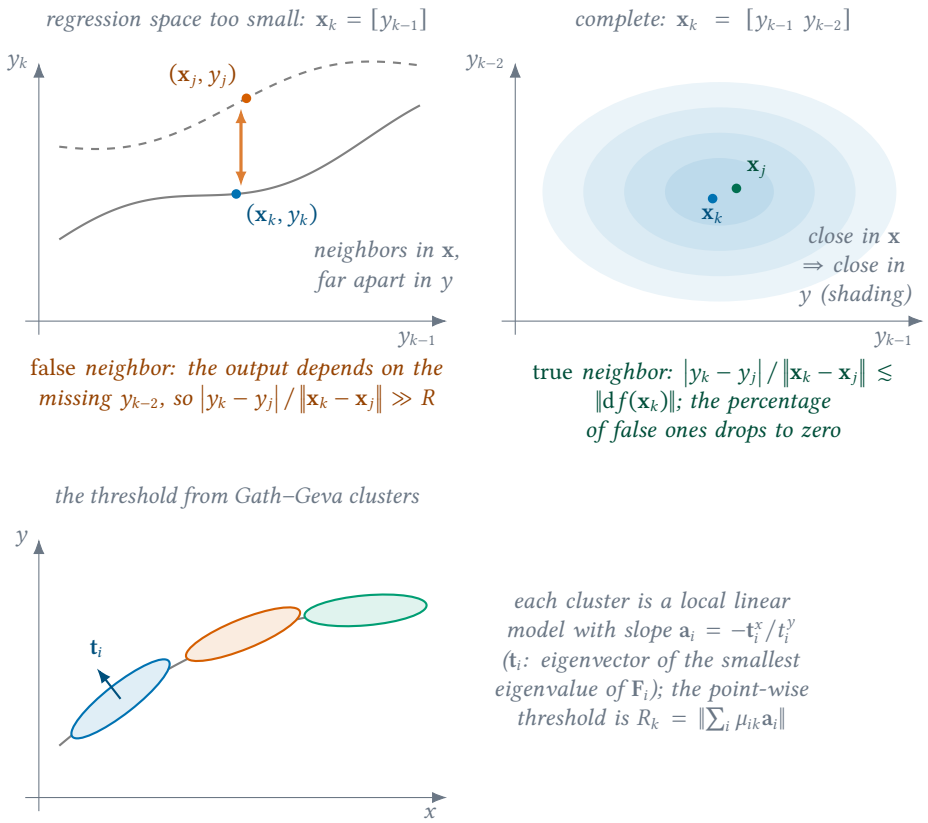


Figure 9.2. The false-nearest-neighbor idea and its clustering-based threshold. Top left: with a regressor missing, the output is not a function of the regression vector, so points that are neighbors in the regression space can have very different outputs. Top right: with the complete regression vector the output is a smooth function and the ratio $|y_k - y_j| / \|\mathbf{x}_k - \mathbf{x}_j\|$ is bounded by the local gradient. Bottom: Gath–Geva clusters of the regression data are flat ellipsoids whose normal, the eigenvector of the smallest eigenvalue, gives the local slope, and the membership-weighted slopes give a threshold that follows the nonlinearity.

the total-least-squares estimate of a local linear model. The gradient at $\mathbf{x}_k$ is approximated by the membership-weighted average of the local slopes, which gives a *point-wise* threshold,

$$\mathrm{d}f(\mathbf{x}_k) \approx \sum_{i=1}^{c} \mu_{ik}\, \mathbf{a}_i, \qquad R_k = \Big\| \sum_{i=1}^{c} \mu_{ik} \frac{-1}{t_i^y} \mathbf{t}_i^x \Big\|_2, \tag{9.16}$$

so that neighbor j of k is false when $|y_k - y_j| / \|\mathbf{x}_k - \mathbf{x}_j\| > R_k$. The threshold adapts to the nonlinearity, large where the surface is steep and small where it is flat, and the only remaining parameter, the number of clusters, has little influence. Gath–Geva clustering, initialized by fuzzy c-means, is preferred to Gustafson–Kessel because the clusters may have different volumes. Regressors and output are standardized first so that the ratio and R_k are dimensionless.

Algorithm 9.3 Model-order selection by FNN with a cluster-based threshold

Input: output y and input u sequences, maximal orders, number of clusters c, tolerance on the percentage of false neighbors

for *every order* (n_a, n_b) *with* $n_a + n_b > 0$ **do**
- build the regression vectors $\mathbf{x}_k$ and outputs y_k; standardize;
- cluster $\mathbf{z}_k = [\mathbf{x}_k^\mathsf{T}\ y_k]^\mathsf{T}$ with c Gath–Geva clusters (FCM initialization);
- local slopes $\mathbf{a}_i$ from the eigenvectors of the smallest eigenvalues by (9.15); thresholds R_k by (9.16);
- **for** *every sample* k **do**
  - nearest neighbor j in the regression space; false if $|y_k - y_j| \,/\, \|\mathbf{x}_k - \mathbf{x}_j\| > R_k$
- $J(n_a, n_b)$ = percentage of false neighbors;

Output: table $J(n_a, n_b)$; the selected order is the smallest $n_a + n_b$ with J not above the tolerance

In a nutshell FNN model-order selection

- Geometric, model-free test: a regression vector is complete when neighbors in the regression space have neighboring outputs.
- The threshold of the test is the local gradient norm; Gath–Geva clusters of the regression data supply it point-wise as the norm of the membership-weighted local slopes.
- Read the order off the corner of the $J(n_a, n_b)$ table where the percentage collapses; the number of clusters hardly matters, a constant threshold does.
- With u absent and a lag τ the same function estimates the embedding dimension of a scalar series.

9.2.4 State-space reconstruction and prediction of chaotic series

Let the state $\mathbf{s}_k \in M$ of a deterministic system evolve as $\mathbf{s}_{k+1} = \mathbf{f}(\mathbf{s}_k)$ and be observed through a smooth scalar function $y_k = g(\mathbf{s}_k)$. Takens' theorem [121] states that, if M is a compact manifold of dimension d_M, then for a generic observation the delay vectors

$$\mathbf{y}_k = [y_k,\ y_{k-\tau},\ \ldots,\ y_{k-\tau(d_e-1)}]^\mathsf{T} \tag{9.17}$$

embed M when $d_e \geq 2d_M + 1$. Sauer, Yorke and Casdagli [122] replaced the manifold by the attractor itself: if its box-counting dimension is d_A, almost every delay map is one-to-one on the attractor when $d_e > 2d_A$. In practice $d_e \geq d_l + 1$

often suffices, where d_l is the local dimension of the attractor, the dimension of the surface on which the trajectory lies. The lag τ is taken as the first zero of the autocorrelation or the first minimum of the mutual information [127]. In the reconstructed space the dynamics is a smooth map $\mathbf{y}_{k+1} = \mathbf{f}_r(\mathbf{y}_k)$, and since the data are typically confined to a d_l-dimensional surface of the d_e-dimensional space, local linear models [128, 129] are the natural approximators (fig. 9.3).

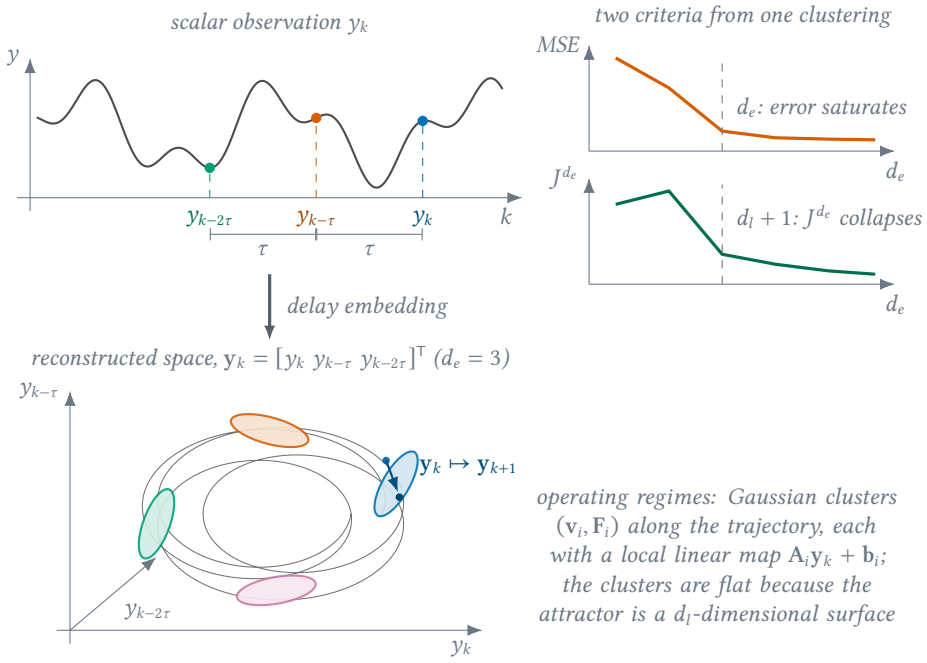


Figure 9.3. State-space reconstruction by delay embedding and clustering. Top left: three samples of the scalar series spaced by the lag τ form one delay vector. Bottom: the delay vectors trace the attractor in the reconstructed space; the clusters of the pairs $(\mathbf{y}_k, \mathbf{y}_{k+1})$ are operating regimes with local linear maps, flat because the trajectory lives on a low-dimensional surface. Top right: the one-step prediction error of the model saturates at the embedding dimension, and the prior-weighted smallest eigenvalue of the cluster covariances collapses one dimension above the local dimension.

The operating-regime model of [123] is the MIMO Takagi–Sugeno model of chapter 8 on the delay vectors,

$$\begin{aligned} \hat{\mathbf{y}}_{k+1} &= \sum_{i=1}^{c} \beta_i(\mathbf{y}_k)\big(\mathbf{A}_i\mathbf{y}_k + \mathbf{b}_i\big) = \sum_{i=1}^{c} \beta_i(\mathbf{y}_k)[\mathbf{y}_k^\mathsf{T}\ 1]\boldsymbol{\theta}_i, \\ \beta_i(\mathbf{y}) &= \frac{\exp\big(-\tfrac{1}{2}(\mathbf{y}-\mathbf{v}_i)^\mathsf{T}\mathbf{F}_i^{-1}(\mathbf{y}-\mathbf{v}_i)\big)}{\sum_{j=1}^{c}\exp\big(-\tfrac{1}{2}(\mathbf{y}-\mathbf{v}_j)^\mathsf{T}\mathbf{F}_j^{-1}(\mathbf{y}-\mathbf{v}_j)\big)}, \end{aligned} \tag{9.18}$$

whose Gaussian regimes and local models are identified simultaneously by clustering the pairs $(\mathbf{y}_k, \mathbf{y}_{k+1})$ with the distance

$$\frac{1}{D_{ik}^2} = p(\eta_i)\,\frac{\exp\!\big(-\tfrac{1}{2}(\mathbf{y}_k-\mathbf{v}_i)^\mathsf{T}\mathbf{F}_i^{-1}(\mathbf{y}_k-\mathbf{v}_i)\big)}{(2\pi)^{d_e/2}\sqrt{\det\mathbf{F}_i}}\;\frac{\exp\!\big(-\tfrac{1}{2}\mathbf{e}_{ik}^\mathsf{T}\mathbf{P}_i^{-1}\mathbf{e}_{ik}\big)}{(2\pi)^{p/2}\sqrt{\det\mathbf{P}_i}}, \qquad \mathbf{e}_{ik} = \mathbf{y}_{k+1} - [\mathbf{y}_k^\mathsf{T}\;1]\boldsymbol{\theta}_i, \tag{9.19}$$

the product of the prior, the Gaussian domain of influence and the Gaussian likelihood of the p-dimensional local prediction error with covariance $\mathbf{P}_i$. The alternating optimization updates $\mathbf{v}_i$ and $\mathbf{F}_i$ as the μ_{ik}^m-weighted mean and covariance of the delay vectors, the local models by weighted least squares $\boldsymbol{\theta}_i = (\mathbf{Y}^\mathsf{T}\mathbf{B}_i\mathbf{Y})^{-1}\mathbf{Y}^\mathsf{T}\mathbf{B}_i\mathbf{Y}_{(+1)}$ with $\mathbf{B}_i = \operatorname{diag}(\mu_{i1},\dots,\mu_{iN})$ (local estimation, which keeps the interpretation as a linear parameter-varying system), the error covariances $\mathbf{P}_i = \sum_k \mu_{ik}^m \mathbf{e}_{ik}\mathbf{e}_{ik}^\mathsf{T} / \sum_k \mu_{ik}^m$, the priors $p(\eta_i) = \frac{1}{N}\sum_k \mu_{ik}$ and the partition by the usual formula. Since all but the first component of $\mathbf{y}_{k+1}$ are past samples, which a free-run simulation simply shifts along, `fuzzyclust` by default identifies only the map to the next sample y_{k+1} ($p = 1$, `target="scalar"`). The full MIMO map is available as `target="full"`; with $\tau = 1$ its components copied from $\mathbf{y}_k$ make $\mathbf{P}_i$ singular unless it is regularized.

Three results come out of this clustering. (i) *Embedding dimension*: the model is identified for $d_e = 1, 2, \dots$ and the one-step prediction errors on test data are compared; the error decreases as long as a coordinate is missing and stops improving at the right d_e, where the ratio of consecutive errors approaches one. (ii) *Local dimension*: the clusters approximate the tangent space of the attractor, so their smallest eigenvalues $\lambda_{i,\min}$ are orders of magnitude below the others once d_e exceeds the dimension of the surface the trajectory lives on; the prior-weighted index

$$J^{d_e} = \sum_{i=1}^{c} p(\eta_i)\,\lambda_{i,\min} \tag{9.20}$$

drops sharply at $d_e = d_l + 1$. The integer d_l is complemented by the fractal correlation dimension of Grassberger and Procaccia [130], the slope of $\log C(r)$ against $\log r$ for the correlation sum $C(r)$ of the reconstructed vectors, which the fuzzy Sammon paper [51] uses to confirm the clustering-based estimate. (iii) *Prediction*: the model predicts one step ahead or, fed with its own outputs, simulates the series in free run; because the system is chaotic, the simulated trajectory leaves the measured one after a horizon set by the largest Lyapunov exponent, but it stays on (a model of) the attractor. The regimes of the reconstructed space are d_e-dimensional ellipsoids with Mahalanobis geometry; the fuzzy Sammon mapping of section 5.2.3 draws them as compact groups in the plane and shows how the trajectory wanders from regime to regime [51].

Algorithm 9.4 State-space reconstruction and prediction by fuzzy clustering

Input: scalar series y_k, lag τ (mutual information or autocorrelation), range of d_e, number of regimes c, exponent m, tolerance ϵ

for *every embedding dimension* d_e *in the range* **do**
  delay vectors $\mathbf{y}_k$ by (9.17); identification pairs $(\mathbf{y}_k, \mathbf{y}_{k+1})$; initial partition from FCM on the $\mathbf{y}_k$;
  repeat
    centers $\mathbf{v}_i$ and covariances $\mathbf{F}_i$ of the delay vectors; local models $\boldsymbol{\theta}_i$ by weighted least squares; error covariances $\mathbf{P}_i$; priors $p(\eta_i)$;
    distances by (9.19); partition $\mu_{ik} = 1/\sum_j (D_{ik}/D_{jk})^{2/(m-1)}$;
  until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;
  record the one-step test error of (9.18) and the index J^{d_e} of (9.20);
d_e: the smallest dimension after which the error stops improving; d_l: one less than the dimension where J^{d_e} collapses;

Output: embedding and local dimension, operating-regime model for one-step prediction and free-run simulation

In a nutshell Chaotic series by clustering in the reconstructed space

- Delay vectors reconstruct the state space (Takens); the dynamics is identified as an operating-regime model with Gaussian regimes and local linear maps by modified Gath–Geva clustering of $(\mathbf{y}_k, \mathbf{y}_{k+1})$.
- Clustering over a range of d_e gives the embedding dimension (saturation of the one-step error), the local dimension (collapse of the prior-weighted smallest eigenvalue) and a predictor.
- Free-run simulation reproduces the attractor but diverges from the measured trajectory with the horizon, as it must for a chaotic system.
- Parameters: lag τ, number of regimes c (10–20), regularization of $\mathbf{F}_i$ and $\mathbf{P}_i$; visualize the regimes with the fuzzy Sammon mapping.

9.2.5 Structure selection with mixed discrete and continuous inputs

Suppose every candidate input x_j takes values in a finite set $A_j = \{a_j^1, \dots, a_j^{c_j}\}$. For a subset U of the inputs the data decompose into *segments*—one for every combination of values $a_U \in \times_{j\in U} A_j$—and if U contains the inputs that drive the output, y varies little inside each segment. The variance of y within the segments, summed over the segments, is therefore a model-free measure of how well the subset explains the output, as in a (discrete) analysis of variance. There are $n_s = \prod_{j\in U} c_j$ segments, so only subsets up to a maximal cardinality are enumerated; exhaustive enumeration up to that size is feasible for up to about fifteen candidate inputs.

Continuous inputs do not take finitely many values, and quantizing them into crisp bins introduces an arbitrary choice of levels. Girimonte, Babuška and Abonyi [124] fuzzify the decomposition: every continuous input x_j is clustered on its own by fuzzy c-means into c_j fuzzy sets (c_j fixed or chosen by the Xie–Beni index [31]), giving a membership matrix $\mathbf{U}^j \in [0,1]^{c_j \times N}$, and a discrete input gets one crisp set per value. The fuzzy segments of a structure U are the product-space conjunctions of the sets of its inputs,

$$\beta_{ik} = \prod_{j\in U} \mu^j_{i_j k}, \qquad i = (i_1, \dots, i_{n_i}) \in \times_{j\in U}\{1, \dots, c_j\}, \qquad \gamma_{ik} = \frac{\beta_{ik}}{\sum_{i'} \beta_{i'k}}, \tag{9.21}$$

(because the memberships of every input sum to one for each sample, the β_{ik} already sum to one over the segments), every segment is represented by the membership-weighted mean of the output, and the *total fuzzy variance* is the sum of squared errors of the singleton fuzzy model that the segments define:

$$b_i = \frac{\sum_k \gamma_{ik} y_k}{\sum_k \gamma_{ik}}, \qquad \sigma_t = \sum_{k=1}^{N} \Big(y_k - \sum_i \gamma_{ik} b_i \Big)^2. \tag{9.22}$$

Structures are ranked by increasing σ_t. An irrelevant input only splits every segment into c_j smaller ones; it does not increase σ_t and usually lowers it a little, so the structures at the top of the list tend to contain the relevant inputs plus one or two dummies. The *variance–complexity trade-off*

$$VCT = \sigma_t \, n_i \tag{9.23}$$

penalizes the number of inputs n_i and separates structures of *similar* variance: plotted along the ranking it shows a clear dip at the structure with the relevant inputs only. The paper compares the VCT only among structures of similar σ_t, and `fuzzyclust` follows this reading: it recommends the structure with the smallest VCT among those whose σ_t lies within a relative tolerance of the smallest one. The method is a screening tool: it returns a ranked list, and the final choice is made by fitting models for a few top candidates, since the eventual validation criterion (e.g. free-run simulation error) differs from the one used in the selection.

Algorithm 9.5 Structure selection by fuzzy variance decomposition

Input: candidate inputs $\mathbf{X}$ (discrete and continuous), output $\mathbf{y}$, numbers of fuzzy sets c_j (or Xie–Beni), maximal number of inputs

for *every input* j **do**
  if x_j *is discrete* **then** $\mathbf{U}^j$: one crisp set per value;
  else $\mathbf{U}^j$: fuzzy c-means of x_j with c_j sets (ordered by center);
for *every subset* U *of at most the maximal number of inputs* **do**
  segment memberships γ_{ik} by (9.21); segment means b_i and σ_t by (9.22); $VCT = \sigma_t n_i$;
sort the structures by σ_t; among those with σ_t close to the minimum pick the smallest VCT;

Output: ranked list of structures with σ_t and VCT, recommended structure, the partitions $\mathbf{U}^j$

In a nutshell Structure selection with mixed inputs

- Fuzzy c-means turns continuous inputs into fuzzy values, so discrete and continuous inputs are treated alike and no model is fitted.
- Every subset of inputs is scored by the total fuzzy variance of the output over its product-space segments; the VCT separates structures of similar variance by their size.
- Exhaustive search up to a maximal cardinality (about 15 candidate inputs at most); the result is a ranked list to be confirmed by modeling.
- Parameters: sets per input (3 is a good default; many sets need many samples), maximal cardinality, tolerance for "similar variance".

9.3 Using it in fuzzyclust

All methods of this chapter are in `fuzzyclust.timeseries`. The segmentation is a `BaseFuzzyClusterer` whose `fit` accepts an optional time axis `t`; the other methods are functions or small estimators with their own signatures, because most of them take sequences rather than sample matrices. Data generators reproduce the benchmarks of the papers, and plot helpers draw the typical figures. Note that the merging threshold γ of section 9.2.1 is the parameter `merge_threshold`; the parameter `gamma` of `FuzzySegmentation` and `ChaoticPredictor` is the covariance shrinkage of section 3.2.4.

fuzzyclust.timeseries.FuzzySegmentation

`n_segments=5, m=2.0, n_components=None|int|accuracy, distance="ppca"|"q"|"t2", pca_update="em"|"eig", merge=False, merge_threshold=0.75, merge_every=100, min_segment_weight=0.02, gamma=0.0, beta=1e15, tol, max_iter, init="uniform"|"random"|"kmeans++"|array, n_init=1, random_state, verbose.`
`fit(X, y=None, t=None)`; attributes `boundaries_` (crisp borders), `time_membership_` ($\beta_i(t_k)$), `membership_` (posteriors), `labels_`, `n_segments_`, `n_components_`, `segment_centers_`, `segment_variances_`, `cluster_centers_`, `covariances_`, `pca_loadings_`, `noise_variances_`, `norm_matrices_`, `eigenvalues_`, `eigenvectors_`, `local_pca_`, `priors_`. Methods `predict(X, t=None)`, `predict_proba(X, t=None)`, `transform(X, t=None)`, `compatibility_matrix()` (O, S_{PCA}, center distances), `merge_compatible_segments(threshold, max_merges)`, `hotelling_t2(X)`, `q_reconstruction_error(X)`.

fuzzyclust.timeseries.bottom_up_segmentation(X, n_components=2, max_error=None, n_segments=None, cost="q"|"t2", init_length=10, center=True)

Returns a `Bunch` with `labels`, `boundaries`, `segments`, `merge_costs`, `n_segments`; `pca_segment_cost(X, n_components, cost)` is the cost of one segment.

fuzzyclust.timeseries.fnn_model_order(y, u=None, max_na=4, max_nb=0, n_clusters=6, threshold=None, fnn_tol=5.0, m=2.0, scale=True, delay=1, random_state=None)

Returns a `Bunch` with `table` (DataFrame of percentages, rows n_a, columns n_b), the selected `n_a` and `n_b`, and `thresholds` (the point-wise R_k of every order); `threshold=` replaces R_k by a constant, `delay=` is the embedding lag for a scalar series. `select_model_order(...)` returns only `(n_a, n_b)`, `false_nearest_neighbors(y, u, max_na, max_nb, R_tol, A_tol, scale, delay)` is the classical test of Kennel et al., and `cluster_based_threshold(Phi, target, n_clusters)` returns the thresholds of one order and the fitted `GG` model.

fuzzyclust.timeseries.ChaoticPredictor

`dim=3, delay=1, n_clusters=10, m=2.0, target="scalar"|"full", gamma=0.0, beta=1e8, tol, max_iter, init="fcm"|"random"|array, random_state, verbose.`
`fit(y)` on a scalar series; `predict(Y)` for delay vectors, `predict_`

`series(y)`, `one_step_error(y)`, `simulate(y_history, n_steps)`, `multi_step_error(y, horizons)`, `regime_membership(Y)` (= `predict_proba`), `transform(Y)` (Mahalanobis distances, for FuzzySammon); attributes `embedding_`, `membership_`, `cluster_centers_`, `covariances_`, `local_models_`, `error_covariances_`, `priors_`, `eigenvalues_`, `local_dimension_index_`. Helpers: `delay_embedding(y, dim, delay)`, `select_delay(y, max_lag, method)`, `embedding_scan(y, dims, delay, n_clusters, train_fraction)`, `select_embedding_dimension(table, ratio_tol)`, `correlation_dimension(Y)`.

`fuzzyclust.timeseries.MixedInputStructureSelection`

`n_clusters=3|"xie-beni"|sequence`, `max_inputs=None`, `m=2.0`, `vct_tolerance=0.2`, `random_state`.
`fit(X, y, discrete=None, feature_names=None)`; `ranking_` (DataFrame with `inputs`, `n_inputs`, `sigma_t`, `vct`, indexed by rank), `best_structure_`, `memberships_`, `centers_`, `transform_segments(inputs)`. `select_structure_mixed(X, y, discrete, ...)` returns the ranking directly; `fuzzy_variance(memberships, y)` is σ_t of one structure.

data generators and plot helpers

`make_piecewise_timeseries` (hidden regime changes of kind mean / correlation / both), `make_nonlinear_io_system` (the polymerization reactor), `make_mackey_glass`, `make_lorenz`, `make_rossler`, `make_mixed_input_system`; `plot_segmentation(model, X)`, `plot_fnn_table(table)`, `plot_embedding_scan(table)`, `plot_phase_space(Y, labels)`, `plot_structure_ranking(ranking)`.

A series is segmented and its borders are read off in a few lines, and the number of segments is found by merging from a large initial number:

```
fs = FuzzySegmentation(n_segments=4, n_components=2).fit(X)
print("segment borders:", fs.boundaries_)
print("time centers   :", np.round(fs.segment_centers_))
print("time std       :", np.round(np.sqrt(fs.segment_variances_), 1))
```

```
fs10 = FuzzySegmentation(n_segments=10, n_components=2, merge=True,
    merge_threshold=0.75).fit(X)
print("segments after merging:", fs10.n_segments_, "borders:",
    fs10.boundaries_)
```

The bottom-up baseline and the model-order selection are single calls:

```
bu_q = bottom_up_segmentation(X, n_components=2, n_segments=4, cost="q",
    init_length=20)
bu_t2 = bottom_up_segmentation(X, n_components=2, n_segments=4, cost="t2",
    init_length=20)
fuzzy = FuzzySegmentation(n_segments=4, n_components=2).fit(X)
print("bottom-up, Q   :", bu_q.boundaries)
print("bottom-up, T^2 :", bu_t2.boundaries)
print("fuzzy          :", fuzzy.boundaries_)
```

```
res = fnn_model_order(y, u, max_na=3, max_nb=3, n_clusters=6,
    random_state=0)
print(res.table.round(2))
print("selected order: n_a =", res.n_a, " n_b =", res.n_b)
```

For a chaotic series the lag is chosen, the embedding dimension is scanned and the model is identified and simulated:

```
states = make_rossler(n_samples=2200, dt=0.05)
x = states[:, 0]  # the only measured variable
tau = select_delay(x, max_lag=60, method="mutual_information")
n_train = 1500
print("lag from the mutual information:", tau)
```

```
table = embedding_scan(x, dims=(1, 2, 3, 4, 5), delay=tau, n_clusters=8,
    train_fraction=n_train / x.size,
                       random_state=0)
choice = select_embedding_dimension(table, ratio_tol=1.3)
print(table.round(5))
print(f"embedding dimension d_e = {choice.embedding_dimension}, local
    dimension d_l = {choice.local_dimension}")
```

```
model = ChaoticPredictor(dim=3, delay=tau, n_clusters=12,
    random_state=0).fit(x[:n_train])
print(f"one-step test MSE: {model.one_step_error(x[n_train:]):.4f}
    (variance of x: {x.var():.2f})")
simulated = model.simulate(x[:n_train], n_steps=x.size - n_train)
```

The structure selection ranks all small subsets of inputs:

```
X, y, discrete = make_mixed_input_system(n_samples=1000, noise=0.1,
    n_dummy=2, random_state=0)
sel = MixedInputStructureSelection(n_clusters=3, max_inputs=5,
    random_state=0).fit(X, y, discrete=discrete)
print(sel.ranking_.head(8).to_string())
print("recommended structure:", sel.best_structure_)
```

9.4 Examples

9.4.1 Fuzzy segmentation of a multivariate series

Two hidden variables generate six observed ones through a mixing matrix that changes at $t = 500$ (correlation change), while their mean jumps at $t = 1000$ and both change at $t = 1500$ (fig. 9.4); the first change leaves every mean in place and is invisible in any single variable. The scree plot of ten uniform initial segments (fig. 9.5) shows two dominant eigenvalues followed by four small ones at the noise level ($0.15^2 \approx 0.02$), so $q = 2$; inside one regime the two components explain 85–88 % of the variance, so an accuracy of 95 % in (9.8) would ask for more components. With four segments the borders are found at 498, 1001 and 1500 (fig. 9.6). The Gaussian segments in time overlap only around the borders, and the posteriors are almost crisp inside the segments because the PCA models of neighboring regimes explain each other's data badly. The PCA similarity factors of the adjacent models are 0.28, 1.00 and 0.58, small where the correlation structure changed and one where only the mean moved, as in the paper. Figure 9.7 varies the two structural parameters. With fewer than four segments, borders are sacrificed: three segments keep the correlation change (found at 539) and the change at 1500 but drop the mean shift at 1000, and two segments keep only the mean shift at 1000. With more than four segments a regime is split, and the surplus borders fall inside the regimes or next to the true ones. The number of components hardly matters for the PPCA distance: $q = 1$ moves one border by sixty samples, and from $q = 2$ on the result is stable. The printed comparison of the distance measures confirms the discussion of section 9.2.1: Q works with $q = 2$ and fails with $q = 5$, T^2 fails with $q = 2$ and works with the q given by an accuracy of 99 %, and PPCA works with both. Finally, merging from ten segments with $\gamma = 0.75$ (fig. 9.8) reduces the number of segments to four with the right borders. The compatibility matrix of the initial segments shows blocks of mutually compatible surplus segments inside each regime, and the printed scan of γ shows that a lower threshold (0.5–0.65) merges the two regimes that differ only in their mean (and moves the first border to 539), whereas 0.75 and 0.85 keep all four.

9.4.2 Bottom-up baseline versus fuzzy segmentation

On the same series the bottom-up algorithm with the Q cost finds the two correlation changes and misses the mean shift, with the T^2 cost it finds the two mean shifts and misses the correlation change, and the fuzzy segmentation finds all three (fig. 9.9); each cost also places one spurious border because it has to deliver four segments. The merge-cost curves (fig. 9.10) jump when pieces of different regimes have to be joined, and for both costs the knee is at three segments, the number of regimes each cost can see. The parameter scan printed by the script shows that the Q cost depends strongly on p (with $p = 1$ the displacement of a mean shift leaves the one-dimensional model and is detected by accident; with $p = 5$ the residual is noise), that the T^2 cost is insensitive to p but blind to the correlation change, and that a long initial piece coarsens the borders. The fuzzy segmentation

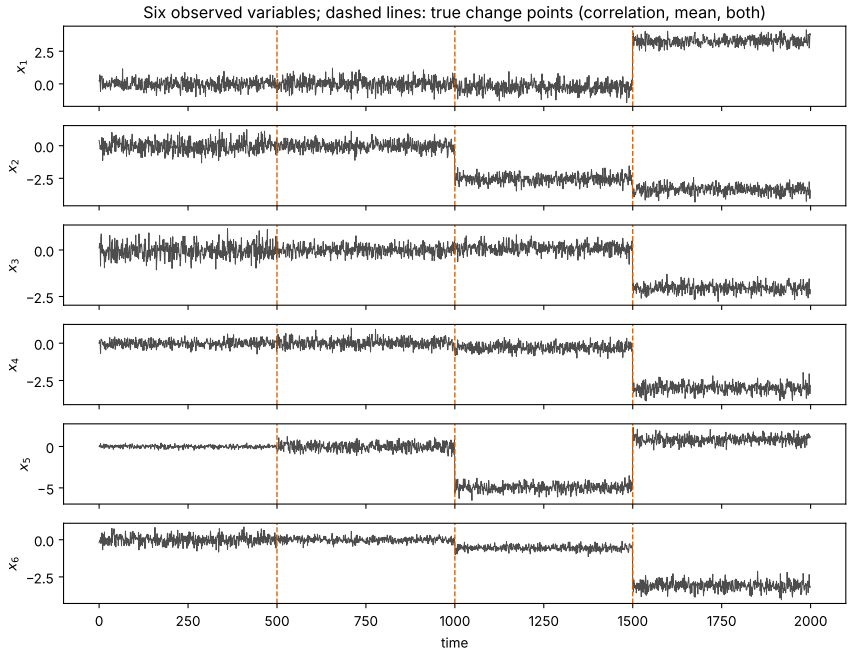


Figure 9.4. The six observed variables of the synthetic series change regime at the dashed lines; the first change alters only the correlation and is visible in no single variable.

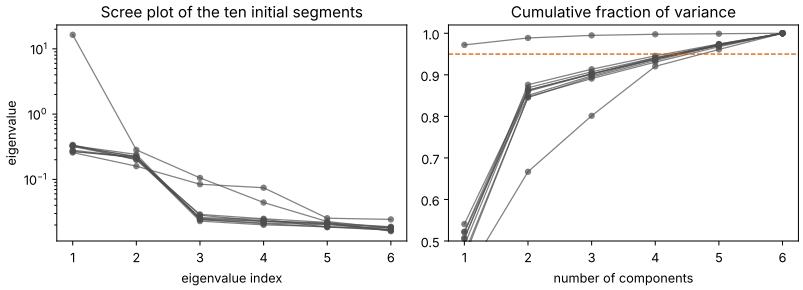


Figure 9.5. The scree plot (left) and the cumulative fraction of variance (right) of the ten initial segments show two dominant components above a noise floor; the dashed line marks an accuracy of 95 %, which two components reach only in the segment dominated by a mean shift.

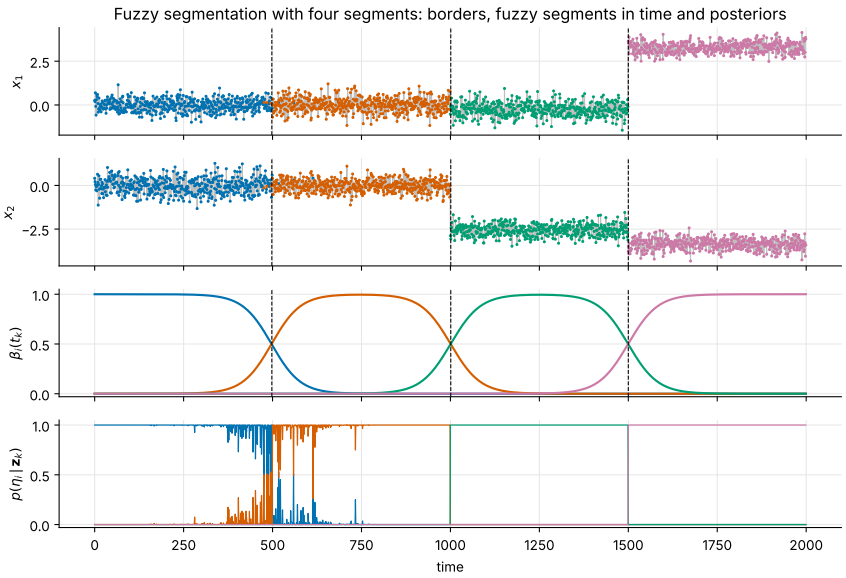


Figure 9.6. Two variables colored by the hard segment, the Gaussian segments in time $\beta_i(t_k)$ and the posterior memberships show the fuzzy segmentation with four segments; dashed lines mark the crisp borders.

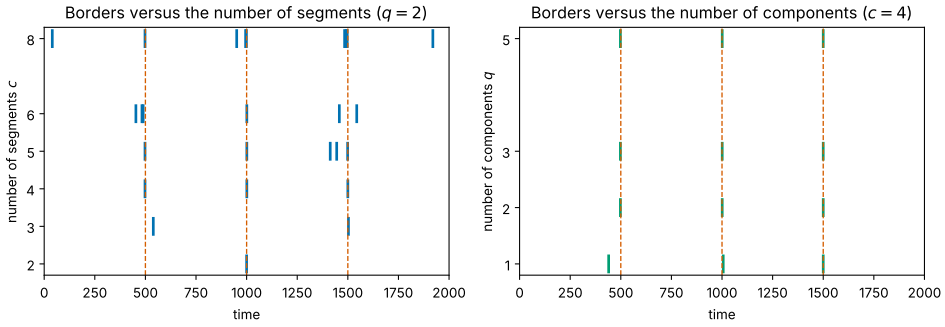


Figure 9.7. The borders found for different numbers of segments (left, $q = 2$) and of components (right, $c = 4$) are compared with the true change points (dashed lines).

with $q = 1, 2, 3, 5$ finds the three borders every time, with $q = 1$ one of them sixty samples off.

9.4.3 Model order of the polymerization reactor

Six hundred samples of the reactor under a random initiator flow (fig. 9.11) are analyzed for orders up to $n_a = n_b = 3$. With the cluster-based threshold (fig. 9.12) the percentage of false neighbors collapses in the corner $n_a \geq 1$, $n_b \geq 2$, and the smallest order there, $(1, 2)$, is the one reported in the paper. The point-wise thresholds of that order (fig. 9.13) differ by more than 60 % between the six operating

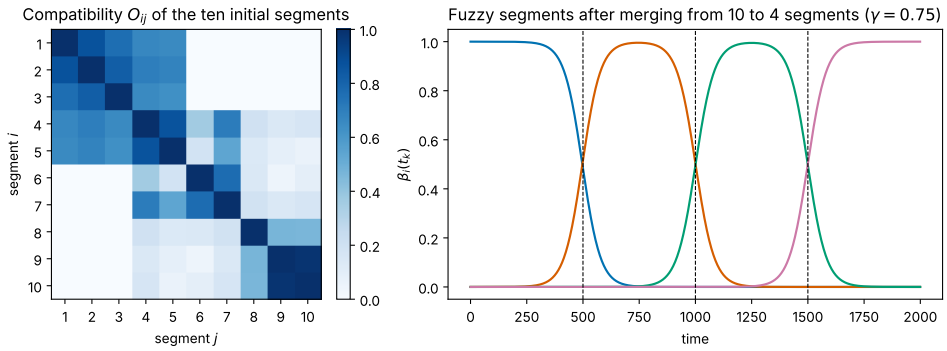


Figure 9.8. The compatibility matrix of the ten initial segments is shown on the left, the fuzzy segments after merging to four with $\gamma = 0.75$ on the right.

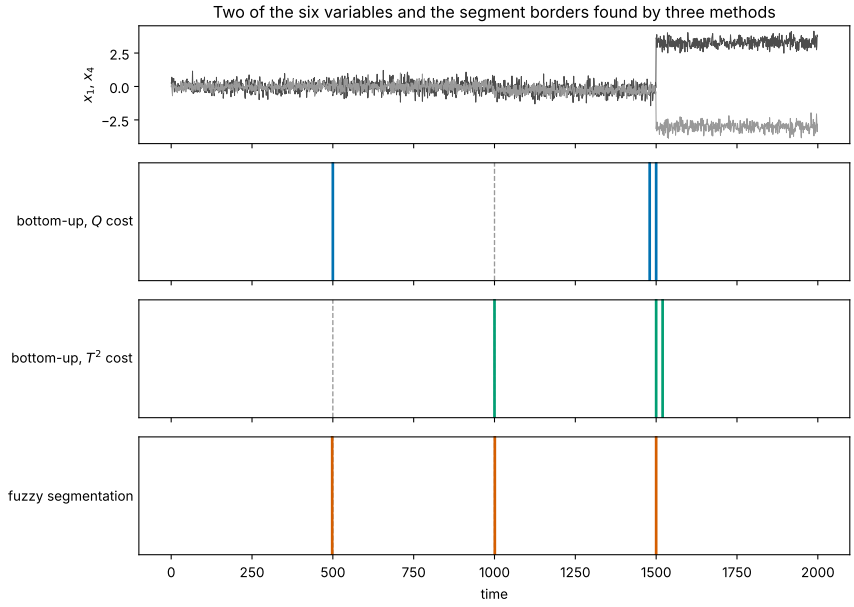


Figure 9.9. The borders found by the bottom-up algorithm with the Q cost, by the bottom-up algorithm with the T^2 cost and by the fuzzy segmentation, all with four segments, are compared with the true change points (gray dashed lines).

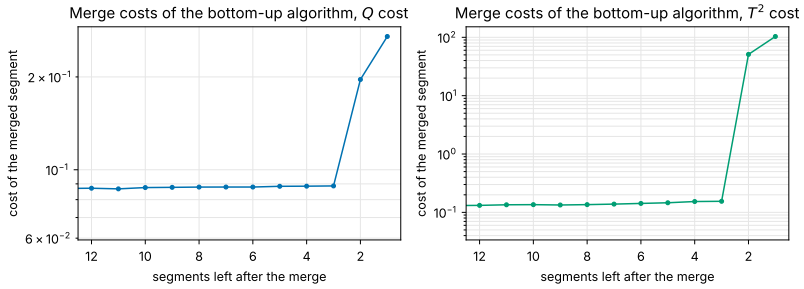


Figure 9.10. The cost of every merge of the bottom-up algorithm with the Q cost (left) and the T^2 cost (right) is plotted against the number of segments left; the jump marks the merge of two different regimes.

regions found by the clusters, so no constant threshold fits everywhere. Figure 9.14 shows how a constant threshold moves the corner. A strict constant ($R = 0.5$, below every local slope) keeps flagging more than a third of the neighbors at every order, so no corner appears; $R = 1$ happens to give the right order; a loose constant ($R = 3$, above every local slope) also passes wrong orders of the same size, $(0, 3)$ and $(2, 1)$, and selects $(0, 3)$. Kennel's two-criterion test is looser still and accepts models that are too small, $(0, 2)$ and $(1, 1)$. The selected order is the same for 3, 6 and 10 clusters. A model-based check with linear ARX models of all orders (fig. 9.15) agrees: the normalized error drops sharply up to $(1, 2)$ and then flattens.

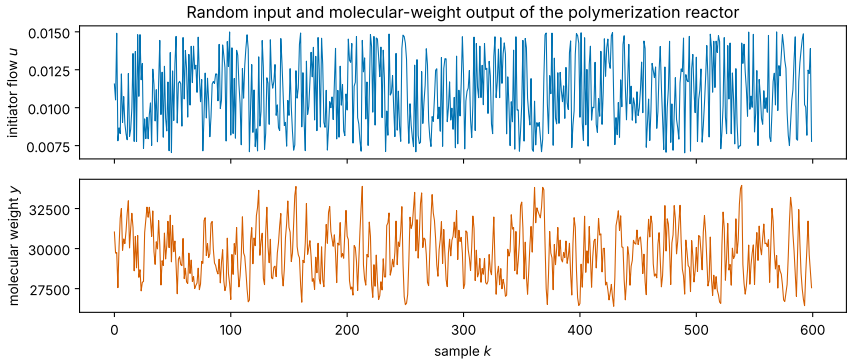


Figure 9.11. The polymerization reactor benchmark is driven by a random initiator flow (top) and observed through the molecular weight (bottom).

9.4.4 Reconstruction and prediction of the Rössler series

The Rössler system is observed through x_1 (fig. 9.16); the first minimum of the mutual information gives a lag of 15 samples. The embedding scan with eight regimes (fig. 9.17) shows the one-step test error dropping by a factor of nine from $d_e = 1$ to 2 and by 1.8 to $d_e = 3$, after which it fluctuates around a floor, so $d_e = 3$.

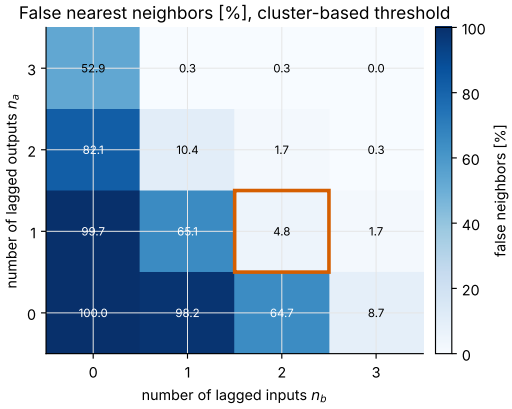


Figure 9.12. The table gives the percentage of false nearest neighbors over the orders with the cluster-based threshold; the frame marks the selected order $(n_a, n_b) = (1, 2)$.

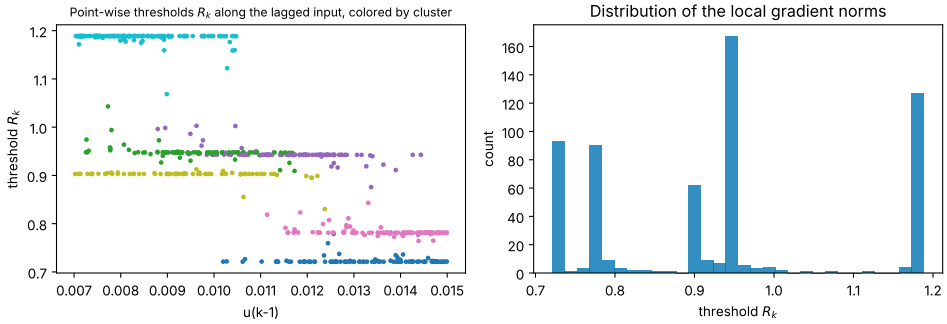


Figure 9.13. The point-wise thresholds R_k of the order $(1, 2)$ are plotted along the lagged input and colored by cluster (left), and their distribution is shown on the right.

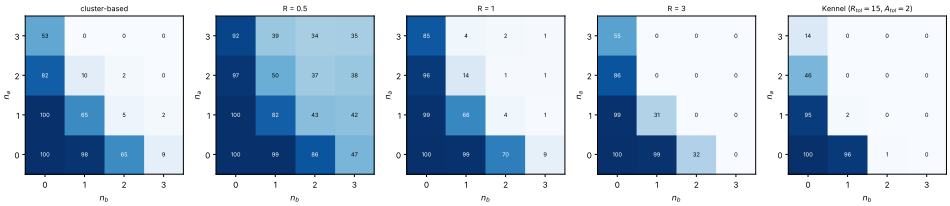


Figure 9.14. The percentage of false nearest neighbors with the cluster-based threshold, with three constant thresholds and with the classical test of Kennel et al. shows that the corner moves with the constant.

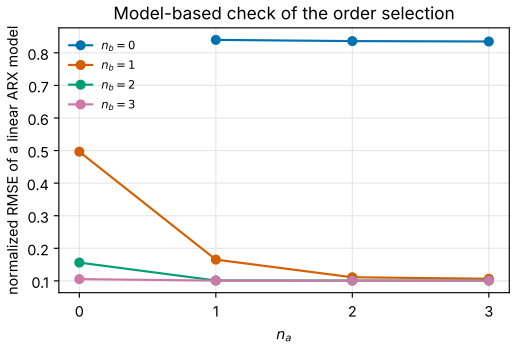


Figure 9.15. The normalized one-step error of linear ARX models over the orders confirms the selected order with a model-based check.

The prior-weighted smallest eigenvalue drops by a factor of three at $d_e = 3$ (from 1.26 to 0.41) and keeps falling, because the clusters are flat from three dimensions on, so $d_l = 2$, which reflects the well-known band structure of the Rössler attractor. The FNN test with the cluster-based threshold on the same series collapses at $d_e = 3$ as well, and the correlation dimension of the reconstructed vectors is 1.92. With twelve regimes the model tiles the reconstructed attractor (fig. 9.18, left); its free-run simulation keeps spiraling on a Rössler-like band but visits the outer part more often than the system does (right). In time (fig. 9.19) the simulation follows the measured series for about two and a half cycles (some 280 samples) before the chaotic divergence takes over, and the multi-step error grows with the horizon from $3 \cdot 10^{-4}$ of the variance one step ahead to 0.4 of the variance after 80 steps. The fuzzy Sammon mapping (fig. 9.20) shows the twelve regimes as compact groups arranged in the cyclic order in which the trajectory visits them, whereas the PCA projection of the same vectors overlaps the clusters that differ in the third coordinate. The parameter scan printed by the script shows that more regimes lower both the one- and the ten-step error at the lags 5 and 15, that a very short lag gives small one-step errors (consecutive samples are almost collinear) but a stretched attractor, and that a long lag (30 samples, about a third of the mean cycle) folds the attractor and raises the errors. The three systems compared at the end have correlation dimensions close to two (1.97, 1.90 and 2.00); the percentage of false neighbors of the classical test falls below the tolerance of 5 % at $d_e = 3$ for the Rössler and Mackey–Glass series and already at $d_e = 2$ (3.9 %) for the Lorenz series.

9.4.5 Structure selection with a switching input

The output switches between x_1^2 and $x_4^2 + x_3$ according to the binary input x_2, and two irrelevant inputs are added. With three fuzzy sets per continuous input and structures of up to five inputs, the three best structures by σ_t all contain the four true inputs; the two with a dummy have a marginally smaller variance but

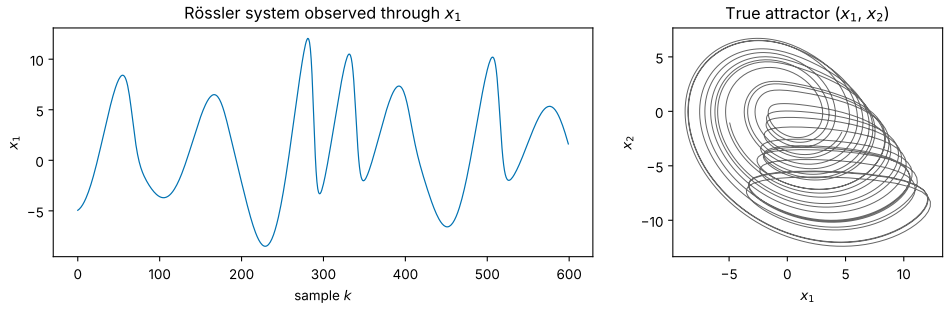


Figure 9.16. The Rössler system is observed through x_1 (left); its true attractor is shown in the (x_1, x_2) plane (right).

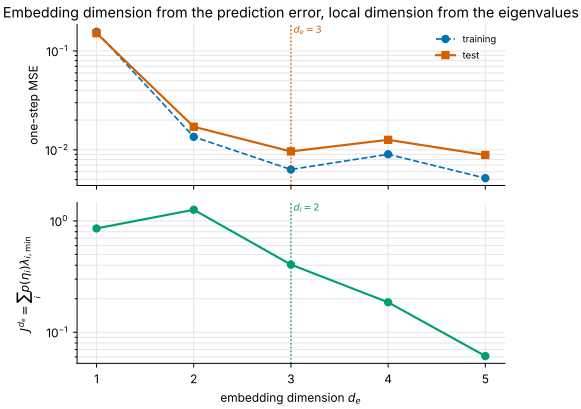


Figure 9.17. The embedding scan plots the one-step training and test errors (top) and the prior-weighted smallest eigenvalue J^{d_e} (bottom) against the embedding dimension and marks the selected $d_e = 3$ and $d_l = 2$.

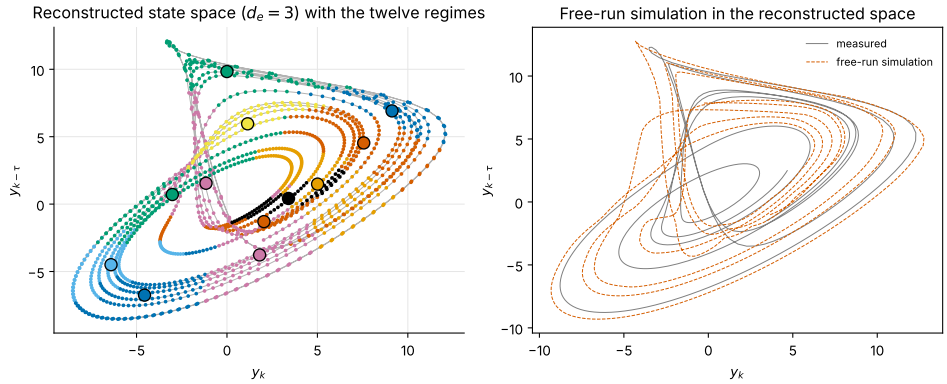


Figure 9.18. The reconstructed state space is shown with the twelve regimes and their centers (left) and with the free-run simulation of the test part (right).

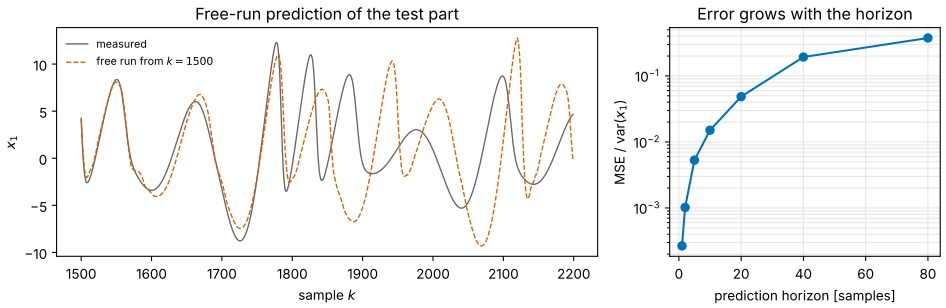


Figure 9.19. The free-run prediction of the test part is compared with the measured series in time (left), and its error is plotted against the horizon (right).

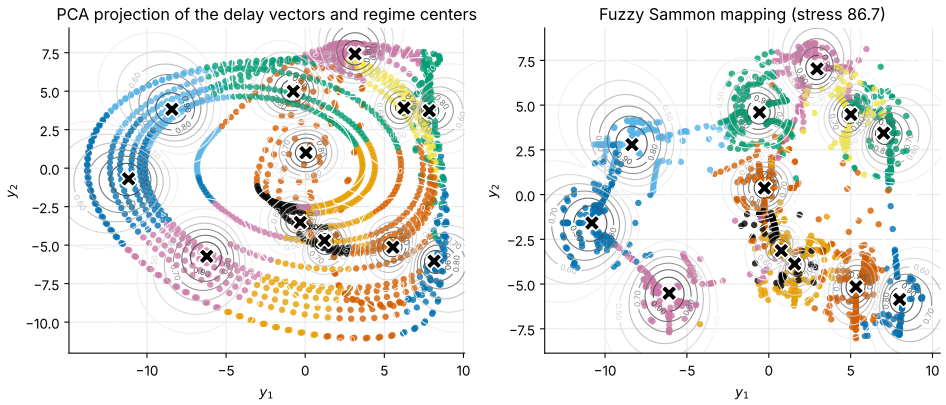


Figure 9.20. The delay vectors and the regime centers are projected to the plane by PCA (left) and by the fuzzy Sammon mapping (right).

a larger VCT, and $\{x_1, x_2, x_3, x_4\}$ is the first dip of the VCT curve (fig. 9.21), as in the paper. Figure 9.22 shows the fuzzy sets of x_1 and the crisp sets of x_2 behind the decomposition. The parameter scan printed by the script shows the trade-off in the number of sets. Two sets resolve the quadratic terms poorly and drop x_1 under strong noise; five sets spread the thousand samples over the 250 segments of the true structure and let a dummy input in; three sets are the best choice for a thousand samples, although with 300 samples a dummy enters there too. On the Friedman function with five relevant and five irrelevant inputs (fig. 9.23) the six best structures by σ_t are the true one and the true one plus each dummy, the VCT singles out the true one, and x_3 is the first relevant input to be dropped further down the list, as in Table I of the paper.

9.5 Further reading

The fuzzy segmentation was published in [114]; its conference precursor with the Q and T^2 variants is [115]. The method builds on Gath–Geva clustering [21], on

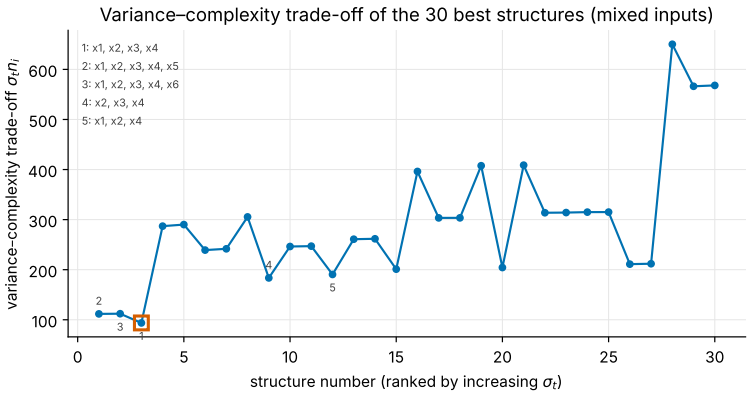


Figure 9.21. The variance–complexity trade-off is plotted for the 30 best structures of the mixed-input problem; the framed dip is the recommended structure $\{x_1, x_2, x_3, x_4\}$.

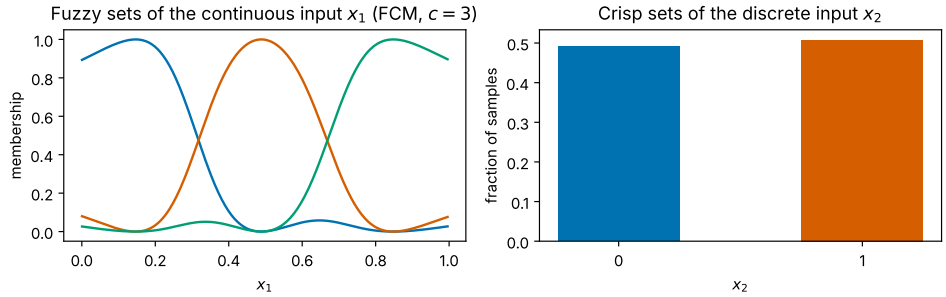


Figure 9.22. Fuzzy c-means gives three fuzzy sets of the continuous input x_1 (left), and the binary input x_2 has two crisp sets (right).

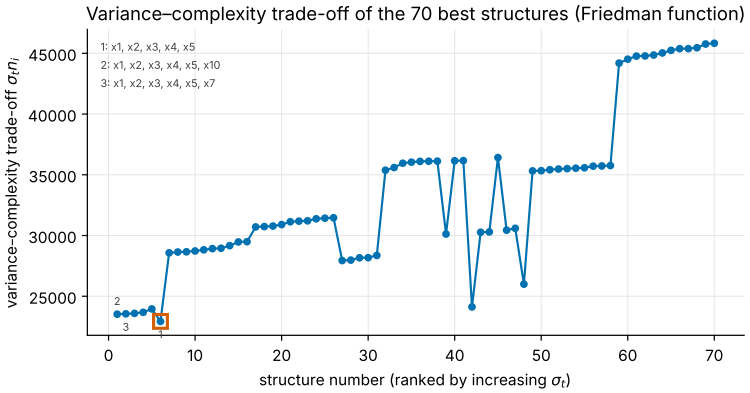


Figure 9.23. The variance–complexity trade-off is plotted for the 70 best structures of the Friedman function with five dummy inputs; the dip at structure six is the true set of inputs.

the modified Gath–Geva algorithm of chapter 8 [96], on probabilistic PCA and its mixtures [125], on Krzanowski's similarity of principal components [126], on compatible cluster merging [40] with Kelly's merging formulas [41], and on the covariance regularization of [19]. The bottom-up algorithm and its relatives are described in [112, 113]. The false-nearest-neighbor test originates in [116], its input–output version in [117, 118] and the Lipschitz-quotient idea in [131]; the cluster-based threshold was proposed in [119] and published in journal form in [120], the polymerization benchmark comes from [132], and model-order selection for linear systems is covered by [133]. State-space reconstruction rests on [121, 122], the choice of the lag on [127], embedding-dimension methods on [116, 134, 135], local prediction on [128, 129], operating-regime modeling on [7, 136], and the correlation dimension on [130]; the clustering-based method was proposed in [123] and its visualization by fuzzy Sammon mapping in [51]. The benchmark systems are those of Mackey and Glass [137], Lorenz [138] and Rössler [139]. The structure selection with mixed inputs was proposed in [124], which uses the Xie–Beni index [31] and the Friedman function of [140]. The c-means functional behind all these methods is Bezdek's [3].

CHAPTER

10



Clustering relational, categorical, and mixed data

All algorithms of the previous chapters work on a matrix of numeric feature vectors: a prototype is a weighted mean, and the distance to it is a norm. This chapter drops that assumption in three directions. When only the pairwise dissimilarities of the objects are known—time series compared by dynamic time warping, records compared by a coefficient of their own—relational fuzzy c-means minimizes the familiar functional through the dissimilarity matrix alone, NERF c-means repairs it for matrices that no point configuration can realize, and fuzzy c-medoids chooses actual objects as prototypes and accepts any dissimilarity. For categorical attributes fuzzy k-modes replaces the mean by the mode, and for mixed records fuzzy k-prototypes combines the two; dissimilarities built for time series and mixed records turn such data into input for the relational methods. The methods extend the medoid idea of the geodesic clustering of chapter 7 and live in `fuzzyclust.relational`.

10.1 Problem and idea

Fuzzy c-means needs two operations: it averages samples to obtain a prototype, $\mathbf{v}_i = \sum_k \mu_{ik}^m \mathbf{x}_k / \sum_k \mu_{ik}^m$, and it measures the distance of a sample to that average. Both presuppose that the objects are vectors of a common space in which a mean exists and means something. Many data sets violate this. Time series that are shifted and stretched in time can be compared sensibly only after they have been aligned, and the result of the alignment is a number per *pair* of series, not a coordinate per series. Survey answers, diagnoses, and product types are categories: one can say whether two answers agree, but there is no average of "yes", "no", and "don't know". Customer records combine both—age and income next to occupation and region. In some applications the dissimilarities are all there is: subjective

ratings of the similarity of products, edit distances between strings, and distances between graphs.

The common way out is to force the objects into a vector space—resample and align the series, code the categories as numbers or 0/1 indicators—and run FCM. The geometry is then an accident of the encoding: the indicator coding makes every categorical mismatch cost the same whatever the attribute, the numeric columns count in their arbitrary units, and the centers are fractional indicator vectors that describe no record. This chapter takes the other route and changes the algorithm instead of the data. Two observations make that possible.

The first is that fuzzy c-means never needs the coordinates themselves, only the squared distances between objects. A prototype that is a weighted mean of the objects can be stored as the vector of its weights, $\mathbf{v}_i \in \mathbb{R}^N$, and the squared distance of an object to such a combination follows from the pairwise squared distances by an identity of elementary geometry. Hathaway, Davenport, and Bezdek [141] turned this into *relational* fuzzy c-means (RFCM), which produces the FCM partition of points it has never seen. Figure 10.1 contrasts the two kinds of data and shows where the prototype lives in each: in the feature space as a point, in the relational setting as a weight vector over the objects or as one of the objects.

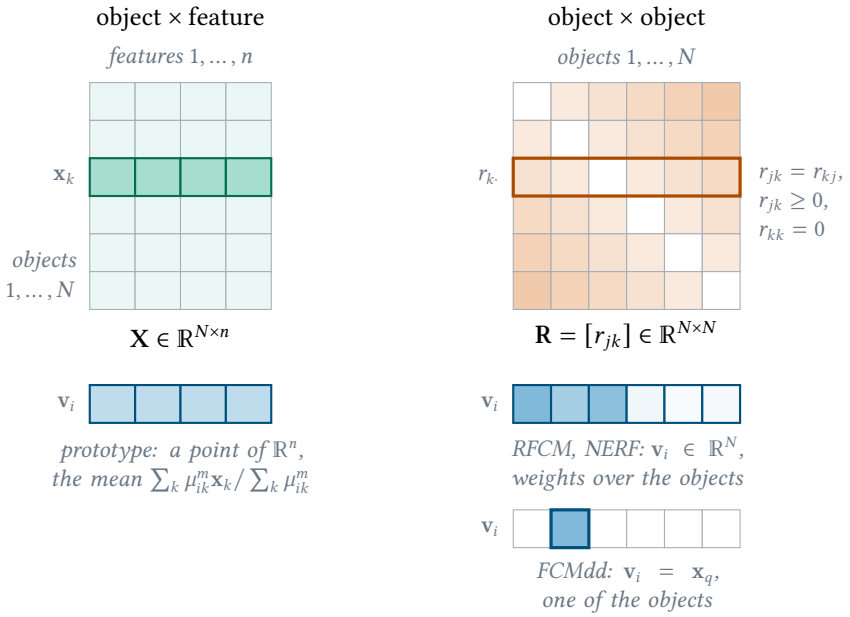


Figure 10.1. Object–feature and object–object data. Left: in the feature setting every object is a row of $\mathbf{X}$ and the prototype is a point of the same space, a weighted mean of the rows. Right: in the relational setting only the symmetric matrix $\mathbf{R}$ of pairwise dissimilarities is known; the prototype of relational c-means is a weight vector over the objects (the shade of a cell is the weight), and that of fuzzy c-medoids is a single object.

The identity behind RFCM holds only if the dissimilarities are squared distances of *some* point configuration, that is, if $\mathbf{R}$ is Euclidean. Dynamic-time-warping matrices, subjective ratings, and most purpose-built dissimilarities are not, and then the relational distance of an object to a prototype can come out negative, a geometric impossibility that breaks the membership formula. Figure 10.2 shows the smallest example: three objects whose dissimilarities violate the triangle inequality, so that no position in any space fits the middle one. Adding the same constant β to all off-diagonal dissimilarities pushes the objects apart until a configuration exists. This *β-spread* is the idea of NERF c-means [142], which applies it adaptively: only when a negative distance appears, and only as far as needed to remove it.

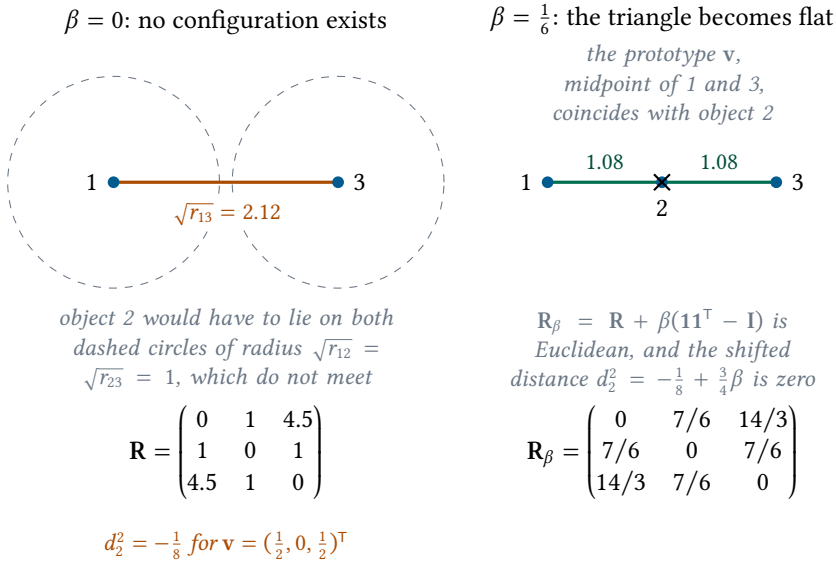
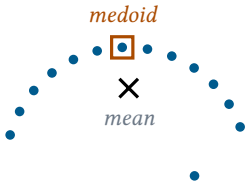


Figure 10.2. The β-spread of NERF c-means. Left: three objects with squared dissimilarities $r_{12} = r_{23} = 1$ and $r_{13} = 4.5$; the long side exceeds the path through object 2, so no point configuration realizes them, and the relational distance of object 2 to the midpoint prototype of objects 1 and 3 is negative. Right: adding $\beta = 1/6$ to every off-diagonal entry makes the matrix Euclidean—the three objects lie on a line—and the distance of object 2 to the prototype becomes exactly zero; this β is the smallest admissible spread.

The second observation is that a prototype need not be a mean at all. Restricting it to the objects themselves—the *medoid*, the object with the smallest weighted sum of dissimilarities to its cluster—requires only the dissimilarities between objects and therefore works with any dissimilarity, Euclidean or not. Krishnapuram, Joshi, Nasraoui, and Yi [143] made the fuzzy c-medoids algorithm (FCMdd) practical for large relational data by searching for the medoid of a cluster only among its most typical members. For categorical attributes the analogous prototype is the *mode*, chosen attribute by attribute as the category that the members of the cluster share most. Huang [144] introduced it for k-means and Huang and Ng [145] for fuzzy partitions, where "most" is measured in membership mass. Figure 10.3 shows why

both describe a cluster better than a mean when the mean lies outside the data or is undefined.

mean versus medoid



the mean of a curved cluster lies where there are no objects and is pulled by the stray point; the medoid is an object on the arc

the mode of a fuzzy cluster

μ_{ik}^m	color	size	shape
0.9	red	S	round
0.8	red	M	round
0.3	blue	M	square
0.2	blue	L	square
0.1	blue	S	round
mode	**red**	**M**	**round**
mass	1.7 / 0.6	1.0 / 1.1 / 0.2	1.8 / 0.5

blue is the most frequent color but carries less membership mass than red; the mode takes the category with the largest $\sum_k \mu_{ik}^m$

Figure 10.3. Mean, medoid, and mode. Left: the weighted mean of a curved cluster lies in empty space; the medoid is one of the objects and summarizes the cluster where it is. Right: for categorical attributes the prototype is the mode, chosen per attribute by membership mass rather than by count; in the color column blue is more frequent but red carries 1.7 of the mass against 0.6.

All methods of the chapter keep the c-means functional and its alternating optimization: the memberships follow from the FCM formula with whatever plays the role of the squared distance, and the prototypes from whatever minimizes the functional for fixed memberships. What changes is the prototype (weights over the objects, an object, a mode, or means and modes) and with it the kind of data the algorithm accepts. The last part of the theory constructs dissimilarities that turn time series and mixed records into relational data.

10.2 Theory

The notation is that of the book: the memberships μ_{ik} form the matrix $\mathbf{U} \in \mathbb{R}^{c\times N}$ (stored transposed, $N \times c$, in the code), and the index i (or j) runs over clusters and k (or l, q) over objects. Relational data form a matrix $\mathbf{R} = [r_{kl}] \in \mathbb{R}^{N\times N}$ with $r_{kl} = r_{lk} \geq 0$ and $r_{kk} = 0$. The matrix $\mathbf{R}$ is called *Euclidean* if there are points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of some $\mathbb{R}^p$ with $r_{kl} = \|\mathbf{x}_k - \mathbf{x}_l\|^2$, that is, if its entries are *squared* distances of a configuration. By the classical result of Schoenberg and of Young and Householder, this is the case exactly when the doubly centered matrix $-\frac{1}{2}\mathbf{HRH}$, $\mathbf{H} = \mathbf{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^\mathsf{T}$, is positive semi-definite [146]. The matrix of squared Euclidean distances of any data set is Euclidean in this sense. So is the matrix of *unsquared* distances, because an entrywise power $0 < p \leq 1$ of a Euclidean matrix remains

Euclidean, but it belongs to a different configuration. Squared city-block distances, cubed distances, and dynamic-time-warping matrices are in general not Euclidean.

10.2.1 Relational fuzzy c-means

Let the prototype of cluster i be a convex combination of the objects, $\mathbf{p}_i = \sum_l v_{il}\mathbf{x}_l$ with weights $v_{il} \geq 0$, $\sum_l v_{il} = 1$, collected in $\mathbf{v}_i \in \mathbb{R}^N$. For any point $\mathbf{x}_k$ and any weights summing to one, expanding the squares gives

$$\left\|\mathbf{x}_k - \textstyle\sum_l v_{il}\mathbf{x}_l\right\|^2 = \sum_l v_{il}\left\|\mathbf{x}_k - \mathbf{x}_l\right\|^2 - \tfrac{1}{2}\sum_{l,q} v_{il}v_{iq}\left\|\mathbf{x}_l - \mathbf{x}_q\right\|^2. \tag{10.1}$$

The right-hand side contains only squared distances between objects. Replacing them by the entries of $\mathbf{R}$ defines the *relational distance*

$$d_{ik}^2 = (\mathbf{R}\mathbf{v}_i)_k - \tfrac{1}{2}\,\mathbf{v}_i^{\mathsf{T}}\mathbf{R}\,\mathbf{v}_i, \tag{10.2}$$

which equals the squared Euclidean distance of $\mathbf{x}_k$ to the weighted mean whenever $\mathbf{R}$ holds squared Euclidean distances. The FCM prototype is the weighted mean with weights $\mu_{ik}^m / \sum_l \mu_{il}^m$, so RFCM [141] alternates

$$\mathbf{v}_i = \frac{(\mu_{i1}^m, \ldots, \mu_{iN}^m)^{\mathsf{T}}}{\sum_{k=1}^N \mu_{ik}^m}, \qquad d_{ik}^2 \text{ by (10.2)}, \qquad \mu_{ik} = \frac{1}{\sum_{j=1}^c \left(d_{ik}^2 / d_{jk}^2\right)^{1/(m-1)}}, \tag{10.3}$$

and minimizes $J = \sum_i \sum_k \mu_{ik}^m d_{ik}^2$, which in the Euclidean case is the c-means functional of Bezdek [3]. The *duality theorem* of the paper states that on the matrix of squared Euclidean distances of a data set, RFCM started from the same initial partition generates exactly the iterates of FCM on that data set (memberships, objective values, and number of iterations), and that the dual centers $\sum_l v_{il}\mathbf{x}_l$ are the FCM centers. RFCM is therefore not an approximation but FCM written in the language of $\mathbf{R}$. The relational prototype $\mathbf{v}_i$ is a probability vector over the objects: it states how much each object contributes to the center of cluster i, and it is concentrated on the typical members of the cluster.

The price is the matrix: one iteration needs the products $\mathbf{R}\mathbf{v}_i$, that is, $O(cN^2)$ operations, and the matrix itself takes $O(N^2)$ memory. A new object enters through its squared distances to the N training objects. Since (10.1) holds for any point, its relational distance is (10.2) with the row of new dissimilarities in place of the k-th row of $\mathbf{R}$, and the membership follows. When $\mathbf{R}$ is not Euclidean, (10.2) has no geometric meaning and can be negative (fig. 10.2); RFCM is then undefined, and the implementation stops with an error that points to NERF.

10.2.2 NERF c-means: the β-spread

Hathaway and Bezdek [142] make relational c-means usable on any dissimilarity matrix by transforming it,

$$\mathbf{R}_\beta = \mathbf{R} + \beta\,(\mathbf{1}\mathbf{1}^{\mathsf{T}} - \mathbf{I}), \qquad \beta \geq 0, \tag{10.4}$$

Algorithm 10.1 Relational fuzzy c-means (RFCM)

Input: Euclidean dissimilarity matrix $\mathbf{R}$ (squared distances), number of clusters c, exponent $m > 1$, tolerance ϵ

initialize the partition $\mathbf{U}$ (random, or from c seed objects);
repeat
  relational prototypes $\mathbf{v}_i = \mathbf{U}_i^m / \sum_k \mu_{ik}^m$ for $i = 1, \dots, c$;
  relational distances $d_{ik}^2 = (\mathbf{R}\mathbf{v}_i)_k - \frac{1}{2}\mathbf{v}_i^\mathsf{T}\mathbf{R}\mathbf{v}_i$ (10.2);
  if *some* $d_{ik}^2 < 0$ **then** stop: $\mathbf{R}$ is not Euclidean (use NERF);
  memberships μ_{ik} by the FCM formula (10.3);
until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: partition $\mathbf{U}$, relational prototypes $\mathbf{v}_i \in \mathbb{R}^N$, objective J

In a nutshell Relational fuzzy c-means

- FCM on a matrix of squared dissimilarities instead of feature vectors; prototypes are weight vectors over the objects.
- Exact dual of FCM: on squared Euclidean distances it reproduces the FCM iterates and centers.
- Assumes a Euclidean $\mathbf{R}$; otherwise negative relational distances stop it.
- Parameters as in FCM (c, m, initialization, restarts); $O(N^2)$ memory, $O(cN^2)$ per iteration.

which adds β to every off-diagonal dissimilarity and keeps the zero diagonal. For a large enough β the transformed matrix is Euclidean: the doubly centered matrix $-\frac{1}{2}\mathbf{H}\mathbf{R}_\beta\mathbf{H} = -\frac{1}{2}\mathbf{H}\mathbf{R}\mathbf{H} + \frac{\beta}{2}\mathbf{H}$ gains $\beta/2$ on every eigenvalue in the subspace orthogonal to $\mathbf{1}$. The effect on the relational distances can be computed without forming $\mathbf{R}_\beta$. Substituting (10.4) into (10.2) and using $\mathbf{1}^\mathsf{T}\mathbf{v}_i = 1$ gives

$$d_{ik}^2(\beta) = d_{ik}^2(0) + \frac{\beta}{2}\left\|\mathbf{e}_k - \mathbf{v}_i\right\|^2, \qquad \left\|\mathbf{e}_k - \mathbf{v}_i\right\|^2 = 1 - 2v_{ik} + \left\|\mathbf{v}_i\right\|^2, \tag{10.5}$$

with $\mathbf{e}_k$ the k-th unit vector. Every distance grows by a positive amount, and the growth is largest for objects that carry little weight in the prototype. The algorithm starts with $\beta = 0$, so that a Euclidean matrix gives the RFCM result, and raises β only when a negative distance appears, by the smallest increment that removes all of them,

$$\Delta\beta = \max_{i,k} \frac{-2\, d_{ik}^2}{\left\|\mathbf{e}_k - \mathbf{v}_i\right\|^2}, \qquad \beta \leftarrow \beta + \Delta\beta, \qquad d_{ik}^2 \leftarrow d_{ik}^2 + \frac{\Delta\beta}{2}\left\|\mathbf{e}_k - \mathbf{v}_i\right\|^2. \tag{10.6}$$

The pair (i, k) that attains the maximum then has distance zero and membership one in cluster i, and the iteration continues with the shifted distances. In the example of fig. 10.2, $d_2^2(0) = -\frac{1}{8}$ and $\left\|\mathbf{e}_2 - \mathbf{v}\right\|^2 = \frac{3}{2}$, so $\Delta\beta = \frac{1}{6}$.

Algorithm 10.2 NERF c-means

Input: any dissimilarity matrix $\mathbf{R}$ (used as squared distances), c, $m > 1$, ϵ

initialize $\mathbf{U}$; $\beta \leftarrow 0$;
repeat
  relational prototypes $\mathbf{v}_i = \mathbf{U}_i^m / \sum_k \mu_{ik}^m$;
  $d_{ik}^2 = (\mathbf{R}\mathbf{v}_i)_k - \frac{1}{2}\mathbf{v}_i^\mathsf{T}\mathbf{R}\mathbf{v}_i + \frac{\beta}{2}\|\mathbf{e}_k - \mathbf{v}_i\|^2$ (10.5);
  if *some* $d_{ik}^2 < 0$ **then**
    increase β by $\Delta\beta$ and shift all distances by (10.6);
  memberships μ_{ik} by the FCM formula;
until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: partition $\mathbf{U}$, prototypes $\mathbf{v}_i$, final spread β, objective of $\mathbf{R}_\beta$

In a nutshell NERF c-means

- Relational c-means for matrices that are not Euclidean (DTW, ratings, squared non-Euclidean metrics).
- The β-spread adds a constant to all off-diagonal dissimilarities; it is raised adaptively and only as far as needed, so a Euclidean matrix gives the RFCM result.
- A large final β flattens the memberships and signals strongly non-Euclidean data; objectives at different β are not comparable.
- Same parameters and costs as RFCM.

The spread has a price. Adding the same constant to all dissimilarities compresses the ratio of far to near dissimilarities, on which the membership formula depends, so a large β flattens the partition toward $1/c$. The objective J, in turn, is that of $\mathbf{R}_\beta$, so the objectives of runs that ended with different β cannot be compared. The final β relative to the mean dissimilarity serves as a diagnostic: zero means that the data behaved as a Euclidean matrix, a large value that the clustering had to distort them considerably. New objects receive the shift with $\|\mathbf{e} - \mathbf{v}_i\|^2 = 1 + \|\mathbf{v}_i\|^2$, the value for an object outside the training set.

10.2.3 Fuzzy c-medoids

Fuzzy c-medoids restricts the prototypes to the objects, $\mathbf{v}_i \in \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, and minimizes the membership-weighted sum of dissimilarities to them,

$$J = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \, r(\mathbf{x}_k, \mathbf{v}_i), \tag{10.7}$$

where r is any dissimilarity: it need not be squared, Euclidean, or even a metric, and apart from non-negativity only $r(\mathbf{x},\mathbf{x}) = 0$ is used. For fixed medoids the memberships follow from the FCM formula with r in the role of the squared distance,

$$\mu_{ik} = \frac{\big(1/r(\mathbf{x}_k,\mathbf{v}_i)\big)^{1/(m-1)}}{\sum_{j=1}^{c}\big(1/r(\mathbf{x}_k,\mathbf{v}_j)\big)^{1/(m-1)}}, \tag{10.8}$$

with $\mu_{ik} = 1$ when $\mathbf{x}_k$ is the medoid of cluster i. For fixed memberships the functional splits into one term per cluster, and the best medoid of cluster i is the object with the smallest weighted sum of dissimilarities,

$$\mathbf{v}_i = \mathbf{x}_q, \qquad q = \arg\min_{l\in C_i} \sum_{k=1}^{N} \mu_{ik}^{m}\, r(\mathbf{x}_k,\mathbf{x}_l). \tag{10.9}$$

With the candidate set C_i containing all objects, the update is exact and costs $O(N^2)$ per cluster. Krishnapuram et al. [143] observed that the medoid of a cluster is almost always one of its most typical members; restricting C_i to the p objects with the largest memberships in cluster i makes the update $O(pN)$ and the whole iteration linear in N (the *linear* FCMdd). The price is that a poor initial cluster may not contain its best medoid among its top candidates, so the iteration can stop in a worse local minimum; a moderate number of candidates and a few random restarts are the remedy. Since the medoids are discrete, the iteration terminates when they no longer change, typically after a few iterations. If two clusters select the same object, the implementation gives the second its best unused candidate, so that the medoids stay distinct.

The same paper proposes a robust variant, RFCMdd. It scores every object by a harmonic-mean-type dissimilarity to all medoids, $h_k = \big(\sum_i r(\mathbf{x}_k,\mathbf{v}_i)^{1/(1-m)}\big)^{1-m}$, keeps only the objects with the smallest scores, and computes the memberships and the medoids from these objects alone, so that a share of badly fitting objects (outliers or noise) cannot pull the medoids. `fuzzyclust` implements a simpler rule in the same spirit, which we call fuzzy c-*trimmed* medoids (FCTMdd): in (10.9) only the $K = \lceil(1-\tau)N\rceil$ smallest terms $\mu_{ik}^{m} r(\mathbf{x}_k,\mathbf{x}_l)$ are summed for every candidate, the memberships of all objects are kept, and the reported objective is the same trimmed sum. FCMdd is related to the geodesic c-medoids of section 7.2.5, which applies the same alternation to graph distances; here, however, the dissimilarity is an input, and the candidate search can be restricted.

10.2.4 Fuzzy k-modes

Let all n attributes be categorical, attribute j taking values in a finite set $\mathcal{A}_j = \{a_j^{(1)},\dots,a_j^{(n_j)}\}$. The *simple matching dissimilarity* counts the attributes in which two records differ,

$$\delta(\mathbf{x}_k,\mathbf{x}_l) = \sum_{j=1}^{n} \delta(x_{kj},x_{lj}), \qquad \delta(a,b) = \begin{cases} 0, & a = b,\\ 1, & a \neq b, \end{cases} \tag{10.10}$$

Algorithm 10.3 Fuzzy c-medoids (FCMdd, linear and trimmed variants)

Input: dissimilarity matrix $[r(\mathbf{x}_k, \mathbf{x}_l)]$, c, $m > 1$, number of candidates p (or all), trimming share τ

choose c distinct initial medoids (random objects or D^2-weighted seeding);
repeat
  memberships μ_{ik} by (10.8);
  for $i = 1, \dots, c$ **do**
    candidates C_i: the p objects with the largest μ_{ik} (all objects if p is not given);
    cost of each candidate l: the sum of the $K = \lceil (1-\tau)N \rceil$ smallest $\mu_{ik}^m r(\mathbf{x}_k, \mathbf{x}_l)$;
    new medoid: the cheapest candidate not taken by another cluster (10.9);
until *the medoids do not change (or* $\max_{ik} |\Delta\mu_{ik}| < \epsilon$*)*;

Output: medoid indices, partition $\mathbf{U}$, objective J

In a nutshell Fuzzy c-medoids

- Prototypes are objects; works with any dissimilarity (no Euclidean assumption, no spread) and returns representative objects that can be shown.
- The linear variant searches for the medoid among the p most typical members: $O(cpN)$ instead of $O(cN^2)$ per iteration, at a small risk of a worse local minimum.
- Trimming the share τ of worst-fitting objects in the medoid update makes the medoids robust to outliers.
- Parameters c, m, p, and τ; restarts are recommended because the medoids are discrete.

and the prototype of a cluster is a *mode*, a record $\mathbf{v}_i \in \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ that need not occur in the data. Huang [144] built k-modes on this pair; Huang and Ng [145] minimize the fuzzy functional

$$J = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \, \delta(\mathbf{x}_k, \mathbf{v}_i). \tag{10.11}$$

For fixed modes the memberships follow from the FCM formula with δ in the role of the squared distance ($\mu_{ik} = 1$ for a record identical to mode i). For fixed memberships the functional separates into one term per cluster and attribute, $\sum_k \mu_{ik}^m \delta(x_{kj}, v_{ij})$, and by the theorem of Huang and Ng each term is minimized by

Algorithm 10.4 Fuzzy k-modes

Input: categorical records $\mathbf{x}_k$, c, $m > 1$ (close to one), ϵ

choose c distinct records as initial modes; memberships from (10.10);
repeat
  for *every cluster i and attribute j* **do**
    $v_{ij} \leftarrow$ the category with the largest membership mass $\sum_{k:\, x_{kj}=a} \mu_{ik}^m$ (10.12);
  matching dissimilarities $\delta(\mathbf{x}_k, \mathbf{v}_i)$ and memberships by the FCM formula;
until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: modes $\mathbf{v}_i$ (with the original category labels), partition $\mathbf{U}$, objective J

In a nutshell Fuzzy k-modes

- Fuzzy c-means for categorical records: simple matching dissimilarity and modes chosen by membership mass.
- The modes are typical records and read as a table of group profiles.
- The integer-valued dissimilarity makes the partition fuzzy; keep m small (about 1.1–2).
- Linear in the number of records; restarts guard against local minima.

the category of attribute j that carries the largest membership mass in cluster i,

$$v_{ij} = a_j^{(r)} \quad \text{with} \quad \sum_{k:\, x_{kj}=a_j^{(r)}} \mu_{ik}^m \;\geq\; \sum_{k:\, x_{kj}=a_j^{(t)}} \mu_{ik}^m \quad \text{for all } t. \tag{10.12}$$

This is "the most frequent category" with every record counted with its weight μ_{ik}^m (fig. 10.3, right). Ties are broken in favor of the first category in a fixed order. Because δ is integer-valued, many records are equally far from several modes, and the membership ratio of a record that differs in a attributes from one mode and in b from another is $(b/a)^{1/(m-1)}$. For $a = 1$ and $b = 2$ the ratio is 4 at $m = 1.5$ but only 1.4 at $m = 3$. The partition is therefore much fuzzier than that of FCM on continuous data at the same m, and values of m close to one are appropriate. The update is $O(cNn)$ per iteration, linear in the number of records, which made k-modes popular for large categorical data sets.

10.2.5 Fuzzy k-prototypes

Mixed records have numeric attributes $j \in \mathcal{N}$ and categorical attributes $j \in \mathcal{C}$. Huang's k-prototypes dissimilarity [144] treats each kind in its own terms and

Algorithm 10.5 Fuzzy k-prototypes

Input: mixed records, numeric attributes $\mathcal{N}$ and categorical attributes $\mathcal{C}$, c, $m > 1$, weight γ (default: mean standard deviation of the numeric attributes)

choose c distinct records as initial prototypes; memberships from (10.13);
repeat
  numeric part: $v_{ij} = \sum_k \mu_{ik}^m x_{kj} / \sum_k \mu_{ik}^m$ for $j \in \mathcal{N}$;
  categorical part: membership-mass mode (10.12) for $j \in \mathcal{C}$;
  dissimilarities (10.13) and memberships by the FCM formula;
until $\max_{ik} \left| \mu_{ik}^{(l)} - \mu_{ik}^{(l-1)} \right| < \epsilon$;

Output: prototypes (means and modes), partition $\mathbf{U}$, objective J

adds them with a weight $\gamma \geq 0$,

$$d(\mathbf{x}_k, \mathbf{v}_i) = \sum_{j \in \mathcal{N}} (x_{kj} - v_{ij})^2 + \gamma \sum_{j \in \mathcal{C}} \delta(x_{kj}, v_{ij}), \tag{10.13}$$

and the fuzzy version minimizes $J = \sum_i \sum_k \mu_{ik}^m d(\mathbf{x}_k, \mathbf{v}_i)$ by the FCM membership formula and a prototype update that combines the two previous cases: the numeric part of $\mathbf{v}_i$ is the fuzzy mean $v_{ij} = \sum_k \mu_{ik}^m x_{kj} / \sum_k \mu_{ik}^m$ ($j \in \mathcal{N}$), and the categorical part is the membership-mass mode (10.12) ($j \in \mathcal{C}$). Both updates are optimal for their part, so the functional decreases monotonically.

The weight γ states how many units of squared numeric distance one categorical mismatch is worth. With $\gamma = 0$ the method is FCM on the numeric attributes; as $\gamma \to \infty$ the categorical part dominates and the method becomes fuzzy k-modes on the categorical attributes. Huang relates the scale of γ to the average standard deviation of the numeric attributes, and this average is the default γ of the implementation; the numeric attributes should be standardized first, so that this scale is meaningful and no attribute dominates through its unit. The prototypes are hybrid records, with means for the measurements and modes for the categories, and remain interpretable.

10.2.6 Dissimilarities for objects that are not vectors

The relational algorithms are only as good as the dissimilarity they receive. Three constructions cover common cases; each also produces the rectangular matrix of new objects against the training objects that the estimators need for prediction.

Dynamic time warping. Two series $\mathbf{x} = (x_1, \ldots, x_T)$ and $\mathbf{y} = (y_1, \ldots, y_T)$ of the same shape but with events at slightly different times are far apart in the Euclidean sense, which compares x_t with y_t. Dynamic time warping [147] compares each

In a nutshell **Fuzzy k-prototypes**

- Fuzzy clustering of mixed records: squared Euclidean distance on the numeric part, matching dissimilarity on the categorical part, weighted by γ.
- Interpolates between FCM on the numeric attributes ($\gamma = 0$) and fuzzy k-modes on the categorical ones ($\gamma \to \infty$).
- Default γ: the mean standard deviation of the numeric attributes; standardize the numeric attributes first.
- Prototypes are records with means and modes; linear in N.

x_t with a matching y_s instead. A warping path is a sequence of index pairs from $(1, 1)$ to (T, T) that advances by one in t, in s, or in both at every step, and the DTW distance is the square root of the smallest summed local cost along such a path. Dynamic programming computes it through the accumulated cost

$$\begin{gathered} g(t, s) = (x_t - y_s)^2 + \min\{g(t-1, s),\ g(t, s-1),\ g(t-1, s-1)\}, \\ \mathrm{DTW}(\mathbf{x}, \mathbf{y}) = \sqrt{g(T, T)}, \end{gathered} \tag{10.14}$$

with $g(0, 0) = 0$ and $g = \infty$ outside the admissible region. The Sakoe–Chiba band $|t - s| \leq w$ restricts the path to the neighborhood of the diagonal: it limits the warping to plausible displacements, prevents pathological alignments that map one point to a long stretch, and reduces the cost from $O(T^2)$ to $O(Tw)$ per pair. With $w = 0$ DTW is the Euclidean distance. DTW is not a metric—the triangle inequality can fail—and its squared matrix is in general not Euclidean, so fuzzy c-medoids (or NERF) is the appropriate partner.

Autocorrelation-based distance. When the series are realizations of stochastic processes, their values cannot be compared point by point, but their dependence structure can. D'Urso and Maharaj [148] represent every series by its estimated autocorrelations up to lag L,

$$\hat{\rho}_k(l) = \frac{\sum_{t=l+1}^{T} (x_{k,t} - \bar{x}_k)(x_{k,t-l} - \bar{x}_k)}{\sum_{t=1}^{T} (x_{k,t} - \bar{x}_k)^2}, \qquad l = 1, \ldots, L, \tag{10.15}$$

and compare series by the Euclidean distance of these vectors. The implementation adds optional lag weights w_q (all equal to one by default),

$$d_{\mathrm{ACF}}(\mathbf{x}_k, \mathbf{x}_l) = \Big(\sum_{q=1}^{L} w_q \big(\hat{\rho}_k(q) - \hat{\rho}_l(q) \big)^2 \Big)^{1/2}. \tag{10.16}$$

The representation ignores level, scale, and phase, has the same length for series of any length, and identifies stationary processes of low order by a few lags. D'Urso and Maharaj cluster the vectors with fuzzy c-means (A-FCM); fuzzy c-medoids on

Algorithm 10.6 Autocorrelation-based fuzzy clustering of time series (A-FCM, A-FCMd)

1. Estimate the autocorrelations $\hat{\rho}_k(1), \dots, \hat{\rho}_k(L)$ of every series by (10.15).
2. A-FCM: run fuzzy c-means on the vectors $(\sqrt{w_q}\,\hat{\rho}_k(q))_{q=1}^{L}$; equivalently RFCM on the matrix of squared distances (10.16).
3. A-FCMd: run fuzzy c-medoids on the squared distances (10.16); the medoids are representative series.
4. Assign new series through their autocorrelation vectors (A-FCM) or their distances to the training series (A-FCMd).

In a nutshell Autocorrelation-based clustering

- Clusters series by their dependence structure, not their values: invariant to level, scale, and phase, and applicable to series of different lengths.
- A Euclidean representation: FCM, RFCM, and fuzzy c-medoids all apply directly.
- L must cover the lags at which the processes differ, but every extra lag adds estimation noise of order $1/\sqrt{T}$; decaying weights w_q make a generous L safe.
- Meant for (approximately) stationary series; for shapes and events use DTW instead.

the squared distances (A-FCMd) returns representative series instead of centers. Because the representation is a feature vector, the squared matrix of (10.16) is Euclidean, and RFCM on it reproduces A-FCM. Two choices remain: the number of lags L, which must reach the lags at which the processes differ, and the weights w_q, which can damp the high lags whose estimates are noisy.

Gower's coefficient. Gower [149] defined a general coefficient for records with numeric and categorical attributes. In its dissimilarity form each attribute contributes a score in $[0, 1]$: the range-scaled absolute difference $|x_{kj} - x_{lj}|\,/R_j$ for a numeric attribute with range R_j, and the mismatch indicator for a categorical one. The dissimilarity is the weighted mean of the scores over the attributes observed in both records,

$$d_G(\mathbf{x}_k, \mathbf{x}_l) = \frac{\sum_j w_j\,\delta_{klj}\,s_{klj}}{\sum_j w_j\,\delta_{klj}}, \qquad \delta_{klj} = \begin{cases} 1, & x_{kj} \text{ and } x_{lj} \text{ observed}, \\ 0, & \text{otherwise}, \end{cases} \tag{10.17}$$

so missing values need no imputation. By construction every attribute contributes on the same scale, and the weights w_j play the role of γ in k-prototypes. Gower showed that the corresponding similarity matrix is positive semi-definite when no values are missing, so that $\sqrt{d_G}$ is a Euclidean distance [146, 149]; the matrix $[d_G]$ itself, read as squared distances, is therefore a valid input for RFCM. With missing

values this guarantee is lost, and records compared on few attributes can appear close to many others.

Table 10.1 summarizes which algorithm accepts which data.

Table 10.1. The algorithms of the chapter differ in the data they accept, in their prototypes, and in their requirements. "any **R**" means a symmetric, non-negative matrix with a zero diagonal.

algorithm	input	prototype	requirement
RFCM	**R** (squared)	object weights	**R** Euclidean
NERF c-means	any **R**	object weights	none (see β)
FCMdd	any **R**	an object	none
fuzzy k-modes	categorical	mode	none
fuzzy k-prototypes	mixed	means and modes	a sensible γ
A-FCM / A-FCMd	time series	ACF vector / series	stationarity

10.3 Using it in fuzzyclust

Everything in this chapter is in `fuzzyclust.relational`. The five estimators derive from `BaseFuzzyClusterer` and have the usual methods `fit`, `predict`, `predict_proba`, `transform`, and `score`, so they work in scikit-learn pipelines and with the validity functions of chapter 4. The relational estimators follow scikit-learn's convention for precomputed matrices: with `metric="precomputed"`, `fit` takes the square $N \times N$ matrix and `predict_proba` the rectangular matrix of the new objects against the N training objects. Any other `metric` names a SciPy distance, and the matrix is computed from feature vectors; the distances are squared for RFCM and NERF, as in FCM, and used as they are for fuzzy c-medoids.

`fuzzyclust.relational.RelationalFCM`

`n_clusters=3`, `metric="euclidean"|"precomputed"|any SciPy metric`, `m=2.0`, `tol=1e-4`, `max_iter=300`, `init="random"|"kmeans++"|partition|object indices`, `n_init=1`, `random_state`, `verbose`. Attributes `relational_centers_` ($\mathbf{v}_i \in \mathbb{R}^N$), `cluster_centers_` (dual centers $\sum_l v_{il}\mathbf{x}_l$ with a feature metric, else the relational centers), `dissimilarity_matrix_`, `membership_`, `labels_`, `distances_` ($\sqrt{d_{ik}^2}$), `objective_`, `objective_history_`, `n_iter_`. Raises an error on a negative relational distance.

fuzzyclust.relational.NERFCM

Same parameters and attributes as `RelationalFCM`. Here `beta_` holds the final spread β and `n_spread_updates_` the number of times β was raised (both are always zero for `RelationalFCM`); the objective is that of $\mathbf{R}_\beta$.

fuzzyclust.relational.FuzzyCMedoids

`n_clusters=3`, `metric="euclidean"|"precomputed"|any SciPy metric` (dissimilarities used as they are, not squared), `n_candidates=None` (all objects) or p (linear FCMdd), `trim=0.0` (τ, FCTMdd), `m=2.0`, `tol=1e-4`, `max_iter=100`, `init="kmeans++"|"random"|index array`, `n_init=1`, `random_state`, `verbose`.
Attributes `medoid_indices_`, `cluster_centers_` (medoid rows of $\mathbf{X}$, or of $\mathbf{R}$ when precomputed), `dissimilarity_matrix_`, `membership_`, `labels_`, `distances_` (dissimilarities to the medoids), `objective_`, `objective_history_`, `n_iter_`.

fuzzyclust.relational.FuzzyKModes, FuzzyKPrototypes

`FuzzyKModes(n_clusters=3, m=1.5, tol, max_iter, init="random", n_init, random_state, verbose)`; all columns categorical (strings, categories, booleans, or integer codes).
`FuzzyKPrototypes(n_clusters=3, gamma=None, categorical_features=None, m=2.0, ...)`; `gamma=None` uses the mean standard deviation of the numeric columns; `categorical_features=None` detects the categorical columns from the dtypes of a DataFrame; alternatively, pass `"all"`, a boolean mask, or column names or positions.
Attributes `modes_` and `prototypes_` (with the original labels; DataFrames for DataFrame input), `numeric_centers_`, `cluster_centers_` (encoded), `categorical_features_`, `numeric_features_`, `categories_`, `gamma_`, `membership_`, `labels_`, `objective_`. Unknown categories of new records count as one mismatch.

dissimilarities, generators, and plot helpers

`pairwise_dissimilarity(X, Y=None, metric)` (any SciPy metric), `dtw_distance_matrix(X, Y=None, window)` and `dtw_alignment(x, y, window)` → distance, path; `autocorrelation_features(X, n_lags)`, `autocorrelation_distance_matrix(X, Y=None, n_lags, weights)`; `gower_distance_matrix(X, Y=None, categorical_features, ranges, feature_weights)`; `matching_dissimilarity(A, B)`; `relational_distances(R, V, beta)`; `check_dissimilarity_matrix(R)`. Generators `make_categorical_blobs`, `make_mixed_blobs`,

`make_warped_series`. Plots `plot_dissimilarity_matrix(R, labels)`, `plot_dtw_alignment(x, y)`, `plot_modes(model, X)`, `plot_series_clusters(X, labels, medoids)`.

The duality of RFCM and FCM, and NERF on a matrix that RFCM rejects:

```
R = pairwise_dissimilarity(X, metric="sqeuclidean")  # squared Euclidean
    distances
U0 = np.random.RandomState(0).rand(N, 3)             # a common random
    start
fcm = FCM(n_clusters=3, init=U0).fit(X)
rfcm = RelationalFCM(n_clusters=3, metric="precomputed", init=U0).fit(R)
print(f"max |U_RFCM - U_FCM| = {np.abs(rfcm.membership_ -
    fcm.membership_).max():.1e}, "
      f"iterations {rfcm.n_iter_} vs {fcm.n_iter_}, "
      f"J = {rfcm.objective_:.4f} vs {fcm.objective_:.4f}")
dual_centers = rfcm.relational_centers_ @ X
print("dual centers - FCM centers:", np.abs(dual_centers -
    fcm.cluster_centers_).max())
```

```
R_l1 = pairwise_dissimilarity(X, metric="cityblock") ** 2   # squared L1
    distances: not Euclidean
try:
    RelationalFCM(n_clusters=3, metric="precomputed",
    random_state=0).fit(R_l1)
except ValueError as err:
    print("RFCM:", str(err)[:70], "...")
nerf = NERFCM(n_clusters=3, metric="precomputed", random_state=0).fit(R_l1)
print(f"NERF: beta = {nerf.beta_:.3f} after {nerf.n_spread_updates_}
    spread update(s), "
      f"ARI = {adjusted_rand_score(y, nerf.labels_):.2f}, {nerf.n_iter_}
    iterations")
```

Fuzzy c-medoids takes any precomputed dissimilarity, here dynamic time warping, and its linear variant needs only one more parameter, `n_candidates`:

```
R_euclid = pairwise_dissimilarity(X)                # compare values at
    equal times
t0 = time.perf_counter()
R_dtw = dtw_distance_matrix(X, window=12)           # Sakoe-Chiba band of
    12 samples
print(f"DTW matrix of {N} series of length {T}: {time.perf_counter() -
    t0:.2f} s")
```

```
fcmdd_dtw = FuzzyCMedoids(n_clusters=3, metric="precomputed", n_init=5,
    random_state=0).fit(R_dtw)
fcmdd_euc = FuzzyCMedoids(n_clusters=3, metric="precomputed", n_init=5,
    random_state=0).fit(R_euclid)
for name, model in (("DTW", fcmdd_dtw), ("Euclidean", fcmdd_euc)):
    print(f"{name:9s}: ARI = {adjusted_rand_score(y, model.labels_):.2f}, "
```

```
          f"medoids = {model.medoid_indices_.tolist()}, J =
    {model.objective_:.2f}, "
          f"{model.n_iter_} iterations")
```

Categorical and mixed records are clustered directly from a DataFrame:

```
km = FuzzyKModes(n_clusters=3, m=1.5, n_init=5, random_state=0).fit(X)
print(km.modes_)                      # DataFrame with the original
    categories
print(f"ARI = {adjusted_rand_score(y, km.labels_):.2f}, {km.n_iter_}
    iterations, J = {km.objective_:.1f}")
```

```
kp = FuzzyKPrototypes(n_clusters=3, n_init=5, random_state=0).fit(X)
print(f"gamma = {kp.gamma_:.2f}, categorical columns:
    {list(X.columns[kp.categorical_features_])}")
print(kp.prototypes_.round(2))
print(f"ARI = {adjusted_rand_score(y, kp.labels_):.2f}")
```

Gower's coefficient and the autocorrelation distance turn mixed records and time series into relational data:

```
G = gower_distance_matrix(X)                          # categorical
    columns detected from the dtypes
onehot = pd.get_dummies(X, columns=cat_cols, dtype=float)
print(f"Gower matrix {G.shape}, entries in [{G.min():.2f}, {G.max():.2f}];
    one-hot data {onehot.shape}")
```

```
A = autocorrelation_features(X, n_lags=5)                    # (75, 5)
    feature vectors
D2 = autocorrelation_distance_matrix(X, n_lags=5) ** 2        # squared
    ACF distances
afcm = FCM(n_clusters=3, n_init=5, random_state=0).fit(A)
               # A-FCM
afcmd = FuzzyCMedoids(n_clusters=3, metric="precomputed", n_init=5,
    random_state=0).fit(D2)  # A-FCMd
rfcm = RelationalFCM(n_clusters=3, metric="precomputed",
    init=afcm.membership_).fit(D2)
raw = FuzzyCMedoids(n_clusters=3, metric="precomputed", n_init=5,
    random_state=0).fit(pairwise_dissimilarity(X))
for name, model in (("A-FCM", afcm), ("A-FCMd", afcmd), ("RFCM on ACF",
    rfcm), ("raw values", raw)):
    print(f"{name:12s}: ARI = {adjusted_rand_score(y, model.labels_):.2f}")
print(f"max |U_RFCM - U_A-FCM| = {np.abs(rfcm.membership_ -
    afcm.membership_).max():.1e}")
print("medoid series of A-FCMd:", afcmd.medoid_indices_.tolist())
```

10.4 Examples

10.4.1 RFCM as the dual of FCM, and NERF

Three Gaussian groups of 100 points are clustered by FCM on the coordinates and by RFCM on the matrix of squared Euclidean distances, both from the same random partition (fig. 10.4). The memberships differ by at most $2 \cdot 10^{-15}$, both runs need eight iterations, the objective trajectories coincide, and the dual centers equal the FCM centers to machine precision. The relational prototypes (fig. 10.5) are weight vectors over the 300 objects, nearly uniform on the members of their cluster and close to zero elsewhere; the ordered dissimilarity matrix shows the block structure the algorithm exploits. The squared city-block distances of the same points are not Euclidean: RFCM stops at a negative relational distance, while NERF raises β once (to 1.5) and recovers the three groups exactly. The parameter study (fig. 10.6) feeds powers p of the Euclidean distances to NERF. Up to $p = 2$ the matrix is Euclidean and $\beta = 0$; beyond that the spread grows to 0.21 of the mean dissimilarity at $p = 3$ and to 0.63 at $p = 5$, and the mean maximal membership falls from 0.96 to 0.57. The spread flattens the partition, but the hard labels stay correct. (Below $p = 2$ the partition is softer for a different reason: a small power compresses the ratio of far to near distances.) The printed study shows the usual effect of m, and that both initializations find the groups; their objectives differ because the runs end with different β.

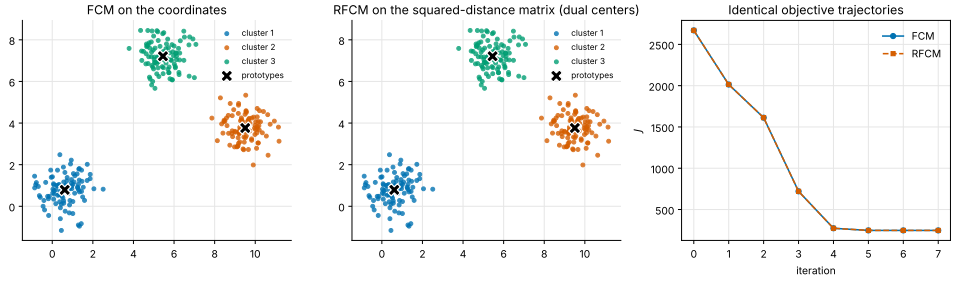


Figure 10.4. FCM on the coordinates (left) and RFCM on the matrix of squared distances with its dual centers (middle) give the same partition; the objective trajectories coincide (right).

10.4.2 Fuzzy c-medoids on dynamic time warping

Three shape classes (a bell, a step, and a double bump) are generated with random time warps and shifts, 25 series of length 60 per class (fig. 10.7). In the Euclidean matrix the class blocks are blurred, because shifted peaks produce large distances within a class; in the DTW matrix with a band of $w = 12$ samples they are clear, and the alignment of two bells reduces their dissimilarity from 1.73 to 0.65 (fig. 10.8). Fuzzy c-medoids with five restarts reaches an adjusted Rand index (ARI) of 0.92 on the DTW matrix and 0.49 on the Euclidean one, whose clusters group the series by *when* the peak occurs rather than by its shape; the medoids of the DTW clustering are typical, well-centered members of the classes (fig. 10.9). The memberships

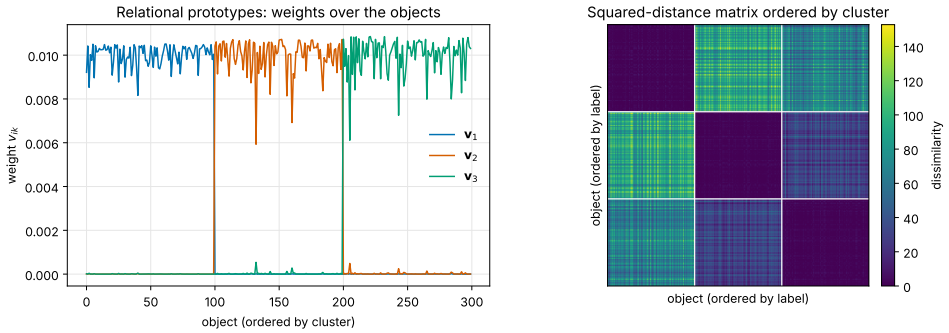


Figure 10.5. The relational prototypes are weight vectors over the objects (left); the squared-distance matrix ordered by cluster shows three blocks (right).

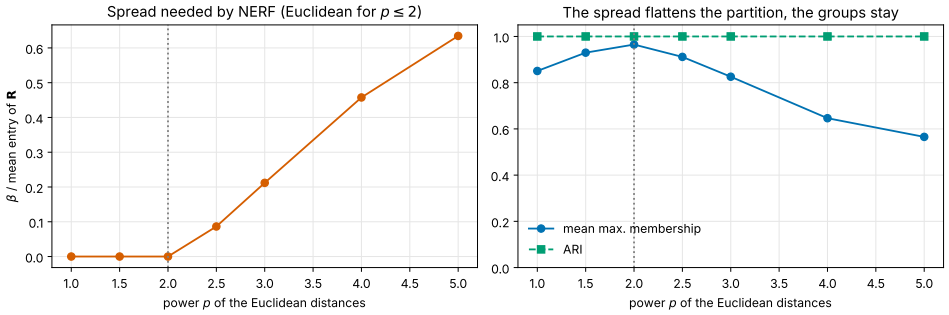


Figure 10.6. NERF on powers p of the Euclidean distances. Left: the final spread relative to the mean dissimilarity is zero up to $p = 2$ and grows beyond it. Right: the partition flattens with the spread while the groups are kept.

(fig. 10.10) are crisp for the steps and shared between bells and double bumps, whose warped versions can resemble each other. On 360 series, the candidate study (fig. 10.11) shows that one or two candidates leave the search in clearly worse optima ($J = 186$ and 176), five come within 1.5 % of the full search ($J = 170.7$ against 168.5), and fifty reach it in about a quarter of the time. The printed band study shows the ARI rising from 0.49 at $w = 0$ (Euclidean) and $w = 4$ through 0.85 at $w = 8$ to 1.0 at $w = 20$ and without a band: the band must cover the displacements in the data. The medoids hardly move with m, while the mean maximal membership falls from 0.95 at $m = 1.3$ to 0.52 at $m = 3$.

10.4.3 Fuzzy k-modes on survey data

Three hundred respondents answer six questions with four possible answers each. Each of three groups has a typical answer pattern, and every answer is replaced by a uniformly random one with probability 0.25. Fuzzy k-modes with $m = 1.5$ recovers the three planted modes exactly (ARI 0.98), and the table of modes (fig. 10.12) reads directly as a table of group profiles; the shade of a cell is the share of the membership mass on the modal answer, between 75 and 85 %. Most respondents

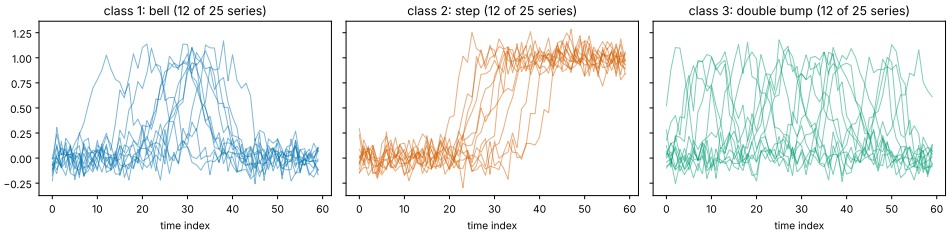


Figure 10.7. Three shape classes with random time warps and shifts: a bell, a step, and a double bump.

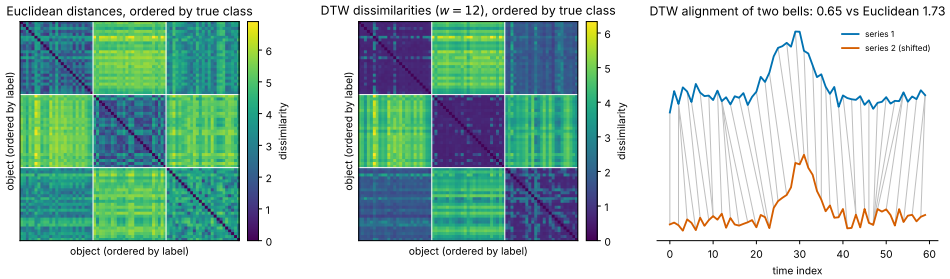


Figure 10.8. Euclidean (left) and DTW (middle) dissimilarities ordered by the true class, and the DTW alignment of two bells (right): the class blocks are clear only under DTW.

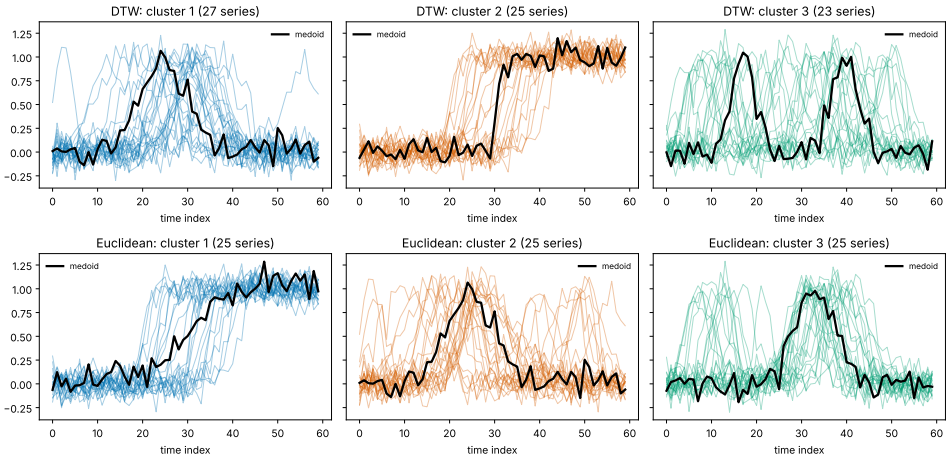


Figure 10.9. Fuzzy c-medoid clusters on the DTW matrix (top) and on the Euclidean matrix (bottom), with the medoid series in black.

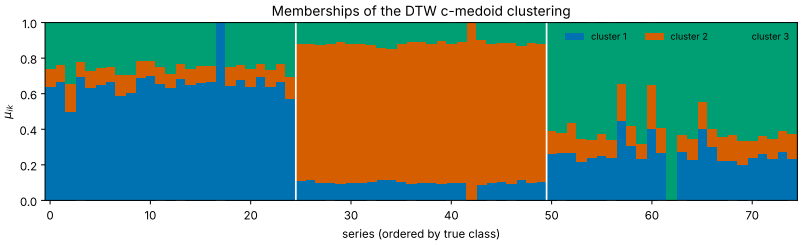


Figure 10.10. Memberships of the DTW c-medoid clustering, with the series ordered by the true class.

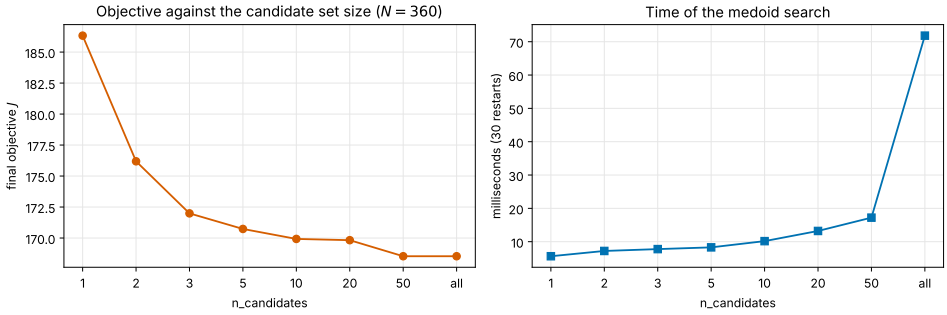


Figure 10.11. Linear fuzzy c-medoids: final objective (left) and time of thirty restarts (right) against the number of medoid candidates.

differ from their own mode in one or two answers. Figure 10.13 illustrates the mode theorem for one question: the modal answer of each cluster is the one with the largest membership mass. The exponent study (fig. 10.14) confirms the sensitivity predicted by the integer dissimilarity: the planted modes are found up to $m = 2.5$, while the mean maximal membership falls from 0.99 to 0.68; at $m = 3$ the modes drift and the ARI drops to 0.72, and at $m = 4$ to 0.41. The printed noise study shows that the modes are recovered even when 65 % of the answers are replaced at random, although the assignment of individual respondents degrades (ARI 0.45). The objective against c (fig. 10.15) has a clear knee at three groups.

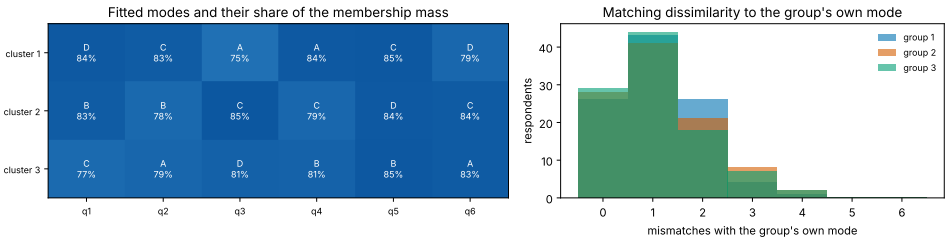


Figure 10.12. Fitted modes with the share of the membership mass on the mode (left) and the matching dissimilarity of the respondents to their group's mode (right).



Figure 10.13. Membership mass of the answers to question q1 in each cluster; the mode is the answer with the largest mass.

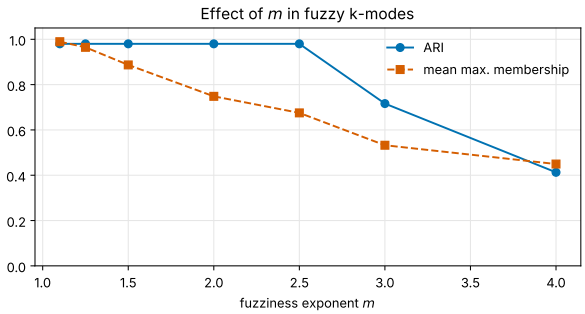


Figure 10.14. Agreement with the groups and mean maximal membership of fuzzy k-modes against the exponent m.

10.4.4 Fuzzy k-prototypes and the weight γ

Mixed records with two numeric and three categorical attributes are generated so that neither part separates the three groups alone: the numeric centers are three units apart with unit spread, and every categorical entry is replaced by a random category with probability 0.25 (fig. 10.16). Fuzzy k-prototypes with the default $\gamma = 1.64$ (the mean standard deviation of the numeric columns) reaches an ARI of 0.94 and returns prototypes with means and modes (fig. 10.17). The γ

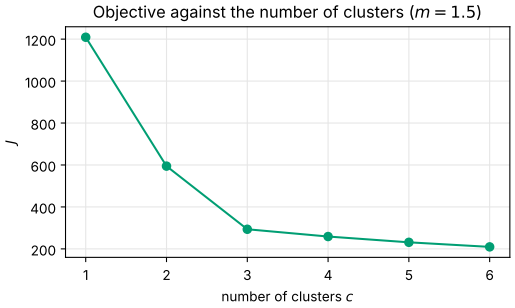


Figure 10.15. The objective of fuzzy k-modes against the number of clusters has its knee at the three planted groups.

study (fig. 10.18) traces the interpolation: at $\gamma = 0$ the partition is that of FCM on the numeric columns (agreement 1.0, ARI 0.75), for $\gamma = 100$ it approaches fuzzy k-modes on the categorical columns (agreement 0.97, ARI 0.83), and in between the two parts correct each other, with the best ARI of 0.97 at $\gamma = 5$. The default lies on the rising flank just below this optimum. The exponent softens the partition as usual (mean maximal membership 0.91, 0.76, and 0.58 for $m = 1.5$, 2, and 3) while the ARI stays at 0.93–0.94, all five random starts reach the same objective, and a new record with an unseen category is assigned with one mismatch to every mode.

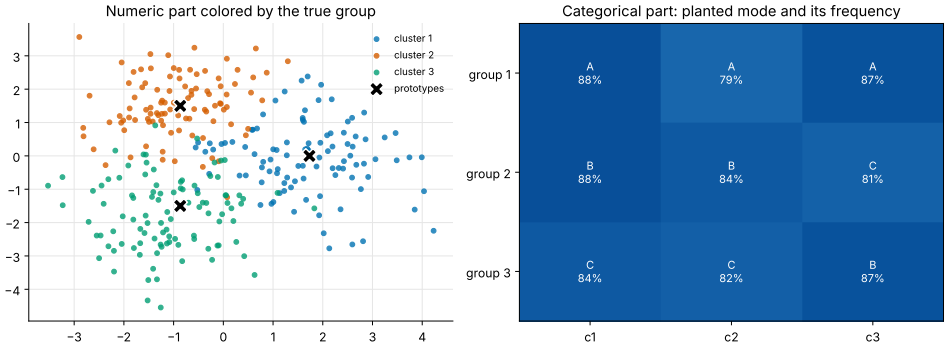


Figure 10.16. Mixed data: the numeric part colored by the true group, with the planted centers as crosses (left), and the planted modes of the categorical part with their frequencies (right).

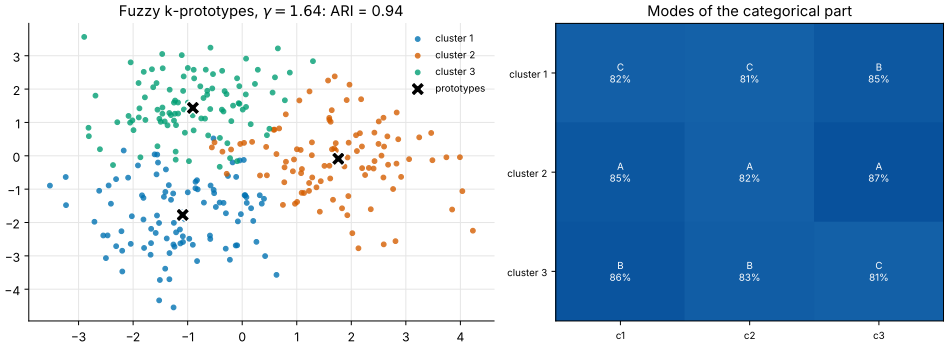


Figure 10.17. Fuzzy k-prototypes with the default weight: the numeric part of the partition with the prototypes (left) and the modes of the categorical part (right).

10.4.5 Gower's coefficient versus one-hot encoding

Data of the same kind, now with four categorical attributes whose entries are replaced at random with probability 0.3, are clustered along two routes. In the Gower matrix every attribute counts equally; in the Euclidean distance of the one-hot encoded records the two numeric columns dominate the twelve indicator

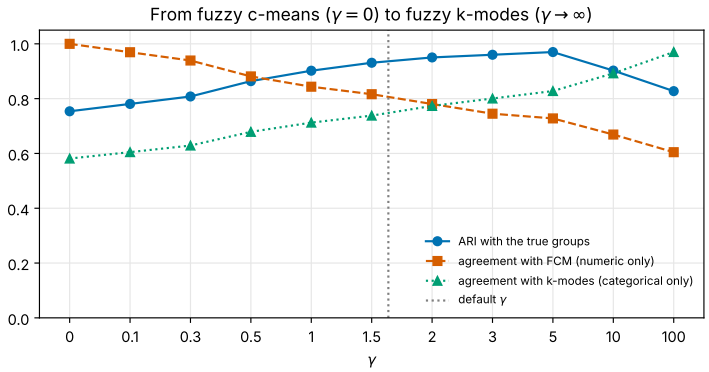


Figure 10.18. The weight γ interpolates between fuzzy c-means on the numeric part ($\gamma = 0$) and fuzzy k-modes on the categorical part ($\gamma \to \infty$); the combination outperforms both.

columns (fig. 10.19). On these complete data the routes agree: Gower with fuzzy c-medoids reaches an ARI of 0.89, Gower with RFCM 0.90, one-hot FCM 0.91, and k-prototypes 0.92 (fig. 10.20); the difference is that the medoids are real records. NERF on the Gower matrix never needs a spread ($\beta = 0$), as Gower's theorem predicts. Scaling the weight of the numeric attributes (fig. 10.21) traces the same path as γ: zero weight (categories only) gives an ARI of about 0.8, equal weights 0.89–0.90, a moderately larger weight up to 0.97, and a weight of 100 (practically numeric only) falls back to 0.78. Gower's coefficient is defined without imputation, and in the printed missing-value study NERF on it keeps an ARI of 0.86 with 10 % and 0.84 with 20 % missing entries. Fuzzy c-medoids, however, collapses to about 0.45 from 10 % on, because a record compared on only two or three attributes looks close to many others and is chosen as a medoid; the medoids carry up to four missing entries out of six. The one-hot route has to discard incomplete records and keeps its accuracy only on the shrinking remainder (165 of 300 records at 10 %, 34 at 30 %).

10.4.6 Autocorrelation-based clustering of time series

Twenty-five series of length 200 are drawn from each of three processes: an AR(1) process with $\phi = 0.5$, an MA(1) process with $\theta = 0.9$, and an AR(1) process with $\phi = -0.6$ (fig. 10.22). The first two have practically the same lag-1 autocorrelation (0.5 against 0.497) and differ from lag 2 on; the estimated autocorrelation functions scatter around the theoretical ones by about $1/\sqrt{T}$. With $L = 5$ lags, A-FCM reaches an ARI of 0.88 and A-FCMd 0.85, RFCM on the squared autocorrelation distances reproduces A-FCM, and fuzzy c-medoids on the Euclidean distances of the raw values finds no structure (ARI 0.02), because independent realizations of one process are as far apart as realizations of different processes. In the plane of the first two autocorrelations (fig. 10.23) the alternating class lies far from the

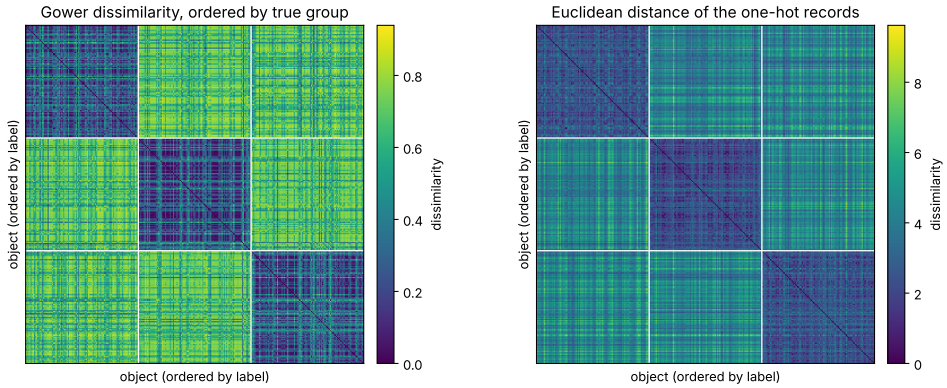


Figure 10.19. Gower's dissimilarity (left) and the Euclidean distance of the one-hot encoded records (right), ordered by the true group.

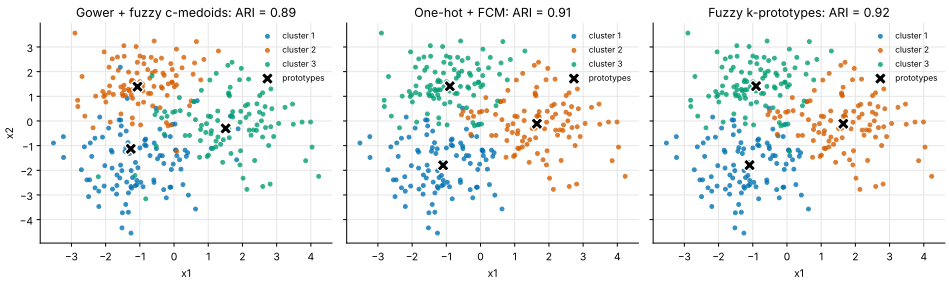


Figure 10.20. Gower with fuzzy c-medoids (left, medoids as crosses), one-hot encoding with FCM (middle), and fuzzy k-prototypes (right), shown in the numeric plane.

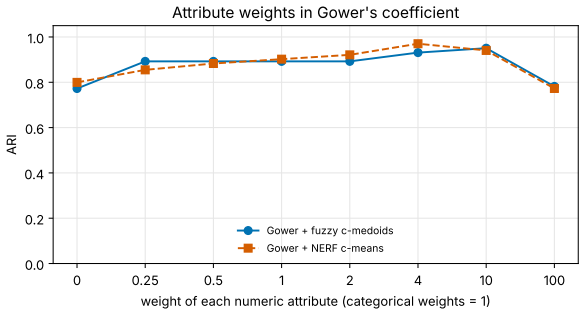


Figure 10.21. Agreement with the groups against the weight of the numeric attributes in Gower's coefficient, for fuzzy c-medoids and NERF c-means.

others, while the AR(1) and MA(1) classes are separated only along $\hat{\rho}(2)$, and their memberships are correspondingly shared. New AR(1) series with ϕ swept from -0.9 to 0.9 (fig. 10.24) receive graded memberships: the alternating cluster dominates for negative ϕ, the three clusters share the membership near white noise, the MA(1) cluster peaks at $\phi \approx 0.2$–0.3, and the persistent cluster takes over toward $\phi = 0.5$. The parameter study (fig. 10.25) shows the roles of L and T. With one lag the AR(1) and MA(1) classes are merged (ARI at most 0.55 for every length), from two lags on they are separated, and beyond a few lags every additional coordinate adds estimation noise, flattens the memberships, and, for short series, lowers the ARI. Longer series improve the result throughout; with $T = 400$ the ARI reaches 0.96. The printed study runs A-FCMd with $L = 20$: uniform weights give an ARI of 0.70, and decaying lag weights $w_q = 0.8^{\,q-1}$ raise it to 0.88, slightly above the 0.85 of A-FCMd with five lags.

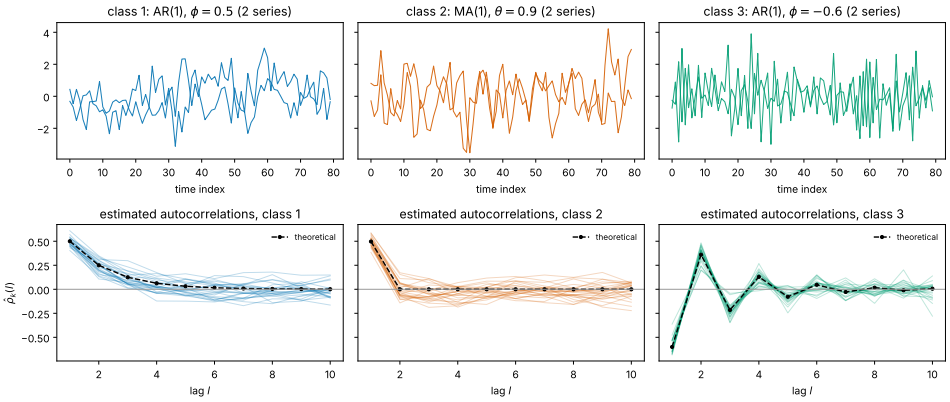


Figure 10.22. Two series of each process (top) and the estimated autocorrelation functions of all series of a class with the theoretical one (bottom).

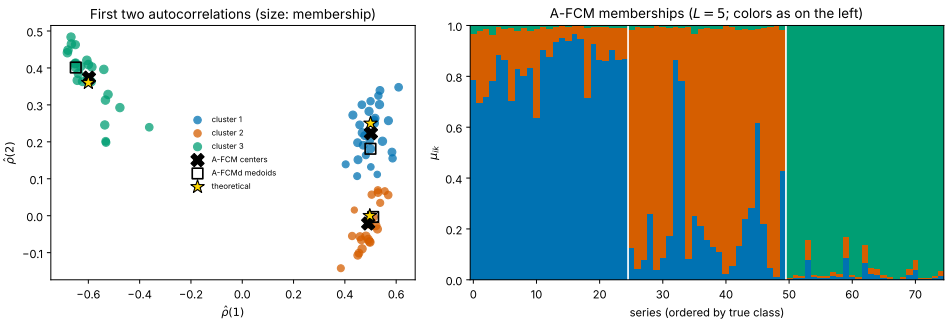


Figure 10.23. A-FCM in the plane of the first two autocorrelations, with centers, A-FCMd medoids, and the theoretical values (left), and the memberships ordered by the true class (right).

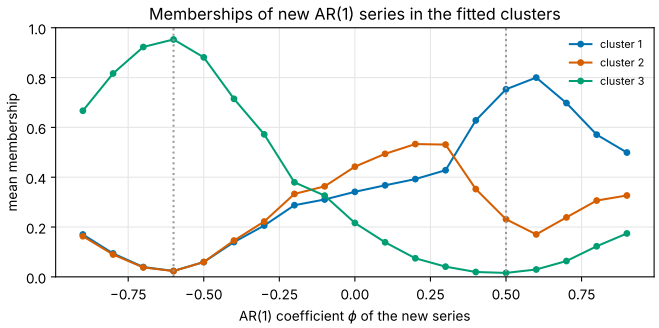


Figure 10.24. Mean memberships of new AR(1) series in the fitted clusters against their coefficient ϕ.

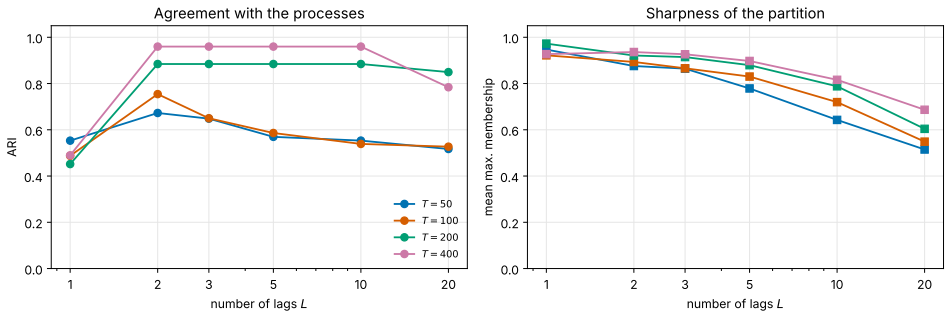


Figure 10.25. Agreement with the processes (left) and mean maximal membership (right) against the number of lags L for four series lengths T.

10.5 Further reading

Relational fuzzy c-means and its duality with FCM are due to Hathaway, Davenport, and Bezdek [141], the non-Euclidean extension with the β-spread to Hathaway and Bezdek [142]; both rest on the c-means functional of Bezdek [3], and the relational methods are surveyed in the broader context of fuzzy pattern recognition in [150]. The Euclidean property of dissimilarity matrices, the double centering, and the effect of additive constants are treated thoroughly by Gower and Legendre [146]. Fuzzy c-medoids, its linear-complexity candidate search, and the robust variant RFCMdd are due to Krishnapuram et al. [143]; the crisp medoid methods PAM and CLARA and the fuzzy relational method FANNY are described in the book of Kaufman and Rousseeuw [5], which also uses Gower's coefficient [149] for mixed data. Huang introduced k-modes and k-prototypes in [144], and Huang and Ng the fuzzy k-modes algorithm and the theorem on the fuzzy mode in [145]. Dynamic time warping with the band constraint goes back to Sakoe and Chiba [147], the autocorrelation-based fuzzy clustering of time series to D'Urso and Maharaj [148]. The geodesic c-medoids of chapter 7 and the kernel fuzzy c-means of chapter 6 are close relatives of the methods of this chapter: both replace the Euclidean geometry of FCM by one derived from pairwise relations of the samples.

CHAPTER

11



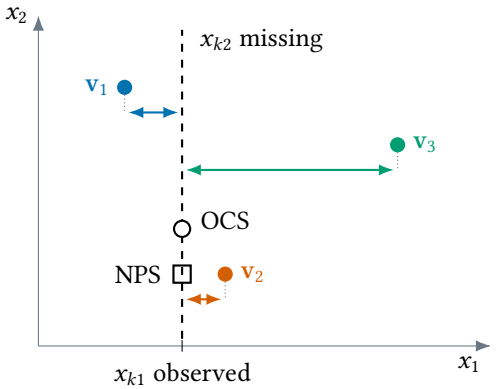
Incomplete, very large, and image data

The algorithms of the previous chapters assume a complete data matrix of moderate size whose rows are unrelated samples. Real data often violate all three assumptions. Values are *missing* because sensors fail or records are merged from sources that measure different things; data sets are *very large* or arrive as an endless stream; and the samples may be the pixels of an *image*, where neighbors almost always belong to the same region. This chapter keeps the fuzzy c-means functional and adapts it to each situation: partial distances and optimal completion for incomplete data (Hathaway and Bezdek); weighted fuzzy c-means with sampling, single-pass, and online schemes for very large data and streams (Havens, Bezdek, and co-workers); and six ways of bringing the pixel neighborhood into the functional for image segmentation. The three sub-packages `fuzzyclust.incomplete`, `fuzzyclust.large`, and `fuzzyclust.image` share the notation and the membership formula of fuzzy c-means, and the image methods reuse weighted fuzzy c-means to cluster gray-level histograms.

11.1 Problem and idea

Missing values. Fuzzy c-means measures the distance of a sample to a prototype as a sum over all features, so a single missing value makes the distance undefined. The common remedies outside clustering—dropping incomplete samples, or filling the holes with the column mean before clustering—either waste data or invent values that ignore the cluster structure: a column mean lies *between* the clusters and drags the filled samples there. Hathaway and Bezdek [151] compared four strategies for running fuzzy c-means on incomplete data (fig. 11.1). The *whole-data strategy* (WDS) drops the incomplete samples, which is defensible only when

they are few. The *partial-distance strategy* (PDS) measures the distance over the observed features only and rescales it to the full dimension, following Dixon [152]; a sample is then judged by what is known about it. The *optimal-completion strategy* (OCS) treats every missing value as an additional variable of the c-means functional and optimizes it together with the partition and the prototypes, so that the data are imputed while they are clustered; the optimal value is a membership-weighted mean of the prototype coordinates. The *nearest-prototype strategy* (NPS) is its crisp version and copies the coordinate of the nearest prototype. All four assume that the values are missing completely at random, so that the hole pattern itself carries no information about the clusters [153].



PDS: distances over the observed features, rescaled by n/I_k:
$D_{ik}^2 = \frac{2}{1}(x_{k1} - v_{i1})^2$ (arrows)
$\mu_k = (0.35,\ 0.62,\ 0.03)$ for $m = 2$

OCS: the missing value is the membership-weighted mean of the prototype coordinates,
$x_{k2} = \frac{\sum_i \mu_{ik}^2 v_{i2}}{\sum_i \mu_{ik}^2} = 1.63$ (circle)

NPS: the coordinate of the nearest prototype, $x_{k2} = v_{22} = 1.0$ (square)

Figure 11.1. Partial distances and the completion of a missing value. A sample with $x_{k1} = 2$ observed and x_{k2} missing can lie anywhere on the dashed line. The partial distance compares it with the prototypes along the observed axis only (arrows) and rescales the result to the full dimension; the memberships follow from these distances as in fuzzy c-means. Optimal completion places the missing value at the membership-weighted mean of the prototype coordinates (circle), the nearest-prototype strategy at the coordinate of the nearest prototype (square).

Very large data. Fuzzy c-means touches every sample in every iteration and stores the $N \times c$ partition. With N in the millions, or with data that arrive as a stream, both become a problem. Since the efficient implementation of Cannon, Dave, and Bezdek [154], many schemes have been proposed that either make each iteration cheaper [155] or cluster only samples or chunks of the data; Havens, Bezdek, Leckie, Hall, and Palaniswami [156] compared the latter systematically. This chapter covers three of them (fig. 11.2). *Random sample and extend* (rseFCM) clusters a random sample and computes the memberships of all samples in one pass from the fixed prototypes. The other two rest on *weighted* fuzzy c-means, in which a sample with weight w counts as w identical samples, so that a prototype standing for a thousand observations can be clustered together with raw samples. *Single-pass* FCM (spFCM) of Hore, Hall, and Goldgof [157] processes the data in chunks and carries the c prototypes of a chunk, weighted by the membership mass they collected, into the next chunk. *Online* FCM (oFCM) [158] clusters every chunk

separately and then clusters the weighted chunk prototypes. In both chunked schemes every chunk is seen once and only one chunk is held in memory, so they can also process a stream.

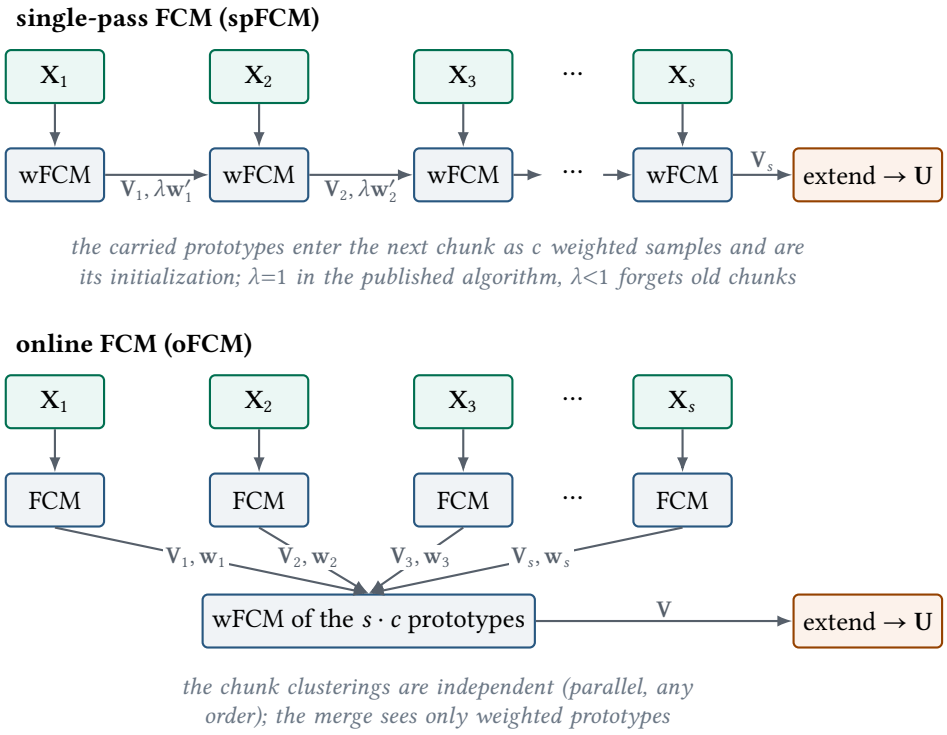


Figure 11.2. The chunk pipelines of single-pass and online FCM. Top: spFCM clusters each chunk together with the weighted prototypes of the previous one, so the prototypes accumulate the whole history. Bottom: oFCM clusters the chunks independently and merges their weighted prototypes with weighted FCM. Both obtain the memberships of all samples by one extension step from the final prototypes; rseFCM is the special case of a single sampled chunk.

Images. A gray-level image is a data set of $N = H \cdot W$ one-dimensional samples, and fuzzy c-means on the pixel values is the simplest segmentation algorithm. It ignores what distinguishes an image from an arbitrary data set: a pixel and its neighbors almost always belong to the same region. Noise that pushes a pixel value across the midpoint between two prototypes therefore produces a wrong label, however clear the surrounding region is. The spatially constrained variants bring the neighborhood N_k of every pixel into the functional (fig. 11.3). FCM_S of Ahmed et al. [159] adds the distances of the neighbors to the prototype; FCM_S1 and FCM_S2 of Chen and Zhang [160] replace them with the distance of a mean- or median-filtered pixel computed once in advance; EnFCM of Szilágyi et al. [161] forms a smoothed image once and clusters its gray-level *histogram*, with iterations whose cost does not depend on the image size; FGFCM of Cai, Chen, and Zhang

[162] does the same with locally adaptive weights that respect edges and ignore outliers; and FLICM of Krinidis and Chatzis [163] adds a parameter-free fuzzy factor that weighs each neighbor by its spatial distance and its lack of membership in the cluster.

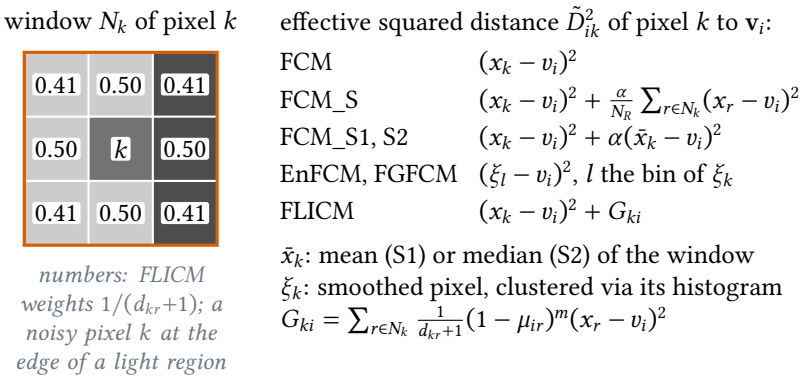


Figure 11.3. Spatial constraints for image segmentation. Top left: the 3×3 neighborhood of a pixel with the spatial weights of FLICM. Top right: the effective squared distance that every method substitutes into the FCM membership formula. Bottom: EnFCM and FGFCM smooth the image once and cluster the gray-level histogram with weighted FCM.

11.2 Theory

We use the notation of section 2.2.9: rows $\mathbf{x}_k$ of $\mathbf{X} \in \mathbb{R}^{N\times n}$, prototypes $\mathbf{v}_i$, and memberships μ_{ik} in $\mathbf{U} \in \mathbb{R}^{c\times N}$ (stored transposed, $N \times c$, in the code), with i (or j) indexing clusters and k samples. All algorithms of the chapter use the membership formula of fuzzy c-means [3],

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{c} \left(D^2_{ik}/D^2_{jk}\right)^{1/(m-1)}}, \tag{11.1}$$

each with its own squared distance D^2_{ik}. We treat incomplete data first (section 11.2.1), then weighted fuzzy c-means and the three schemes for very large data (sections 11.2.2 to 11.2.5), and finally the six image segmentation methods (section 11.2.6).

11.2.1 Fuzzy c-means with missing values

Let $I_{kj} = 1$ when x_{kj} is observed and 0 when it is missing, $I_k = \sum_j I_{kj}$ the number of observed features of sample k, X_P the set of observed and X_M the set of missing values. Every sample must have at least one observed feature, and every feature must be observed in at least one sample.

Whole-data strategy (WDS). Fuzzy c-means runs on the complete samples only; the incomplete ones are classified afterwards with partial distances. With n features and values missing independently with probability p, a sample is complete with probability $(1-p)^n$; with four features and 30 % of the values missing, three quarters of the samples are lost.

Partial-distance strategy (PDS). The squared distance is computed over the observed features and rescaled to the full dimension,

$$D_{ik}^2 = \frac{n}{I_k} \sum_{j=1}^{n} I_{kj} \, (x_{kj} - v_{ij})^2, \tag{11.2}$$

which is the ordinary squared Euclidean distance for a complete sample and an unbiased estimate of it for an incomplete one when the coordinates contribute equally on average [152]. The memberships follow from (11.1), and every prototype coordinate is the membership-weighted mean of the samples that observe it,

$$v_{ij} = \frac{\sum_{k=1}^{N} \mu_{ik}^m I_{kj} \, x_{kj}}{\sum_{k=1}^{N} \mu_{ik}^m I_{kj}}. \tag{11.3}$$

Optimal-completion strategy (OCS). The c-means functional on the completed data is minimized over the partition, the prototypes, *and* the missing values,

$$\min_{\mathbf{U},\mathbf{V},X_M} J = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^m \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2 . \tag{11.4}$$

For fixed $\mathbf{U}$ and $\mathbf{V}$ the functional is a sum of independent quadratics in the missing values, and setting the derivative to zero gives

$$x_{kj} = \frac{\sum_{i=1}^{c} \mu_{ik}^m v_{ij}}{\sum_{i=1}^{c} \mu_{ik}^m}, \qquad x_{kj} \in X_M, \tag{11.5}$$

the membership-weighted mean of the prototype coordinates (fig. 11.1). For fixed X_M the conditions are those of fuzzy c-means on the completed data. Alternating the three blocks decreases J monotonically, and the clustering yields an imputation of the data as a by-product. The price is a functional with more local optima. The imputed values take part in the distances, so an imputed value that leans toward one prototype makes the sample look closer to it, which raises its membership and pulls the value further toward that prototype in the next step. From a poor start, samples can be captured by the wrong cluster in this way, which is why OCS is best started from the prototypes of a PDS fit (`init=pds.cluster_centers_`).

Algorithm 11.1 Fuzzy c-means for incomplete data (WDS, PDS, OCS, NPS)

Input: data $\mathbf{X}$ with missing values, number of clusters c, exponent m, strategy, tolerance ϵ

if *WDS* **then** run fuzzy c-means on the complete samples, classify the others by (11.2), stop;
initial prototypes $\mathbf{V}$; initial memberships from the partial distances (11.2);
if *OCS or NPS* **then** fill X_M by (11.5) or by the nearest prototype;
repeat
  if *PDS* **then**
    prototypes by (11.3); distances by (11.2)
  else
    prototypes by the FCM update on the completed data; missing values by (11.5) (OCS) or by the nearest prototype (NPS); Euclidean distances on the completed data
  memberships by (11.1);
until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: partition $\mathbf{U}$ of all samples, prototypes $\mathbf{V}$, completed data

Nearest-prototype strategy (NPS). NPS is the crisp version of (11.5): the missing value is the coordinate of the prototype nearest to the sample, $x_{kj} = v_{i^* j}$ with $i^* = \arg\min_i D_{ik}^2$, the nearest prototype being judged by the partial distance (11.2). It is cheaper than OCS and a little coarser.

For every strategy `fuzzyclust` returns a completed data matrix: the optimized values for OCS and NPS, and a post hoc estimate by (11.5) from the final partition for PDS and WDS. New samples, complete or not, are classified by their partial distances. The initial prototypes are seeded on the data with the holes provisionally filled by column means, but the initial partition is computed from partial distances, and OCS and NPS start from the imputation of that partition rather than from the column means.

11.2.2 Weighted fuzzy c-means and the extension step

Let every sample carry a non-negative weight w_k, the number of observations it stands for. Weighted fuzzy c-means (wFCM) minimizes

$$J_w = \sum_{i=1}^{c} \sum_{k=1}^{N} w_k \, \mu_{ik}^m \, \|\mathbf{x}_k - \mathbf{v}_i\|^2 . \tag{11.6}$$

The weight multiplies all terms of sample k and cancels in its membership condition, so the memberships follow (11.1) with the Euclidean distance unchanged; the

In a nutshell **Fuzzy c-means for incomplete data**

- Four strategies for data with values missing completely at random: drop incomplete samples (WDS), partial distances (PDS), missing values as optimization variables (OCS), and nearest-prototype completion (NPS).
- PDS keeps every observed value at no extra cost and is the most reliable; it is the default. OCS and NPS impute while they cluster; OCS has more local optima and should be started from the PDS prototypes.
- Every fit comes with a completed data matrix, which is a good imputation when the data have cluster structure.
- Parameters: `strategy`; c, m, `init`, and `n_init` as for FCM.

Algorithm 11.2 **Weighted fuzzy c-means (wFCM)**

1. Initialize $\mathbf{V}$ (random or weighted k-means++ seeding); memberships by (11.1).
2. Repeat: prototypes by (11.7); Euclidean distances; memberships by (11.1); until $\max_{ik} |\Delta\mu_{ik}| < \epsilon$.
3. Output $\mathbf{U}$, $\mathbf{V}$, J_w, and the prototype weights $w'_i = \sum_k w_k \mu_{ik}$.

prototypes become weighted means,

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} w_k \, \mu_{ik}^m \, \mathbf{x}_k}{\sum_{k=1}^{N} w_k \, \mu_{ik}^m}. \tag{11.7}$$

With integer weights, wFCM is fuzzy c-means on the data set in which sample k is repeated w_k times; with all weights equal to one, it is fuzzy c-means. The weight of a prototype after a weighted clustering is the membership mass it collected, $w'_i = \sum_k w_k \mu_{ik}$. These weights add up to the total weight of the data, so a set of weighted prototypes is a lossy but mass-preserving summary. The second building block is the *extension*: for fixed prototypes, one evaluation of (11.1) gives the memberships of any number of samples, chunk by chunk with $O(\text{chunk} \cdot c)$ memory. The histogram-based image methods of section 11.2.6 use wFCM with the gray-level counts as weights.

11.2.3 Random sample and extend (rseFCM)

The simplest scheme draws $n_s \ll N$ samples at random, runs fuzzy c-means on them, and extends the result to all samples. Its cost is one FCM run on n_s samples plus one pass over the data, $O(n_s \, c \, l + N \, c)$ for l iterations, and Havens et al. [156], who measure every scheme against literal fuzzy c-means (LFCM), recommend it for very large image data in more than one dimension. The scheme has no approximation guarantee. If FCM on the sample finds a different local

In a nutshell Weighted fuzzy c-means

- Fuzzy c-means in which sample k counts w_k times; the membership formula is unchanged and the prototypes are weighted means.
- Lets weighted summaries (prototypes, histogram bins) be clustered together with raw data; the building block of spFCM, oFCM, EnFCM, and FGFCM.
- Parameters: those of FCM; the weights are passed to `fit` as `sample_weight`.

Algorithm 11.3 Random sample and extend FCM (rseFCM)

1. Draw n_s samples without replacement.
2. Run fuzzy c-means on the sample (with restarts if requested): prototypes $\mathbf{V}$.
3. Extend: memberships of all N samples by (11.1) with $\mathbf{V}$ fixed, chunk by chunk.

optimum than FCM on the whole data, for instance because a small cluster is underrepresented in the sample, the extension carries the error to all samples.

11.2.4 Single-pass FCM (spFCM)

The data are split into chunks $\mathbf{X}_1, \dots, \mathbf{X}_s$ of n_l samples each. The first chunk is clustered with fuzzy c-means; its prototypes $\mathbf{V}_1$ with the weights $w'_{1i} = \sum_k \mu_{ik}$ are appended to the second chunk as c weighted samples, and the augmented chunk is clustered with wFCM, initialized at $\mathbf{V}_1$:

$$(\mathbf{U}_l, \mathbf{V}_l) = \mathrm{wFCM}\big([\mathbf{X}_l; \mathbf{V}_{l-1}],\ [\mathbf{1}; \lambda\, \mathbf{w}'_{l-1}]\big), \qquad w'_{li} = \sum_k w_k\, \mu_{ik}, \tag{11.8}$$

for $l = 2, \dots, s$, where the sum runs over the augmented chunk. The prototypes of the last chunk summarize the data set, and the memberships of all samples follow by extension. Because a carried prototype weighs as much as the observations it

In a nutshell Random sample and extend FCM

- FCM on a random sample, one membership step for the rest; the fastest scheme.
- Adequate when the sample is large enough for FCM to find the right solution on it; small clusters are at risk.
- Parameters: `sample_size` (count or fraction), `chunk_size` of the extension, and those of FCM.

Algorithm 11.4 Single-pass FCM (spFCM)

Input: data $\mathbf{X}$ (or a stream of chunks), c, m, chunk size n_l, forgetting factor λ

split $\mathbf{X}$ into chunks $\mathbf{X}_1, \dots, \mathbf{X}_s$ (random order unless streaming);
$(\mathbf{U}_1, \mathbf{V}_1) \leftarrow$ fuzzy c-means on $\mathbf{X}_1$; $\mathbf{w}'_1 \leftarrow$ row sums of $\mathbf{U}_1$;
for $l = 2, \dots, s$ **do**
    $(\mathbf{U}_l, \mathbf{V}_l) \leftarrow$ wFCM on $[\mathbf{X}_l; \mathbf{V}_{l-1}]$ with weights $[\mathbf{1}; \lambda \mathbf{w}'_{l-1}]$, initialized at $\mathbf{V}_{l-1}$;
    $w'_{li} \leftarrow \sum_k w_k \mu_{ik}$ over the augmented chunk;
extend: memberships of all samples from $\mathbf{V}_s$ by (11.1);

Output: prototypes $\mathbf{V}_s$ and weights, chunk history, partition $\mathbf{U}$

In a nutshell Single-pass FCM

- Each chunk is clustered together with the weighted prototypes of the previous one; one pass over the data, memory of one chunk, `partial_fit` for streams.
- Close to FCM when every chunk contains all clusters; inherits the solution of the first chunk, so give that chunk restarts.
- The forgetting factor $\lambda < 1$ (an extension of the published algorithm) lets the prototypes follow drifting clusters.
- Parameters: `chunk_size`, `forgetting_factor`, `shuffle`, and those of FCM.

represents, the final prototypes approximate those of fuzzy c-means on the whole data; the approximation is good when every chunk contains all clusters. The weights also explain the weakness of the scheme: after a few chunks the carried prototypes outweigh any new chunk, so the result is anchored at the solution of the first chunk, and a poor local optimum there, such as two merged clusters, is never left. Restarts on the first chunk (`n_init`) are a cheap safeguard.

The published algorithm uses $\lambda = 1$; `fuzzyclust` adds a *forgetting factor* $\lambda \in [0, 1]$ that multiplies the carried weights before every chunk. The weight of chunk l in the summary after chunk L is then λ^{L-l}, an effective memory of about $1/(1-\lambda)$ chunks, which lets the prototypes follow clusters that drift in a stream; $\lambda = 0$ clusters every chunk on its own, started from the previous prototypes.

11.2.5 Online FCM (oFCM)

Online FCM clusters the chunks independently and merges the results:

$$\begin{aligned} (\mathbf{U}_l, \mathbf{V}_l) &= \mathrm{FCM}(\mathbf{X}_l), \qquad w_{li} = \sum_k \mu_{ik}, \qquad l = 1, \dots, s, \\ \mathbf{V} &= \mathrm{wFCM}\big([\mathbf{V}_1; \dots; \mathbf{V}_s],\ [\mathbf{w}_1; \dots; \mathbf{w}_s]\big), \end{aligned} \tag{11.9}$$

Algorithm 11.5 Online FCM (oFCM)

1. For every chunk $\mathbf{X}_l$: fuzzy c-means (started from the previous chunk's prototypes or afresh), prototypes $\mathbf{V}_l$ and weights $w_{li} = \sum_k \mu_{ik}$.
2. Cluster the $s \cdot c$ weighted prototypes into c clusters with wFCM, initialized at the prototypes of the last chunk: $\mathbf{V}$.
3. Extend: memberships of all samples from $\mathbf{V}$ by (11.1).

In a nutshell Online FCM

- Independent FCM runs on the chunks, a weighted FCM of their prototypes at the end; chunks can be processed in parallel or in any order.
- The least sensitive of the three schemes to poor chunk solutions, and the slowest.
- Parameters: `chunk_size`, `chunk_init` (`"previous"` or `"independent"`), `shuffle`, and those of FCM.

followed by the extension $\mathbf{U} = \text{extend}(\mathbf{X}, \mathbf{V})$. In the original formulation the chunk clusterings do not influence each other, so they can run in parallel, on different machines, or in any order as the chunks arrive; the final weighted clustering of $s \cdot c$ prototypes is cheap and can be repeated after every chunk, which is what `partial_fit` does. A poor local optimum on one chunk contributes only c of the $s \cdot c$ merged points and is outvoted. The price is the storage of all chunk prototypes and a full FCM run to convergence on every chunk, which makes oFCM the slowest of the three schemes. By default, `fuzzyclust` starts each chunk run from the prototypes of the previous chunk, which saves iterations and keeps the cluster order aligned; `chunk_init="independent"` restores fully independent starts. Like spFCM with $\lambda = 1$, oFCM summarizes the whole past and lags behind drifting clusters; its chunk prototypes $\mathbf{V}_l$, on the other hand, are memoryless estimates of the current clusters.

11.2.6 Spatially constrained fuzzy c-means for images

Let x_k be the value of pixel k (a gray level, or a vector of channels for the pixel-based methods), $\mathbf{p}_k$ its position (row and column index), N_k the set of its neighbors in a square window of side a with the center excluded, $N_R = a^2 - 1$ their number, $d_{kr} = \|\mathbf{p}_k - \mathbf{p}_r\|$ the Euclidean distance between the positions of k and r, and $\mathbf{v}_i$ the prototype of region i. All methods iterate the membership formula (11.1) with an *effective* squared distance $\tilde{D}^2_{ik}$ (fig. 11.3) and an adapted prototype update. Windows that extend beyond the image border are filled by reflection.

Algorithm 11.6 FCM_S

1. Precompute the neighborhood sums $\sum_{r\in N_k} \mathbf{x}_r$; initialize the partition.
2. Repeat: prototypes by (11.11); squared distances of all pixels; their neighborhood sums; effective distances by (11.11); memberships by (11.1); until $\max_{ik} |\Delta\mu_{ik}| < \epsilon$.

In a nutshell FCM_S

- Adds the distances of the neighbors to the prototype; suppresses Gaussian noise but smooths with fixed weights.
- Cost $O(N\, c\, N_R)$ per iteration; mean-type smoothing spreads impulse noise instead of removing it.
- Parameters: `alpha` (α), `window_size`.

FCM_S. Ahmed et al. [159] add a term to the functional that penalizes neighbors of a pixel that lie far from the prototype of the pixel's cluster,

$$J_S = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^m \|\mathbf{x}_k - \mathbf{v}_i\|^2 + \frac{\alpha}{N_R} \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^m \sum_{r\in N_k} \|\mathbf{x}_r - \mathbf{v}_i\|^2 , \tag{11.10}$$

which gives the effective distance and the prototype update

$$\begin{aligned} \tilde{D}_{ik}^2 &= \|\mathbf{x}_k - \mathbf{v}_i\|^2 + \frac{\alpha}{N_R} \sum_{r\in N_k} \|\mathbf{x}_r - \mathbf{v}_i\|^2 , \\ \mathbf{v}_i &= \frac{\sum_k \mu_{ik}^m \left(\mathbf{x}_k + \frac{\alpha}{N_R} \sum_{r\in N_k} \mathbf{x}_r\right)}{(1+\alpha)\sum_k \mu_{ik}^m} . \end{aligned} \tag{11.11}$$

A pixel whose value is corrupted but whose neighbors are clean is pulled into the neighbors' cluster; the weight α sets how strongly. The paper's second component, the estimation of the intensity bias field of MR images, is not implemented. The neighborhood sums of the distances to every prototype must be recomputed in every iteration.

FCM_S1 and FCM_S2. Chen and Zhang [160] replace the neighbors with a single filtered value $\bar{\mathbf{x}}_k$, the mean (S1) or the median (S2) of the window, computed once in advance:

$$J_{S1,2} = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^m \left(\|\mathbf{x}_k - \mathbf{v}_i\|^2 + \alpha \|\bar{\mathbf{x}}_k - \mathbf{v}_i\|^2 \right), \qquad \mathbf{v}_i = \frac{\sum_k \mu_{ik}^m (\mathbf{x}_k + \alpha\bar{\mathbf{x}}_k)}{(1+\alpha)\sum_k \mu_{ik}^m} . \tag{11.12}$$

The cost per iteration is that of fuzzy c-means. The median makes FCM_S2 robust to impulse noise, but only when the median outweighs the pixel: for a white impulse in a gray region the pixel term favors the white cluster and the median term the gray one, and with $\alpha = 1$ the two balance exactly, so $\alpha > 1$ is needed. The paper's main contribution, a kernel-induced distance, is not implemented.

Algorithm 11.7 FCM_S1 and FCM_S2

1. Filter the image once: mean (S1) or median (S2) of every window, $\bar{\mathbf{x}}_k$.
2. Repeat: prototypes by (11.12); effective distances $\|\mathbf{x}_k - \mathbf{v}_i\|^2 + \alpha \|\bar{\mathbf{x}}_k - \mathbf{v}_i\|^2$; memberships by (11.1); until convergence.

In a nutshell FCM_S1 and FCM_S2

- The filtered pixel is a second observation weighted by α; the cost per iteration is that of FCM.
- S1 (mean) suits Gaussian noise; S2 (median) removes impulses and keeps edges, but needs $\alpha > 1$ against salt-and-pepper noise.
- Parameters: `alpha`, `window_size`.

EnFCM. Szilágyi et al. [161] form the linearly weighted sum image once,

$$\xi_k = \frac{1}{1+\alpha}\Big(x_k + \frac{\alpha}{N_R}\sum_{r\in N_k} x_r\Big), \tag{11.13}$$

quantize it into q gray levels ξ_l with pixel counts γ_l, and cluster the levels with the counts as weights,

$$J_{En} = \sum_{i=1}^{c}\sum_{l=1}^{q} \gamma_l\, \mu_{il}^m (\xi_l - v_i)^2, \qquad v_i = \frac{\sum_l \gamma_l \mu_{il}^m \xi_l}{\sum_l \gamma_l \mu_{il}^m}, \tag{11.14}$$

which is wFCM (11.6) on the q levels. A pixel inherits the memberships of its level. After one pass over the image, the iterations cost $O(q\,c)$ instead of $O(N\,c)$, about a thousand times less for a 512×512 image and $q = 256$. The method needs a single-channel image.

FGFCM. Cai, Chen, and Zhang [162] keep the histogram route but form the summed image with locally adaptive weights,

$$\begin{aligned} \xi_k &= \frac{\sum_{r\in N_k} S_{kr}\, x_r}{\sum_{r\in N_k} S_{kr}}, \\ S_{kr} &= \exp\Big(-\frac{\|\mathbf{p}_k - \mathbf{p}_r\|_\infty}{\lambda_s}\Big)\exp\Big(-\frac{(x_k - x_r)^2}{\lambda_g\,\sigma_k^2}\Big), \end{aligned} \tag{11.15}$$

Algorithm 11.8 EnFCM

1. Sum image ξ by (11.13); histogram: levels ξ_l and counts γ_l, $l = 1, \dots, q$.
2. wFCM on the levels with weights γ_l (11.14).
3. Every pixel takes the memberships of its level.

In a nutshell EnFCM

- Linear smoothing once, then fuzzy c-means on the gray-level histogram; the fastest method, with iterations independent of the image size.
- Same noise behavior as FCM_S1 (good for Gaussian, poor for impulse noise); gray-level images only.
- Parameters: `alpha`, `window_size`, `n_levels` (q).

Algorithm 11.9 FGFCM

1. Local similarities S_{kr} and adaptive image ξ by (11.15).
2. Histogram of ξ, wFCM on the levels (11.14), memberships of the pixels from their levels.

where $\|\cdot\|_\infty$ is the maximum norm, so that the spatial factor depends on the larger of the row and column offsets, and $\sigma_k^2 = \frac{1}{N_R}\sum_{r\in N_k}(x_r - x_k)^2$ is the local mean squared difference. The gray-level factor compares a neighbor's difference with the typical difference in the window: in a flat region all neighbors count, whereas an outlier among them or a neighbor on the far side of an edge hardly counts at all. Noise is smoothed without blurring edges, and an impulse does not leak into its neighbors. The clustering is then (11.14) on the histogram of ξ.

FLICM. Krinidis and Chatzis [163] add a *fuzzy factor* to the distance,

$$G_{ki} = \sum_{r\in N_k} \frac{1}{d_{kr}+1}\,(1-\mu_{ir})^m\,\|\mathbf{x}_r - \mathbf{v}_i\|^2\,, \qquad \tilde{D}_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2 + G_{ki}, \tag{11.16}$$

with the prototypes updated as in fuzzy c-means, $\mathbf{v}_i = \sum_k \mu_{ik}^m \mathbf{x}_k / \sum_k \mu_{ik}^m$. A neighbor that belongs firmly to cluster i ($\mu_{ir} \approx 1$) adds nothing to the distance of pixel k to $\mathbf{v}_i$; a neighbor that does not belong to it adds its own distance, damped by the spatial distance. A pixel surrounded by members of cluster i is thus drawn into it even when its own value is far from $\mathbf{v}_i$, and the factor needs no weight parameter. Because G_{ki} depends on the memberships, it is evaluated with the memberships of the previous iteration. The iteration is therefore a fixed-point scheme rather

In a nutshell FGFCM

- EnFCM with edge-preserving, outlier-rejecting weights; as fast as EnFCM.
- Handled both Gaussian and impulse noise well in our experiments; gray-level images only.
- Parameters: `lambda_s`, `lambda_g` (scales of the spatial and gray-level factors), `window_size`, `n_levels`.

Algorithm 11.10 FLICM

1. Initialize the partition.
2. Repeat: prototypes as in FCM; squared distances of all pixels; fuzzy factors G_{ki} by (11.16) with the current memberships; memberships by (11.1) with $\tilde{D}^2_{ik}$; until convergence.

In a nutshell FLICM

- A fuzzy factor built from the neighbors' distances and their lack of membership; no smoothing weight to tune.
- Among the best under Gaussian noise; moves strong impulses only halfway when the extreme gray levels are classes of their own.
- Cost $O(N\, c\, N_R)$ per iteration; sensitive to the initialization. Parameters: `window_size`, `m`.

than an exact minimization of the functional $\sum_{i,k}(\mu_{ik}^m \|\mathbf{x}_k - \mathbf{v}_i\|^2 + G_{ki})$, and in our experiments its result depended more on the initialization than that of the other methods; the examples use k-means++ seeding.

11.3 Using it in fuzzyclust

The incomplete-data estimator and the very-large-data schemes derive from `BaseFuzzyClusterer` and have the usual methods `fit`, `predict`, `predict_proba`, `transform`, and `score`; the chunked schemes add `partial_fit`. `SpatialFCM` takes an image of shape (H, W) or (H, W, C) instead of a data matrix and produces label and membership images.

`fuzzyclust.incomplete.IncompleteFCM`

`n_clusters=3`, `m=2.0`, `strategy="pds"|"wds"|"ocs"|"nps"`, `tol`, `max_iter`, `init` (string or prototypes, e.g. those of a PDS fit), `n_init`, `random_state`, `verbose`. Missing values are `NaN`.
Attributes `cluster_centers_`, `membership_`, `labels_` (all samples), `distances_`, `missing_mask_`, `n_missing_`, `X_completed_` (optimized for OCS/NPS, post hoc for PDS/WDS), `covariances_`, `objective_`, `n_iter_`. `predict_proba` and `transform` accept incomplete samples (partial distances).

fuzzyclust.large.WeightedFCM

Parameters of FCM; `fit(X, sample_weight=None)`. Attributes as for FCM, plus `sample_weight_`. The function `extend_partition(X, V, m=2.0, chunk_size=None)` returns memberships and squared distances for fixed prototypes.

fuzzyclust.large.RandomSampleFCM

`n_clusters=3`, `sample_size=0.1` (count or fraction), `m`, `tol`, `max_iter`, `init`, `n_init`, `chunk_size=None` (extension), `random_state`, `verbose`. Attributes `sample_indices_`, `fcm_` (the FCM fitted on the sample), `cluster_centers_`, `membership_`, `labels_`, `objective_` (on all data).

fuzzyclust.large.SinglePassFCM, fuzzyclust.large.OnlineFCM

`n_clusters=3`, `chunk_size=10000`, `m`, `tol`, `max_iter`, `init`, `n_init` (first chunk; every chunk of `OnlineFCM` with `chunk_init="independent"`), `shuffle=True`, `random_state`, `verbose`; `SinglePassFCM` also `forgetting_factor=1.0`, `OnlineFCM` also `chunk_init="previous"|"independent"`.
Methods `fit(X)` and `partial_fit(chunk)`. Attributes `cluster_centers_`, `center_weights_`, `chunk_centers_`, `chunk_weights_`, `chunk_objective_`, `chunk_n_iter_`, `n_chunks_`, `membership_`, `labels_` (of the data passed last); `OnlineFCM.centers_model_` is the final weighted clusterer.

fuzzyclust.image.SpatialFCM

`n_clusters=3`, `method="flicm"|"fcm"|"fcm_s"|"fcm_s1"|"fcm_s2"|"enfcm"|"fgfcm"`, `m=2.0`, `alpha=1.0`, `window_size=3`, `lambda_s=3.0`, `lambda_g=6.0`, `n_levels=256`, `tol`, `max_iter=100`, `init`, `n_init`, `random_state`, `verbose`.
Attributes `cluster_centers_`, `membership_` (H, W, c), `labels_` (H, W), `filtered_image_`, `histogram_`, `level_values_`, `level_membership_` (histogram methods), `objective_`, `n_iter_`. `predict(image)` segments a new image with the fitted prototypes.

helpers, data generators, and plots

`partial_distances(X, V)`; `make_mcar(X, missing_rate)`, `make_incomplete_blobs`; `plot_incomplete_clusters`, `plot_imputation`. `make_large_blobs`, `make_drifting_stream` (a generator of chunks); `plot_chunk_timeline`, `plot_center_trajectories`. `neighborhood_`

`sum`, `weighted_sum_image`, `local_similarity_image`; `make_synthetic_image`, `add_image_noise`; `segmentation_accuracy`, `match_labels`; `plot_image`, `plot_label_image`, `plot_membership_maps`, `plot_segmentation`.

All four strategies run on the same incomplete matrix, and OCS is best started from the PDS prototypes:

```
Xm = make_mcar(X, missing_rate=0.25, random_state=0)
fits = {s: IncompleteFCM(n_clusters=3, strategy=s, random_state=0).fit(Xm)
    for s in STRATEGIES}
for s, model in fits.items():
    print(f"{s.upper()}: {model.n_iter_:3d} iterations, ARI =
    {adjusted_rand_score(y, model.labels_):.3f}, "
          f"prototype error = {center_error(model.cluster_centers_,
    reference.cluster_centers_):.3f}")
```

```
Xm, mask = make_mcar(X, missing_rate=0.2, random_state=0, return_mask=True)
pds = IncompleteFCM(n_clusters=4, strategy="pds", random_state=0).fit(Xm)
models = {
    "PDS (post hoc)": pds,
    "NPS": IncompleteFCM(n_clusters=4, strategy="nps",
    random_state=0).fit(Xm),
    "OCS, random start": IncompleteFCM(n_clusters=4, strategy="ocs",
    random_state=0).fit(Xm),
    "OCS, PDS start": IncompleteFCM(n_clusters=4, strategy="ocs",
    init=pds.cluster_centers_).fit(Xm),
}
errors = {name: rmse(model.X_completed_, mask) for name, model in
    models.items()}
```

The very-large-data schemes are drop-in replacements for FCM, and the chunked ones consume a stream with `partial_fit`:

```
literal, t_literal = timed_fit(WeightedFCM(n_clusters=5, random_state=0),
    X)
schemes = {
    "rseFCM": RandomSampleFCM(n_clusters=5, sample_size=0.01,
    random_state=0),
    "spFCM": SinglePassFCM(n_clusters=5, chunk_size=2000, random_state=0),
    "oFCM": OnlineFCM(n_clusters=5, chunk_size=2000, random_state=0),
}
results = {}
for name, model in schemes.items():
    model, seconds = timed_fit(model, X)
    results[name] = (t_literal / seconds,
    adjusted_rand_score(literal.labels_, model.labels_))
    print(f"{name}: speedup {results[name][0]:6.1f}, ARI with literal FCM
    {results[name][1]:.4f}")
```

```
models = {
    "spFCM, $\\lambda=1$ (paper)": SinglePassFCM(n_clusters=3,
    chunk_size=500, random_state=0),
    "spFCM, $\\lambda=0.5$": SinglePassFCM(n_clusters=3, chunk_size=500,
    forgetting_factor=0.5, random_state=0),
    "oFCM (merged)": OnlineFCM(n_clusters=3, chunk_size=500,
    random_state=0),
}
errors = {name: [] for name in models}
errors["oFCM, last chunk only"] = []
for Xc, _, C in stream:
    for name, model in models.items():
        model.partial_fit(Xc)
        errors[name].append(tracking_error(model.cluster_centers_, C))
    errors["oFCM, last chunk only"].append(tracking_error(models["oFCM
    (merged)"].chunk_centers_[-1], C))
```

An image is segmented with a single call:

```
noisy = add_image_noise(base.clean, noise="gaussian", noise_level=0.2,
    random_state=0)
model = SpatialFCM(n_clusters=4, method="flicm", init="kmeans++",
    random_state=0).fit(noisy)
accuracy = segmentation_accuracy(base.truth, model.labels_)
```

```
model = SpatialFCM(n_clusters=4, method="enfcm", alpha=4.0, window_size=5,
    init="kmeans++", random_state=0)
labels = model.fit(gauss).labels_
interior_acc, detail_acc = scores(labels)
```

11.4 Examples

11.4.1 Four strategies for missing values

Three well-separated Gaussian clusters of 150 points each in the plane lose 25 % of their values (fig. 11.4). WDS, PDS, and NPS recover the groups (ARI 0.99), with prototype errors against complete-data FCM of 0.11, 0.05, and 0.04. The completed coordinates show the character of each strategy: NPS snaps the imputed coordinates onto the prototypes, whereas PDS and WDS impute only afterwards. OCS from a random start ends in a poor solution (ARI 0.82, prototype error 0.47): samples of the lower-left cluster that lost their first coordinate were imputed with the first coordinate of the right cluster early on and remain captured there, forming the vertical streak of hollow markers in the figure. The memberships of incomplete samples (fig. 11.5) are lower when the observed coordinate is ambiguous. The parameter study in four dimensions (fig. 11.6) follows the experiment of Hathaway and Bezdek: WDS loses samples quickly, since only $(1 - p)^4$ of them remain complete, and its prototypes wander off beyond 40 %; PDS degrades most gracefully

(prototype error 0.13 at 40 %), NPS is close to it (0.14), and the randomly started OCS falls behind (0.28).

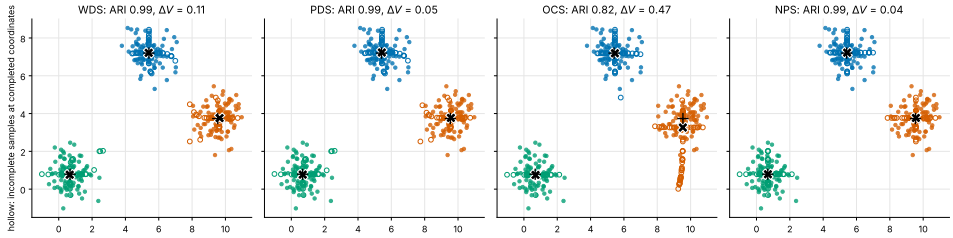


Figure 11.4. The four strategies on data with 25 % missing values; hollow markers are incomplete samples at their completed coordinates, crosses are the prototypes of each strategy, and plus signs those of FCM on the complete data. The randomly started OCS captures samples in the wrong cluster.

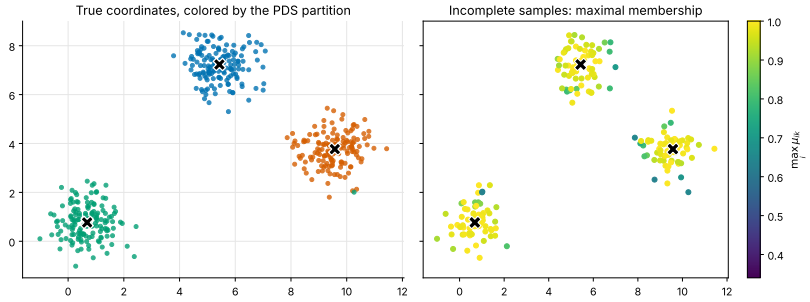


Figure 11.5. The PDS partition at the true coordinates (left) and the maximal membership of the incomplete samples (right): an ambiguous observed coordinate gives a low membership.

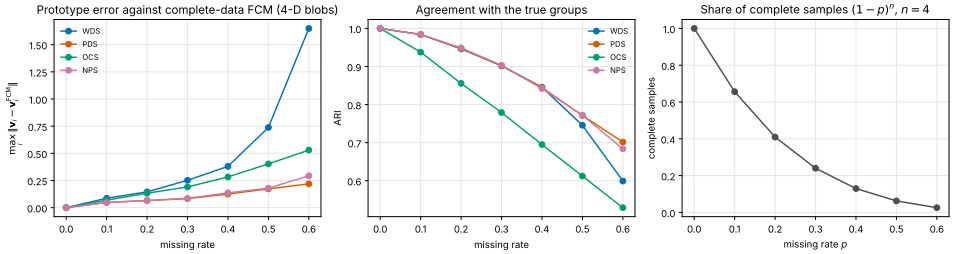


Figure 11.6. Prototype error against complete-data FCM and agreement with the true groups for missing rates up to 60 % (four-dimensional blobs), and the share of complete samples.

11.4.2 Imputation by clustering

The second data set has four clusters in four dimensions with 20 % of the values missing (fig. 11.7). Imputation by the column mean has an RMSE of 2.56; an

oracle that knows every sample's cluster mean reaches 0.64. The clustering-based imputations lie close to the oracle: PDS post hoc 0.81, NPS 0.83, and OCS started from the PDS prototypes 0.90, while OCS from a random start reaches only 1.29; the horizontal bands in its panel are samples imputed with the coordinates of a wrong prototype. Figure 11.8 shows why: the randomly started OCS is far more confident about the incomplete samples than PDS, which computes the memberships from the observed features only, and its large errors sit on samples that PDS considers ambiguous. The scans (fig. 11.9) show that the ranking holds at every missing rate (at 30 %: PDS 0.94, NPS and OCS from the PDS start 1.03, OCS from a random start 1.54) and that the exponent matters little for PDS, with a shallow optimum near $m = 2$; the randomly started OCS improves with m because softer memberships weaken the self-reinforcement of the imputed values.

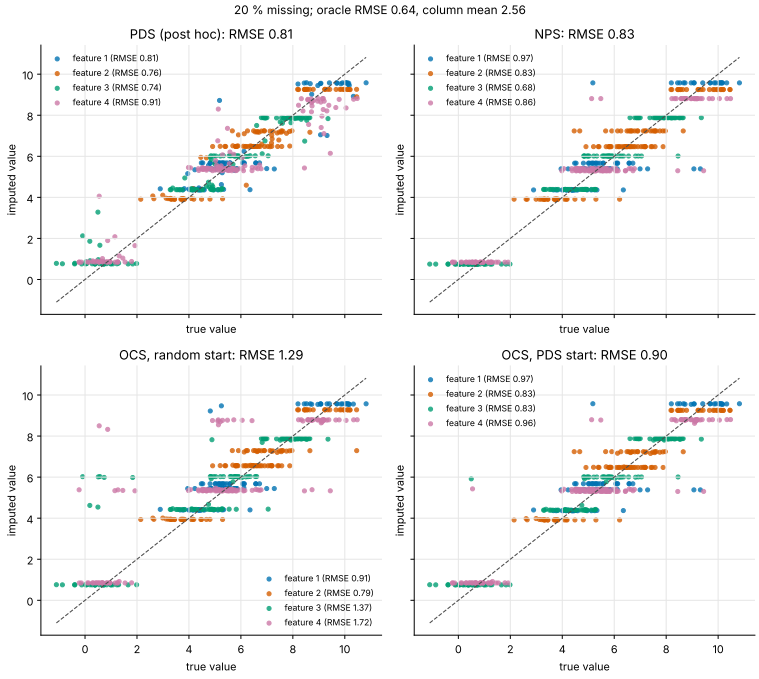


Figure 11.7. Imputed against true values with 20 % of the values missing, for PDS (post hoc), NPS, and OCS from a random and from the PDS start.

11.4.3 200 000 samples

The third data set has 200 000 points in five clusters of very different sizes, from 35 % down to 5 % of the data (fig. 11.10). Literal fuzzy c-means needs 32 iterations over all samples. With 1 % of the data at a time, rseFCM is more than ten times, spFCM about six times, and oFCM about twice as fast as literal FCM, and all three agree with the literal partition with an ARI of 0.998 or better; the measured speedups vary from run to run with the machine and its load. The chunk timeline

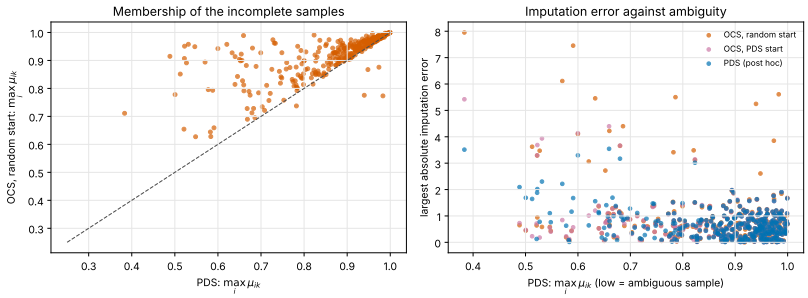


Figure 11.8. Memberships of the incomplete samples under OCS and PDS (left), and the largest imputation error of a sample against its PDS membership (right).

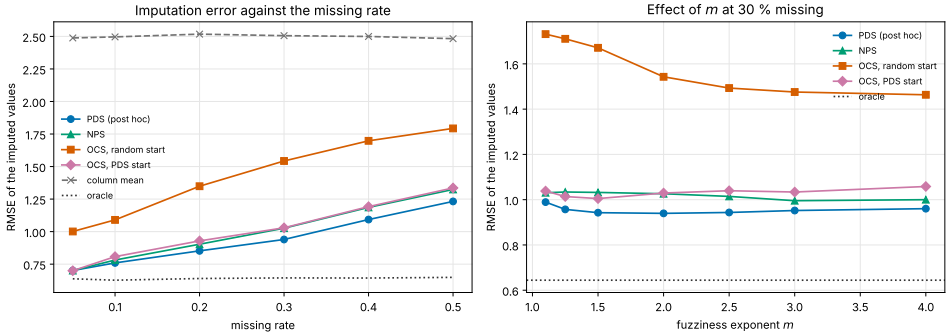


Figure 11.9. Imputation error against the missing rate (left) and against the exponent m at 30 % missing (right), with the column mean and the oracle as references.

of spFCM (fig. 11.11, left) is flat after the first chunks: the carried prototypes soon outweigh any new chunk. The chunk prototypes of oFCM scatter around the merged ones by the sampling error of a chunk, most for the small cluster, of which a chunk holds only about a hundred members. The chunk-size study (fig. 11.12) exposes the weakness of the first two schemes: with 600 samples or fewer, FCM on the sample (or on the first chunk) merges two clusters and splits the largest (ARI 0.87 with the literal partition), and spFCM carries this solution through all later chunks. oFCM outvotes the poor chunk solutions and stays above 0.99 at every size, and five restarts on the first chunk repair spFCM at a small cost.

11.4.4 A drifting stream

Three clusters move along straight lines while 40 chunks of 500 samples are fed to `partial_fit` (fig. 11.13). The published spFCM and oFCM summarize the whole past, so their prototypes sit near the time average of the centers, halfway along the path; with $\lambda = 0.5$ spFCM follows the clusters. The tracking error (fig. 11.14) grows linearly for the two published schemes and settles at a small constant for $\lambda = 0.5$, close to that of a memoryless FCM on the last chunk. The parameter study with chunks of 200 samples (fig. 11.15) shows the trade-off between lag and noise:

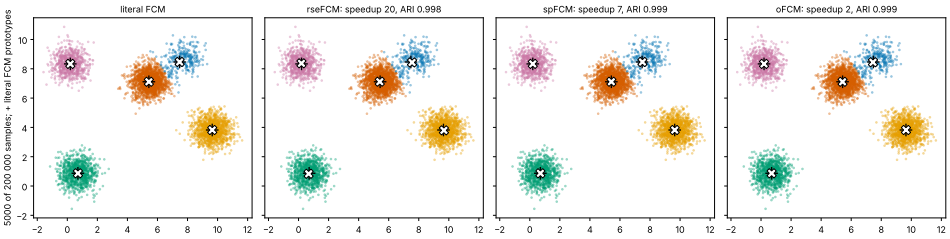


Figure 11.10. Literal FCM and the three schemes with 1 % chunks on 200 000 samples (5000 of them plotted); the white crosses are the prototypes of each scheme and cover the plus signs of the literal FCM prototypes, and the panel titles give speedup and agreement.

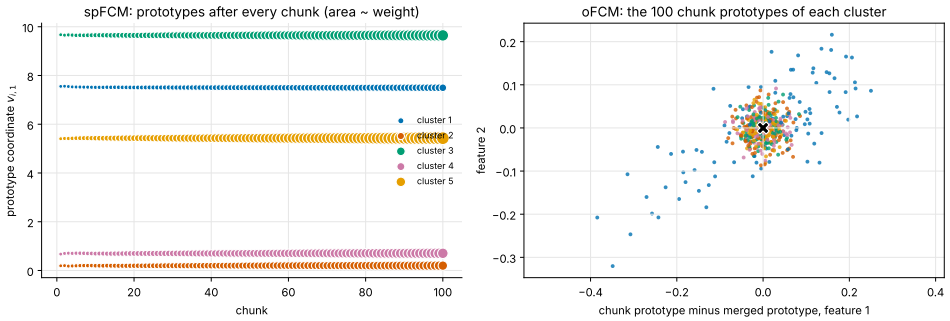


Figure 11.11. The spFCM prototypes after every chunk, marker area proportional to the carried weight (left), and the oFCM chunk prototypes relative to the merged ones (right).

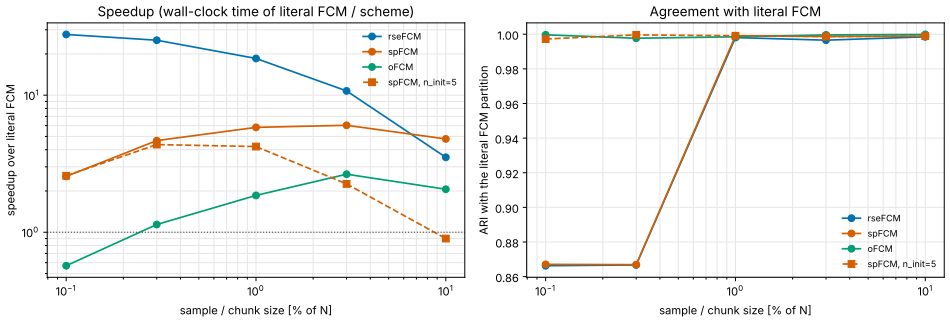


Figure 11.12. Speedup over literal FCM (left) and agreement with its partition (right) against the sample or chunk size.

without drift, a long memory is best ($\lambda = 0.95$, error 0.014 against 0.097 for $\lambda = 0$); with a drift of 0.02 per chunk, the best factor is 0.5; and with 0.05 or more, it is 0.25.

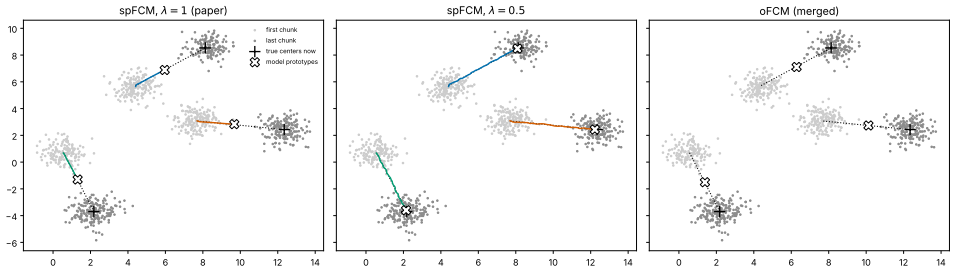


Figure 11.13. The first and the last chunk of a drifting stream with the true center paths (dotted), the prototype paths of spFCM with $\lambda = 1$ and $\lambda = 0.5$, and the final prototypes of the three models.

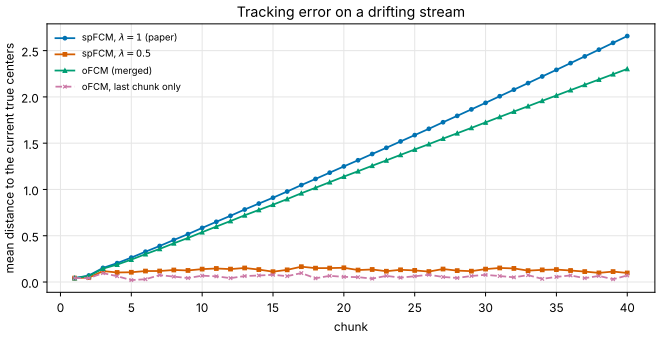


Figure 11.14. Distance of the prototypes from the current true centers along the stream.

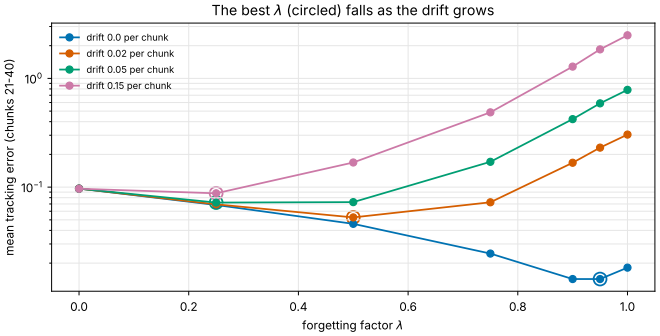


Figure 11.15. Mean tracking error over the second half of the stream against the forgetting factor for four drift speeds; circles mark the best factor.

11.4.5 Segmentation under noise

A 128 × 128 image with four regions at gray levels 0, 1/3, 2/3, and 1, and one disc of every class inside another region, is segmented by all seven methods with $\alpha = 4$, a 3 × 3 window, and k-means++ seeding. Under Gaussian noise with $\sigma = 0.2$ (fig. 11.16), pixel-wise FCM labels 62 % of the pixels correctly; FCM_S leaves clumps (0.91), FCM_S1 and FCM_S2 leave isolated specks (0.95, 0.94), and EnFCM, FGFCM, and FLICM give clean regions (0.96–0.97). All smoothing methods share one artifact: where the black and the white regions meet, the smoothed values are intermediate and a one-pixel band is assigned to a gray class. Salt-and-pepper noise on 20 % of the pixels (fig. 11.17) reverses the picture. The impulses coincide with the darkest and the brightest class, so FCM still labels 85 % of the pixels correctly. The mean-based methods spread the impulses and *lose* accuracy (FCM_S 0.68, EnFCM 0.80), whereas the median of FCM_S2 removes them (0.93) and the adaptive weights of FGFCM ignore them (0.92). FLICM (0.72) moves many impulses only halfway: a white pixel in the black region ends up in the dark-gray class, because its distance to the black prototype plus the small fuzzy factor of the black class exceeds its smaller distance to the dark-gray prototype plus the moderate fuzzy factor of that class. Across the noise levels (fig. 11.18), the adaptive methods keep their accuracy longest under Gaussian noise (at $\sigma = 0.3$: FLICM 0.90, FGFCM 0.88, FCM 0.51), and only FCM_S2 and FGFCM beat FCM consistently under impulse noise.

11.4.6 How much neighborhood?

Two thin bars, one and two pixels wide, are added to the test image, and the accuracy is measured separately on the interior of the regions and on the 11 % of the pixels next to a region boundary, labeled “detail” in the figures (fig. 11.19). For EnFCM under Gaussian noise, increasing α from 0 to 4 raises the interior accuracy from 0.63 to 0.99 but lowers the boundary accuracy from 0.65 to 0.54, and the one-pixel bar breaks up; larger windows round the corners, shrink the discs, and dissolve both bars. The α scan (fig. 11.20) shows the same trade-off for all three methods under Gaussian noise: the boundary accuracy peaks at $\alpha \approx 0.25$–1 for FCM_S and EnFCM and at $\alpha \approx 2$ for the edge-preserving median of FCM_S2. Under salt-and-pepper noise, FCM_S and EnFCM gain at most a few points on the interior over pixel-wise FCM (0.89; at best 0.92 for FCM_S at $\alpha = 16$) and lose half of the boundary accuracy (from 0.90 to below 0.50), and FCM_S2 helps only for $\alpha > 1$ (0.92 at $\alpha = 2$, 0.95 at 4, and 0.999 at 8), as predicted by the balance of the pixel and median terms. Enlarging the window with $\alpha = 4$ (fig. 11.21) costs boundary accuracy for all methods and, for FCM_S, interior accuracy as well (0.97 at 5 × 5, 0.84 at 11 × 11), while the median keeps the interior of FCM_S2 at 0.995.

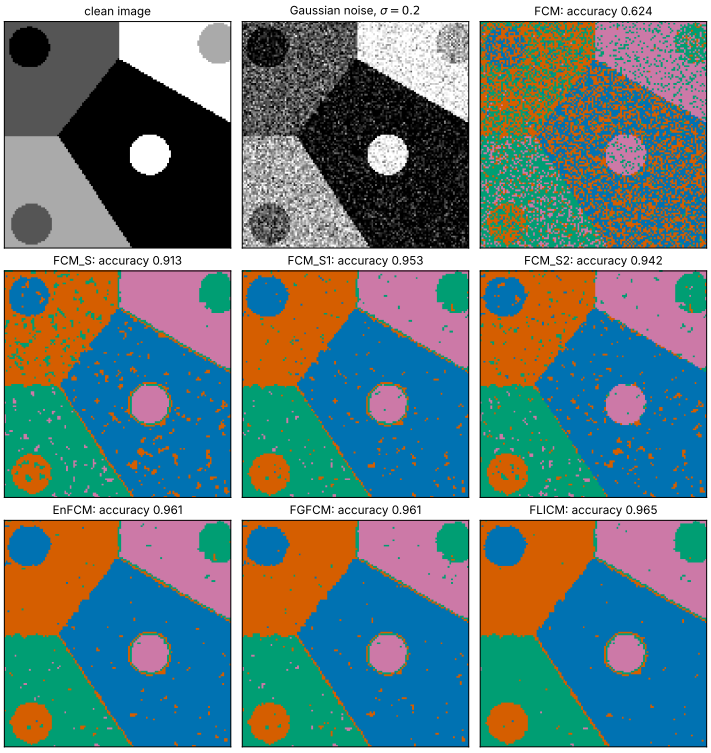


Figure 11.16. Seven segmentations of an image with Gaussian noise ($\sigma = 0.2$); the titles give the share of correctly labeled pixels.

11.5 Further reading

The four strategies for incomplete data are compared in [151]; the partial distance goes back to Dixon [152], and the statistical framework of missing data, including the distinction between values missing completely at random and other mechanisms, is given in [153]. Havens et al. [156] compare fuzzy c-means algorithms for very large data, covering rseFCM, spFCM, oFCM, and their kernel versions. Single-pass FCM is introduced in [157], and online FCM and its use for large magnetic resonance volumes in [158]; earlier work on making each FCM iteration cheaper includes [154, 155]. The spatially constrained segmentation methods are FCM_S [159], FCM_S1 and FCM_S2 with their kernel versions [160], EnFCM [161], FGFCM [162], and FLICM [163]. A related approach for MR images, which estimates the bias field together with the partition and adds a smoothness penalty on that field, is the adaptive fuzzy c-means of Pham and Prince [164]. All rest on the c-means functional of Bezdek [3].

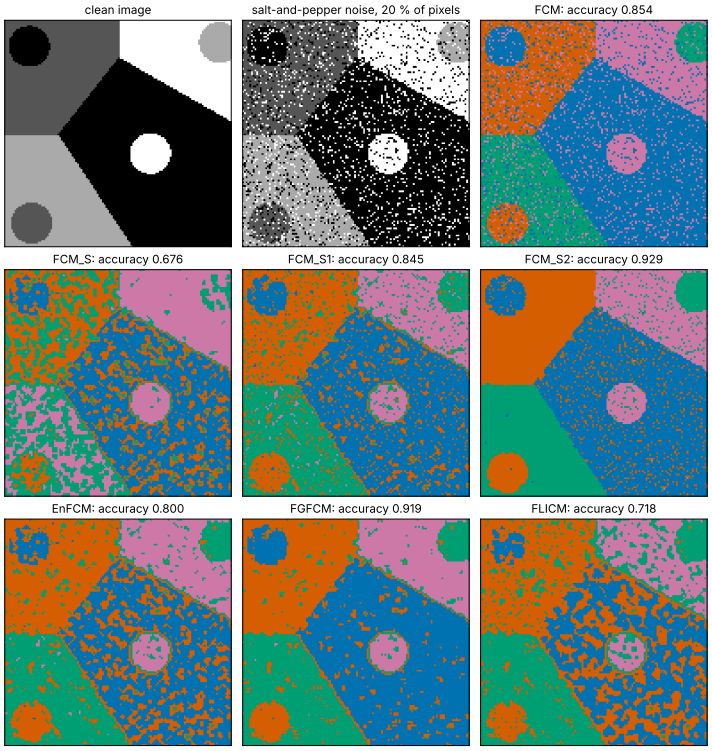


Figure 11.17. The same image with 20 % salt-and-pepper noise: averaging spreads the impulses, whereas the median and the adaptive weights remove them.

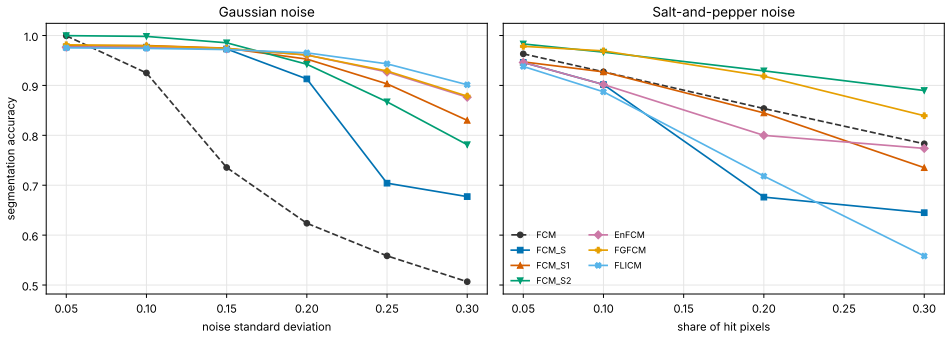


Figure 11.18. Segmentation accuracy against the level of Gaussian (left) and salt-and-pepper noise (right).

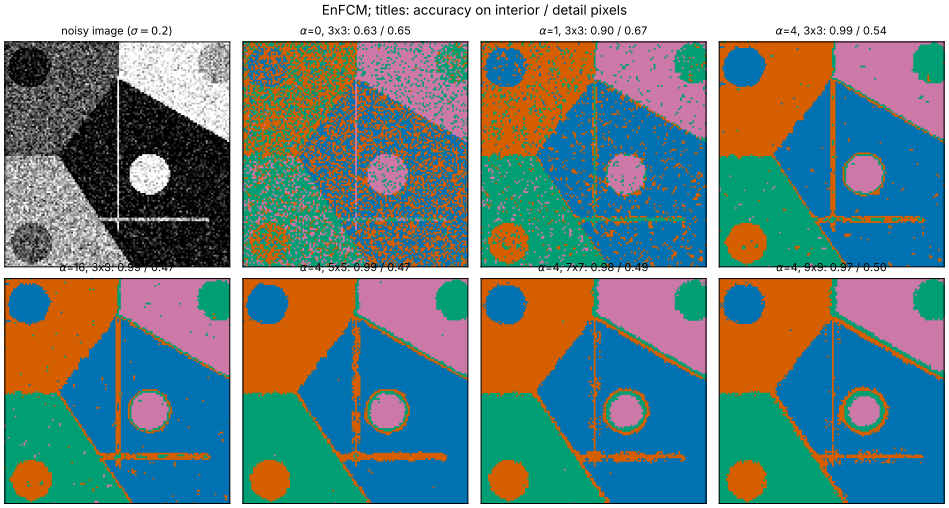


Figure 11.19. EnFCM with increasing weight α in a 3×3 window (top row and bottom left) and with increasing window size at $\alpha = 4$ (rest of the bottom row); the titles give the accuracy on the interior and on the boundary (detail) pixels.

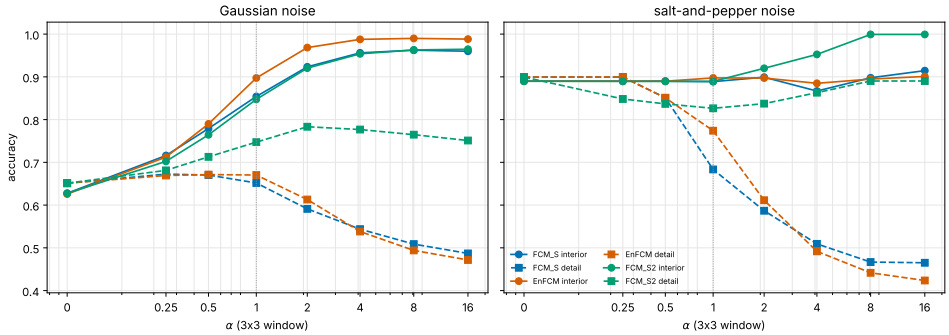


Figure 11.20. Interior (solid) and boundary (dashed) accuracy against α for FCM_S, EnFCM, and FCM_S2 under Gaussian (left) and salt-and-pepper noise (right).

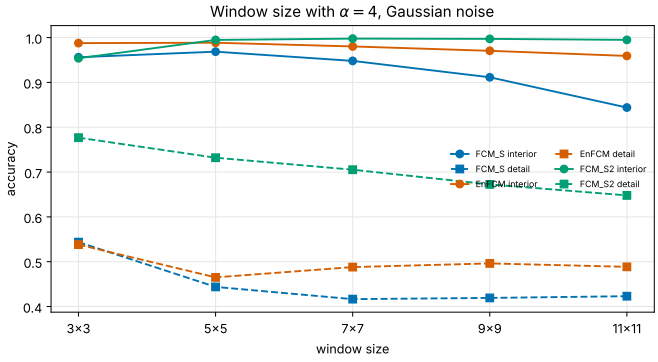


Figure 11.21. Interior and boundary accuracy against the window size with $\alpha = 4$ under Gaussian noise.

CHAPTER

12



Beyond a single unsupervised run: partial supervision, co-clustering, collaboration, and ensembles

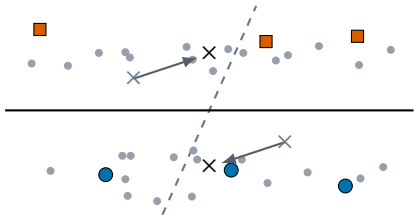
All algorithms so far take one data matrix, cluster its rows without outside knowledge, and return the partition of one run. This chapter relaxes each of these restrictions. Partial supervision lets a few class labels (Pedrycz and Waletzky) or a few must-link and cannot-link pairs (Grira, Crucianu, and Boujemaa) steer the partition toward the structure the user means; co-clustering (FCCM of Oh, Honda, and Ichihashi; Fuzzy CoDoK of Kummamuru, Dhawale, and Krishnapuram) clusters the rows *and* the columns of an object–item matrix at once; collaborative clustering (Pedrycz) reconciles the partitions of several data sites that describe the same objects without exchanging their data; and cluster ensembles (Strehl and Ghosh, Punera and Ghosh, and Fred and Jain) combine a library of runs into a consensus that is typically more accurate than its members and more reproducible than a single run, and they read the number of clusters off the agreement of the runs. Partial supervision and collaboration add a term to the c-means functional of chapter 3, co-clustering replaces it by an aggregation criterion, and ensembles combine its results; the methods live in `fuzzyclust.semisupervised`, `.coclustering`, `.collaborative`, and `.ensemble`.

12.1 Problem and idea

Fuzzy c-means finds the partition that the distance and the prototype shape make optimal, and that partition is not always the one the user wants. Two long parallel

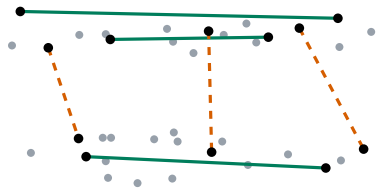
bars of points are a simple example (fig. 12.1): with two spherical prototypes the c-means functional prefers a split *across* the bars, because the variance along the bars dominates the gap between them, whereas the user knows that the bars are the classes. Pedrycz and Waletzky [165] observed that the user often knows the class of a few samples, and added to the functional a term that penalizes the difference between the memberships of these labeled samples and their labels. Since the labeled and the unlabeled samples share the prototypes, a handful of labels moves the prototypes and rotates the whole partition. Labels presuppose an agreed list of classes; a weaker and often cheaper kind of knowledge says only that two samples belong together (must-link) or not (cannot-link): two images show the same scene, two records describe the same customer. Grira, Crucianu, and Boujemaa [166] penalize the expected number of violated pairs and combine the penalty with the competitive agglomeration of Frigui and Krishnapuram [167], which lets an overestimated number of clusters shrink to the number the data and the constraints support.

(a) a few class labels



gray: FCM prototypes and boundary (dashed); black: with the labeled samples (squares, disks), whose term $\alpha \sum_{i,k} (\mu_{ik} - f_{ik} b_k)^2 D_{ik}^2$ *pulls the shared prototypes*

(b) must-link and cannot-link pairs



solid: pair $(k, l) \in \mathcal{M}$*, cost* $\sum_i \sum_{j \neq i} \mu_{ik} \mu_{jl}$*; dashed: pair* $(k, l) \in C$*, cost* $\sum_i \mu_{ik} \mu_{il}$*—the expected number of violated pairs is penalized*

Figure 12.1. Two kinds of partial supervision on two parallel bars that FCM splits across. (a) Labels: a few samples carry a class label, and the supervision term pulls their memberships, and with them the shared prototypes, toward the classes. (b) Pairs: must-link and cannot-link constraints say only whether two samples belong together; their expected violation is penalized.

A second limitation is the data matrix itself. For a document–term matrix, a user–product table, or any other non-negative *co-occurrence* matrix, the rows and the columns are both entities worth clustering, and a cluster is a *pair*: a group of documents and the vocabulary that makes them a group (fig. 12.2). Clustering the rows with a distance between sparse count vectors misses this duality. Oh, Honda, and Ichihashi [168] give each co-cluster two membership vectors, one over the objects and one over the items, and maximize the *aggregation* of the matrix entries that both memberships select; Kummamuru, Dhawale, and Krishnapuram [169] solve the same problem with a different regularizer that scales to large document collections.

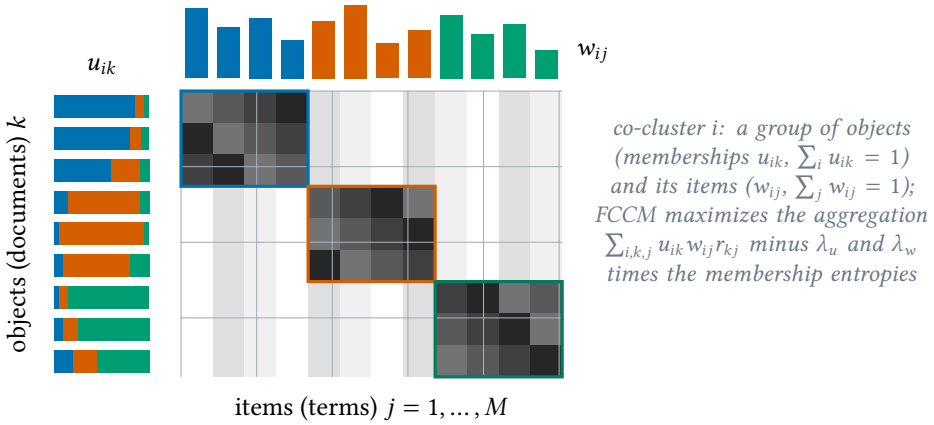


Figure 12.2. Co-clustering of an object–item matrix (rows and columns reordered). Every co-cluster has a membership vector over the objects (stacked bars, left) and one over the items (bars on top, colored by the co-cluster of largest membership); the dense blocks on the diagonal are the co-occurrences that both memberships select.

The third limitation is organizational. The same objects—patients, customers, production batches—are often described at several *data sites* by different features, and the raw data cannot be pooled for reasons of confidentiality, size, or ownership. Pedrycz's collaborative clustering [170] runs fuzzy c-means at every site and lets the sites exchange only their partition matrices (fig. 12.3): each site re-clusters its own data with an additional term that pulls its memberships toward those of the other sites, weighted by a collaboration matrix. The structures are reconciled without moving any data, and how far each site has to move tells how much it disagrees with the rest.

The last limitation is the single run. An alternating-optimization clusterer finds a local optimum that depends on its start, and the partition depends on the sample and on the features it saw; restarts (`n_init`) keep the run with the lowest objective but discard the information in the others. A *cluster ensemble* [171] keeps all of them (fig. 12.4). A library of partitions is generated by random starts, sub-samples, feature subsets, or different numbers of clusters. It is summarized by the *co-association matrix*, the average similarity of the membership vectors of two objects, which needs no matching of cluster labels between the members, and a consensus function partitions the objects from this matrix. Punera and Ghosh [172] showed how to use the soft partitions directly instead of hardening them first. Fred and Jain [173] used the co-association matrix to choose the number of clusters: when the members over-segment the data, the natural groups form the partition of its dendrogram that survives the widest range of thresholds.

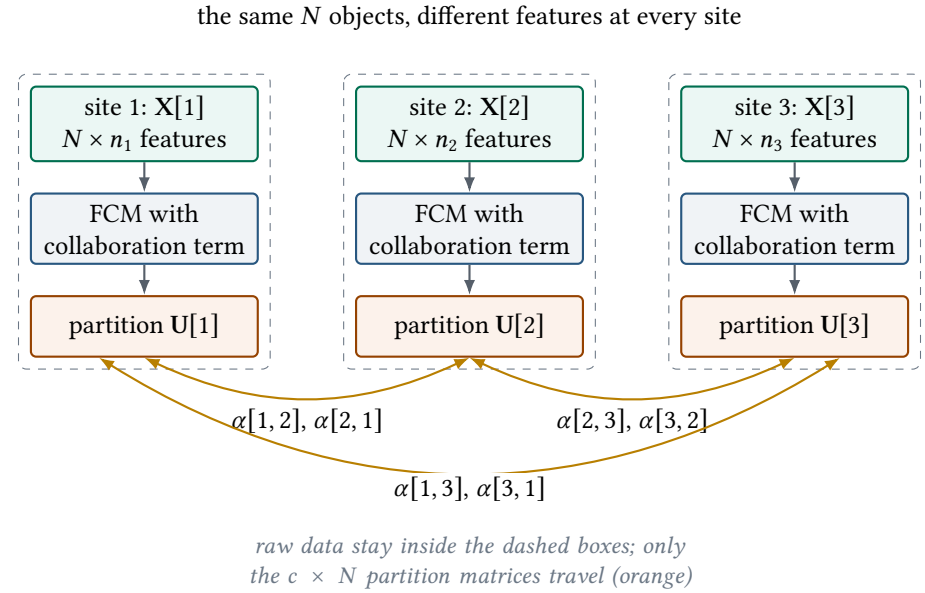


Figure 12.3. Collaborative clustering of three data sites. Every site clusters its own features of the same N objects; the sites exchange only their partition matrices, and the collaboration weight $\alpha[ii, jj]$ says how strongly site ii is pulled toward the partition of site jj.

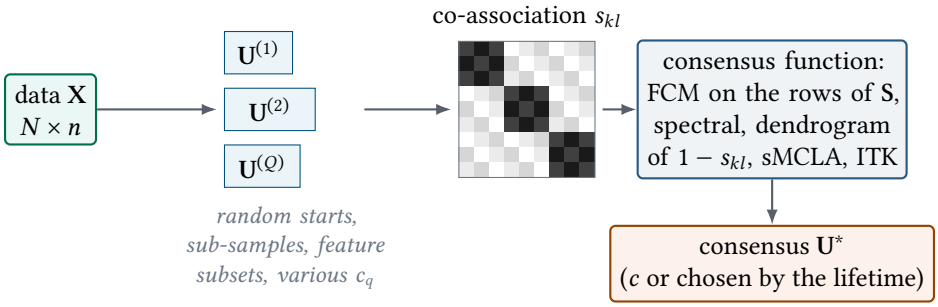


Figure 12.4. From a library of soft partitions to a consensus. The members may have different numbers of clusters; the co-association matrix compares only co-memberships, so their cluster labels need not be matched, and the consensus function partitions the objects from it.

12.2 Theory

The notation is that of the book: $\mathbf{X} \in \mathbb{R}^{N\times n}$ with rows $\mathbf{x}_k$, memberships μ_{ik} in $\mathbf{U} \in \mathbb{R}^{c\times N}$ (stored transposed in the code), prototypes $\mathbf{v}_i$, squared Euclidean distances $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$, index i or j for clusters and k or l for samples. The starting point is fuzzy c-means with $m = 2$: the functional $J = \sum_i \sum_k \mu_{ik}^2 D_{ik}^2$ (3.3), its membership formula $\mu_{ik} = 1/\sum_j (D_{ik}^2/D_{jk}^2)$ (3.6), and its prototype update (3.7). The supervised and the collaborative algorithms add a quadratic term to J and keep $m = 2$, the value for which their papers derive closed-form updates. We treat partial supervision by labels (section 12.2.1) and by pairs (section 12.2.2), co-clustering (section 12.2.3), collaboration (section 12.2.4), and ensembles (sections 12.2.5 and 12.2.6).

12.2.1 Fuzzy c-means with partial supervision

Let $b_k \in \{0, 1\}$ indicate whether sample k is labeled and let $f_{ik} \geq 0$, $\sum_i f_{ik} = 1$, be its label—the indicator of its class or, more generally, a fuzzy label vector; class j is tied to cluster j, and surplus clusters are unsupervised. Pedrycz and Waletzky [165] minimize

$$J = \sum_{i=1}^{c} \sum_{k=1}^{N} \mu_{ik}^2 D_{ik}^2 + \alpha \sum_{i=1}^{c} \sum_{k=1}^{N} \left(\mu_{ik} - f_{ik} b_k\right)^2 D_{ik}^2 \tag{12.1}$$

under $\sum_i \mu_{ik} = 1$. The second term measures, in the metric of the clustering, how far the memberships of the labeled samples are from their labels; for an unlabeled sample it reads $\alpha \mu_{ik}^2 D_{ik}^2$ and merely rescales the c-means term. Setting the derivative of the Lagrangian to zero gives

$$\mu_{ik} = \frac{1}{1+\alpha} \left[\frac{1 + \alpha\left(1 - b_k \sum_{j=1}^{c} f_{jk}\right)}{\sum_{j=1}^{c} D_{ik}^2/D_{jk}^2} + \alpha f_{ik} b_k \right], \tag{12.2}$$

which is the FCM membership for an unlabeled sample and, for a labeled one, the convex combination $\frac{1}{1+\alpha}\mu_{ik}^{\mathrm{FCM}} + \frac{\alpha}{1+\alpha} f_{ik}$: the label is followed to the degree $\alpha/(1+\alpha)$. The prototypes are weighted means with the weights of both terms,

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} w_{ik} \mathbf{x}_k}{\sum_{k=1}^{N} w_{ik}}, \qquad w_{ik} = \mu_{ik}^2 + \alpha\left(\mu_{ik} - f_{ik} b_k\right)^2. \tag{12.3}$$

The influence of the labels on the prototypes is not monotone in α. For a labeled sample whose membership has reached its label, the supervision weight $\alpha(\mu_{ik} - f_{ik})^2$ vanishes, while every unlabeled sample carries the weight $(1+\alpha)\mu_{ik}^2$; a very large α therefore clamps the labeled memberships but lets the unlabeled majority outweigh them in (12.3). The paper recommends α proportional to N/N_b, the ratio of all samples to the $N_b = \sum_k b_k$ labeled ones, so that the labeled minority is not ignored; the examples show that moderate values, $\alpha \approx 1$, move the partition most. Because class j is tied to cluster j, a random start that places cluster j next to the samples of

Algorithm 12.1 Fuzzy c-means with partial supervision

Input: data $\mathbf{X}$; labels f_{ik} and indicators b_k of the labeled samples; $c \geq$ number of classes; weight $\alpha \geq 0$ (or N/N_b); tolerance ϵ

initialize $\mathbf{v}_i$ at the means of the labeled samples of class i (random for unsupervised clusters);
repeat
  $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$;
  memberships by (12.2);
  prototypes by (12.3);
until $\max_{ik} \left|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}\right| < \epsilon$;

Output: $\mathbf{U}$, $\mathbf{V}$; predicted class of a sample = class tied to its cluster of largest membership

In a nutshell Fuzzy c-means with partial supervision

- Adds to the c-means functional the distance-weighted squared difference between the memberships of the labeled samples and their labels; the labels act on the whole partition through the shared prototypes.
- Labeled memberships are convex combinations of the FCM membership and the label, with weight $\alpha/(1+\alpha)$ on the label; unlabeled samples keep the FCM formula.
- Parameters: the weight α (moderate values move the partition most; N/N_b is the paper's rule), the number of clusters (at least the number of classes), and the start (at the labeled means).
- Use when a few samples of every class are known and the unsupervised optimum differs from the classes; $m = 2$.

another class makes the supervision fight the geometry; starting every supervised prototype at the mean of its labeled samples avoids this. New samples carry no label and receive FCM memberships to the supervised prototypes.

12.2.2 Pairwise constraints and competitive agglomeration

Let $\mathcal{M}$ be the set of must-link and $\mathcal{C}$ the set of cannot-link pairs (k, l). Under memberships μ_{ik}, the degree to which a must-link pair is *separated* is $\sum_i \sum_{j \neq i} \mu_{ik}\mu_{jl}$ and the degree to which a cannot-link pair is *together* is $\sum_i \mu_{ik}\mu_{il}$; summed over all pairs, these degrees give the expected number of violated constraints when every sample is assigned at random according to its memberships. Grira, Crucianu, and

Boujemaa [166] minimize, with the fuzzy cardinalities $N_i = \sum_k \mu_{ik}$,

$$J = \sum_{i=1}^{c}\sum_{k=1}^{N} \mu_{ik}^2 D_{ik}^2 + \alpha\left[\sum_{(k,l)\in\mathcal{M}}\sum_{i=1}^{c}\sum_{j\neq i}\mu_{ik}\mu_{jl} + \sum_{(k,l)\in C}\sum_{i=1}^{c}\mu_{ik}\mu_{il}\right] - \beta\sum_{i=1}^{c} N_i^2. \quad (12.4)$$

The last term is the competitive agglomeration bias of Frigui and Krishnapuram [167]: it rewards large clusters, so the clusters compete for samples and those that lose them can be discarded. Let $C_{ik} = \sum_{l:(k,l)\in\mathcal{M}}(1-\mu_{il}) + \sum_{l:(k,l)\in C}\mu_{il}$ be the cost of placing sample k in cluster i: the expected number of constraints it would violate there, given the current memberships of its partners. The Lagrange conditions, solved with the cross terms held at their previous values, give the FCM membership plus two corrections,

$$\mu_{ik} = \frac{1/D_{ik}^2}{\sum_j 1/D_{jk}^2} + \frac{\alpha}{2D_{ik}^2}\left(\bar{C}_k - C_{ik}\right) + \frac{\beta}{D_{ik}^2}\left(N_i - \bar{N}_k\right), \quad (12.5)$$

where $\bar{C}_k$ and $\bar{N}_k$ are the means of C_{jk} and N_j over the clusters, weighted by $1/D_{jk}^2$. A cluster in which k violates fewer constraints than on average gains membership, and with $\beta > 0$ a cluster larger than average gains membership. The corrections are weighted by $1/D_{ik}^2$ and cancel for a sample that is much closer to one prototype than to the others, so the constraints act most strongly on the ambiguous samples near the cluster boundaries. Negative values are clipped and the memberships renormalized, as in competitive agglomeration. The penalty α is in the units of the squared distances. Grira et al. renew it at every iteration so that the constraint term is commensurate with the data term: the mean membership-weighted squared distance, scaled by the ratio of samples to constraints,

$$\alpha = \frac{N}{|\mathcal{M}| + |C|}\,\frac{\sum_{i,k}\mu_{ik}^2 D_{ik}^2}{\sum_{i,k}\mu_{ik}^2}.$$

The automatic rule of `fuzzyclust` omits the normalization by $\sum_{i,k}\mu_{ik}^2$ and uses $\alpha = \sum_{i,k}\mu_{ik}^2 D_{ik}^2/(|\mathcal{M}| + |C|)$, the published value times $\sum_{i,k}\mu_{ik}^2/N \in [1/c, 1]$; the two agree for a crisp partition. The agglomeration weight follows the schedule of competitive agglomeration,

$$\beta(l) = \eta_0\, e^{-l/\tau}\,\frac{\sum_{i,k}\mu_{ik}^2 D_{ik}^2}{\sum_i N_i^2}, \quad (12.6)$$

strong at the start and decaying with the iteration l, and clusters whose cardinality N_i falls below a minimum are removed. The prototypes are the FCM means (3.7).

12.2.3 Fuzzy co-clustering of objects and items

Let $\mathbf{R} = [r_{kj}] \in \mathbb{R}_{\geq 0}^{N\times M}$ be an object–item matrix (counts, frequencies, TF-IDF weights). A co-cluster i has object memberships u_{ik} and item memberships w_{ij},

Algorithm 12.2 Pairwise-constrained competitive agglomeration (PCCA)

Input: data $\mathbf{X}$; constraint sets $\mathcal{M}$, $\mathcal{C}$; initial (maximal) c; penalty α (or the automatic rule); η_0, τ, minimal cardinality; tolerance ϵ

random initial partition $\mathbf{U}$;
repeat
  prototypes $\mathbf{v}_i$ by (3.7); $D_{ik}^2 = \|\mathbf{x}_k - \mathbf{v}_i\|^2$;
  costs C_{ik} from the constraints and the current $\mathbf{U}$; $N_i = \sum_k \mu_{ik}$; $\beta(l)$ by (12.6);
  memberships by (12.5); clip at zero and renormalize;
  if $\eta_0 > 0$ **then** discard clusters with N_i below the minimum, renormalize;
until $\max_{ik} |\Delta\mu_{ik}| < \epsilon$;

Output: $\mathbf{U}$, $\mathbf{V}$, final number of clusters, share of satisfied constraints

In a nutshell Pairwise-constrained competitive agglomeration

- Penalizes the expected number of violated must-link and cannot-link pairs; the membership update shifts membership toward the clusters in which a sample violates fewer constraints.
- Pairs need no list of classes and are cheap to collect; fewer pairs need a larger penalty, and a few pairs may be satisfied by bending the boundary locally.
- Optional competitive agglomeration: start with too many clusters, let them compete, and discard the small ones; the usable window of η_0 lies between "no competition" and "one cluster takes all".
- Parameters: α (or the automatic rule), η_0, τ, minimal cardinality; $m = 2$.

with the constraints $\sum_i u_{ik} = 1$ for every object (an object is shared among the co-clusters, as in FCM) and $\sum_j w_{ij} = 1$ for every co-cluster (each co-cluster distributes one unit of membership over the items). The quantity to be maximized is the *aggregation degree*

$$\sum_{i=1}^{c}\sum_{k=1}^{N}\sum_{j=1}^{M} u_{ik} w_{ij} r_{kj} = \sum_{i,k} u_{ik} a_{ik} = \sum_{i,j} w_{ij} b_{ij}, \qquad \begin{aligned} a_{ik} &= \textstyle\sum_j w_{ij} r_{kj}, \\ b_{ij} &= \textstyle\sum_k u_{ik} r_{kj}, \end{aligned} \tag{12.7}$$

large when the objects of a co-cluster cooccur with its items. The aggregation a_{ik} is a weighted *mean* of the row of object k over the items of co-cluster i, and b_{ij} is a weighted *sum* of the column of item j over the objects of the co-cluster. The aggregation degree alone is linear in each membership set and is maximized by crisp, degenerate memberships, with the whole item membership of a co-cluster on its single most frequent item. Oh, Honda, and Ichihashi [168] regularize both

Algorithm 12.3 Fuzzy clustering for categorical multivariate data (FCCM)

Input: non-negative matrix $\mathbf{R}$; c; $\lambda_u, \lambda_w > 0$; tolerance ϵ

random object memberships u_{ik} with $\sum_i u_{ik} = 1$;

repeat

  $b_{ij} = \sum_k u_{ik} r_{kj}$; item memberships w_{ij} by (12.9);

  $a_{ik} = \sum_j w_{ij} r_{kj}$; object memberships u_{ik} by (12.9);

until $\max_{ik} |\Delta u_{ik}| < \epsilon$;

Output: object and item memberships; hard co-clusters of objects and items by the largest membership

In a nutshell FCCM

- Clusters rows and columns of a non-negative co-occurrence matrix at once; a co-cluster is a group of objects with the items that characterize it.
- Maximizes the aggregation degree regularized by the entropies of both membership sets; the updates are soft-max assignments, and no distance between rows is needed.
- Parameters: λ_u (in the units of a row mean) and λ_w (in the units of a column sum over a co-cluster); outside a window of λ_w the solution degenerates.
- Use for document–term, user–item, or other count tables; restarts (`n_init`) keep the largest objective.

sets by their entropies, the device of Miyamoto and Mukaidono for fuzzy c-means [23], and maximize

$$J_{\mathrm{FCCM}} = \sum_{i,k,j} u_{ik} w_{ij} r_{kj} - \lambda_u \sum_{i,k} u_{ik} \log u_{ik} - \lambda_w \sum_{i,j} w_{ij} \log w_{ij}. \tag{12.8}$$

The necessary conditions are soft-max assignments,

$$u_{ik} = \frac{\exp(a_{ik}/\lambda_u)}{\sum_{l=1}^{c} \exp(a_{lk}/\lambda_u)}, \qquad w_{ij} = \frac{\exp(b_{ij}/\lambda_w)}{\sum_{l=1}^{M} \exp(b_{il}/\lambda_w)}, \tag{12.9}$$

alternated until the object memberships settle. The weights are measured in the units of the aggregations they temper: λ_u like a typical entry of $\mathbf{R}$, and λ_w like an entry summed over the objects of a co-cluster, so its scale grows with the size of the data. A small λ makes the corresponding memberships crisp; a large one drives them to the uniform values $1/c$ and $1/M$. New objects receive memberships by the first formula with the fitted item memberships.

Algorithm 12.4 Fuzzy co-clustering of documents and keywords (Fuzzy CoDoK)

1. Random object memberships u_{ik} with $\sum_i u_{ik} = 1$.
2. Item memberships by (12.11) from $b_{ij} = \sum_k u_{ik} r_{kj}$; clip at zero and renormalize every co-cluster to $\sum_j w_{ij} = 1$.
3. Object memberships by (12.11) from $a_{ik} = \sum_j w_{ij} r_{kj}$; clip at zero and renormalize every object to $\sum_i u_{ik} = 1$.
4. Repeat 2–3 until $\max_{ik} |\Delta u_{ik}| < \epsilon$.

In a nutshell Fuzzy CoDoK

- The FCCM aggregation with quadratic instead of entropy regularizers: linear updates, clipping instead of exponentials.
- Sparse item memberships (irrelevant items get zero), no overflow for large aggregations, and a wider usable range of the item weight T_w.
- Parameters: T_u and T_w in the units of FCCM's λ_u and λ_w.

Kummamuru, Dhawale, and Krishnapuram [169] replace the entropies by quadratic regularizers,

$$J_{\text{CoDoK}} = \sum_{i,k,j} u_{ik} w_{ij} r_{kj} - T_u \sum_{i,k} u_{ik}^2 - T_w \sum_{i,j} w_{ij}^2. \tag{12.10}$$

The necessary conditions are then *linear*: a membership is the uniform value plus a multiple of the deviation of its aggregation from the mean aggregation,

$$u_{ik} = \frac{1}{c} + \frac{1}{2T_u}\Big(a_{ik} - \frac{1}{c}\sum_{l=1}^{c} a_{lk}\Big), \qquad w_{ij} = \frac{1}{M} + \frac{1}{2T_w}\Big(b_{ij} - \frac{1}{M}\sum_{l=1}^{M} b_{il}\Big), \tag{12.11}$$

followed by clipping at zero and renormalization. No exponentials are evaluated, which removes the overflow of (12.9) for large aggregations, and the clipping sets the item memberships of the irrelevant terms to exactly zero, which gives a sparse and readable result for document collections with large vocabularies.

12.2.4 Collaborative fuzzy clustering

The same N objects are described at P sites by data $\mathbf{X}[ii] \in \mathbb{R}^{N \times n_{ii}}$, $ii = 1, \ldots, P$. All sites look for c clusters, with memberships $\mu_{ik}[ii]$, prototypes $\mathbf{v}_i[ii]$ in their own feature spaces, and squared distances $D_{ik}^2[ii]$. The collaboration matrix $\alpha[ii, jj] \geq 0$ states how strongly site ii takes the partition of site jj into account. Pedrycz [170] lets site ii minimize

$$Q[ii] = \sum_{i,k} \mu_{ik}^2[ii]\, D_{ik}^2[ii] + \sum_{jj \neq ii} \alpha[ii, jj] \sum_{i,k} \big(\mu_{ik}[ii] - \mu_{ik}[jj]\big)^2 D_{ik}^2[ii], \tag{12.12}$$

in which the other sites enter only through their memberships. With $\bar{\alpha}[ii] = \sum_{jj \neq ii} \alpha[ii, jj]$ and the *communicated memberships*

$$\varphi_{ik}[ii] = \frac{\sum_{jj \neq ii} \alpha[ii, jj]\, \mu_{ik}[jj]}{1 + \bar{\alpha}[ii]}, \tag{12.13}$$

the Lagrange conditions give

$$\mu_{ik}[ii] = \varphi_{ik}[ii] + \frac{1 - \sum_{l=1}^{c} \varphi_{lk}[ii]}{\sum_{l=1}^{c} D_{ik}^2[ii] / D_{lk}^2[ii]}, \tag{12.14}$$

$$\mathbf{v}_i[ii] = \frac{\sum_k w_{ik}[ii]\, \mathbf{x}_k[ii]}{\sum_k w_{ik}[ii]}, \tag{12.15}$$

$$w_{ik}[ii] = \mu_{ik}^2[ii] + \sum_{jj \neq ii} \alpha[ii, jj] \big(\mu_{ik}[ii] - \mu_{ik}[jj]\big)^2$$

The membership is a convex combination of the other sites' memberships, with total weight $\bar{\alpha}/(1 + \bar{\alpha})$, and of the site's own FCM membership; the prototype weights add to the FCM weight the disagreement with the other sites. The algorithm has two phases. In the first, every site runs FCM independently, and the clusters of the sites are matched by the Hungarian algorithm on the membership overlap $\mathbf{U}[1]\mathbf{U}[jj]^\mathsf{T}$, because the sites must refer to the same clusters. In the second, the sites iterate synchronously: each updates its prototypes and memberships by (12.14) and (12.15) from the partitions the others communicated in the previous round, until no membership changes by more than ϵ. A site whose row of the collaboration matrix is zero keeps minimizing its plain FCM functional in the second phase, so it stays at its phase-one solution up to the tolerance of the first phase. The collaboration matrix need not be symmetric: a site may listen without being listened to.

Two quantities summarize the result. The distance between the partitions of two sites, $\frac{1}{N} \sum_{i,k} (\mu_{ik}[ii] - \mu_{ik}[jj])^2$, or their fuzzy Rand index (4.19), measures how well the sites agree; the mean absolute change of a site's memberships from the first to the second phase measures how much the site had to give in, and therefore how much its view of the objects differed from the others'.

12.2.5 Cluster ensembles and consensus functions

Consider a library of Q soft partitions $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(Q)}$ of the same N objects; member q has c_q clusters and membership vectors $\boldsymbol{\mu}_k^{(q)} \in \mathbb{R}^{c_q}$. In `fuzzyclust` the ensemble generates the library itself: every member is the base clusterer (fuzzy c-means by default), fitted with its own random start on a random share of the samples and of the features and with a number of clusters drawn from a given set, and the memberships of *all* samples come from its `predict_proba`. Strehl and Ghosh [171] call the task of combining such partitions without access to the data *knowledge reuse*; the members' cluster labels are arbitrary, so a consensus function must be invariant to them.

Algorithm 12.5 Collaborative fuzzy clustering

Input: sites $\mathbf{X}[1], \ldots, \mathbf{X}[P]$ with the same N rows; c; collaboration matrix $\alpha[ii, jj]$; tolerance ϵ

`// phase 1: independent clustering`

for $ii = 1, \ldots, P$ **do** FCM on $\mathbf{X}[ii]$ with c clusters;

match the clusters of every site to those of site 1 (Hungarian algorithm on $\mathbf{U}[1]\mathbf{U}[ii]^{\mathsf{T}}$);

`// phase 2: collaboration, synchronous`

repeat

  for $ii = 1, \ldots, P$ **do**

    prototypes $\mathbf{v}_i[ii]$ by (12.15) and memberships $\mu_{ik}[ii]$ by (12.14) from the partitions $\mathbf{U}[jj]$ of the previous round;

until *no membership changes by more than* ϵ;

Output: partitions and prototypes of all sites; partition distances; change of every site

In a nutshell Collaborative fuzzy clustering

- The same objects are described at several sites by different features; the sites exchange only partition matrices, never data.
- Each site minimizes its FCM functional plus the distance-weighted disagreement with the other sites' memberships; memberships become convex combinations of own and communicated memberships.
- The collaboration matrix sets how much site ii trusts site jj and may be asymmetric; a larger weight pulls the partitions together monotonically.
- The membership change of a site from the independent to the collaborative phase locates the sites that disagree; $m = 2$.

The co-association matrix. Comparing objects rather than clusters makes the labels irrelevant. The soft co-association of two objects is the library average of the similarity of their membership vectors,

$$s_{kl} = \frac{1}{Q}\sum_{q=1}^{Q} \frac{\left\langle \boldsymbol{\mu}_k^{(q)}, \boldsymbol{\mu}_l^{(q)} \right\rangle}{\left\|\boldsymbol{\mu}_k^{(q)}\right\| \left\|\boldsymbol{\mu}_l^{(q)}\right\|} \quad \text{or} \quad s_{kl} = \frac{1}{Q}\sum_{q=1}^{Q}\sum_{i=1}^{c_q} \mu_{ik}^{(q)}\mu_{il}^{(q)}. \tag{12.16}$$

For crisp members both equal the fraction of members that put k and l into the same cluster, which is the evidence accumulation matrix of Fred and Jain [173] and the similarity of the CSPA algorithm of Strehl and Ghosh. For soft members the inner product (right) is the probability that two independent draws from the membership vectors land in the same cluster; it is below one even for $k = l$, and the more so the fuzzier the member. The cosine (left, the default) normalizes the

membership vectors of every member, so that an object is fully co-associated with itself and with every object that has the same membership vector; the scale of $\mathrm{S} = [s_{kl}]$ then does not depend on the fuzziness or the number of clusters of the members. The matrix S is symmetric with entries in $[0, 1]$.

Consensus functions. Five ways to obtain c consensus clusters are implemented. (i) *FCM on the rows of* S: the row $\mathbf{s}_k$ is the profile of object k's co-associations with all objects, and objects of one consensus cluster have similar profiles; fuzzy c-means with $m = 2$ clusters these N-dimensional rows. In such a high dimension FCM from a random partition drifts to the trivial fixed point of all prototypes at the grand mean, so the prototypes are seeded with k-means++. (ii) *Spectral partitioning* [174] of the graph with adjacency S; it takes the place of the METIS partitioner [175] used in CSPA and in the soft sCSPA of Punera and Ghosh [172]. (iii) *Hierarchical clustering* of the distance $1 - s_{kl}$ (evidence accumulation, section 12.2.6). (iv) The *soft meta-clustering algorithm* sMCLA [172]: every cluster of every member is a *hyperedge*, its membership vector over the N objects; the $\sum_q c_q$ hyperedges are grouped into c meta-clusters by the similarity of these vectors (`fuzzyclust` uses their cosine similarity and spectral partitioning in place of METIS), every meta-cluster is collapsed into its mean membership vector, and an object's consensus membership is its collapsed membership normalized over the meta-clusters. (v) *Information-theoretic k-means* ITK [172]: the concatenated membership vectors of an object, divided by Q, form a distribution $\mathbf{p}_k$ over all clusters of the library, and k-means with the Kullback–Leibler divergence $\mathrm{KL}(\mathbf{p}_k \| \mathbf{m}_i)$ as distortion partitions these distributions; its centroids $\mathbf{m}_i$ are arithmetic means, and restarts keep the smallest total divergence. The hard consensus functions (ii), (iii), and (v) receive soft memberships in the space of the co-associations: the mean row of every consensus cluster is its prototype $\mathbf{z}_i = \sum_{k \in i} \mathbf{s}_k / |i|$, and every object receives the FCM membership of its row,

$$\mu_{ik}^{*} = \frac{1}{\sum_{j=1}^{c} \|\mathbf{s}_k - \mathbf{z}_i\|^2 \,/\, \|\mathbf{s}_k - \mathbf{z}_j\|^2}, \tag{12.17}$$

the same rule as for (i), so that all consensus memberships are on one scale. A new object is passed through every member, its co-associations with the training objects follow from (12.16), and (12.17) (or the meta-clusters of sMCLA) gives its consensus membership. The classical hard consensus functions CSPA and MCLA of Strehl and Ghosh are (ii) and (iv) applied to crisp members.

12.2.6 Evidence accumulation and the lifetime criterion

Fred and Jain [173] use the co-association matrix of members that deliberately *over-segment* the data: k-means runs with many more clusters than expected and a different number for each run. Objects of one natural group are split differently by different members but stay together in most of them; objects of different groups are rarely together. The distance $d_{kl} = 1 - s_{kl}$ is clustered hierarchically (single

Algorithm 12.6 Fuzzy cluster ensemble

Input: data $\mathbf{X}$; base clusterer; library size Q; sample and feature fractions; set of member cluster numbers; c (or none); consensus function

for $q = 1, \dots, Q$ **do**
  draw a sub-sample, a feature subset and c_q; fit the base clusterer with its own random start;
  $\mathbf{U}^{(q)} \leftarrow$ its memberships of all N objects (`predict_proba`);
co-association matrix $\mathbf{S}$ by (12.16);
if *c is not given* **then** $c \leftarrow$ the number of clusters of longest lifetime (algorithm 12.7);
consensus partition by FCM on the rows of $\mathbf{S}$, spectral partitioning, evidence accumulation, sMCLA or ITK; soft memberships by (12.17);

Output: consensus memberships $\mathbf{U}^*$; $\mathbf{S}$; the library

In a nutshell Fuzzy cluster ensemble

- Combines a library of soft partitions (random starts, sub-samples, feature subsets, various c) through the co-association matrix, which needs no matching of cluster labels.
- The consensus is typically better than the average member and far more reproducible than a single run; most of the gain comes with the first few members.
- Parameters: library size, sample and feature fractions, member cluster numbers, consensus function (FCM on the rows of $\mathbf{S}$, spectral, hierarchical, sMCLA, ITK), similarity (cosine or inner product).
- Costs Q fits plus an $N \times N$ matrix; use for unstable clusterers, weak views of the data, or to choose c (next section).

or average link); let $h_1 \le h_2 \le \cdots \le h_{N-1}$ be the merge heights of the dendrogram and $h_N = 1$ the largest possible distance. Cutting the dendrogram at a threshold in $[h_{N-k}, h_{N-k+1})$ yields exactly k clusters, and the length of this interval,

$$\tau_k = h_{N-k+1} - h_{N-k}, \qquad k = 1, 2, \dots, \tag{12.18}$$

is the *lifetime* of the k-cluster partition. The chosen number of clusters is the one of longest lifetime, the partition least sensitive to the choice of the threshold. The criterion inherits the behavior of the linkage. Average link copes with soft members; single link merges two groups as soon as one chain of objects connects them, and since the co-associations of soft members between neighboring groups never vanish, the one-cluster partition then lives longest. The members must also

Algorithm 12.7 Evidence accumulation with the lifetime criterion

1. Generate a library of over-segmenting partitions (e.g. FCM with c_q drawn from a range well above the expected number of groups) and its co-association matrix $\mathbf{S}$.
2. Cluster the distance $1 - s_{kl}$ hierarchically (average or single link); merge heights $h_1 \leq \cdots \leq h_{N-1}$, $h_N = 1$.
3. Lifetimes τ_k by (12.18) for $k = 1, \ldots, k_{\max}$; $c = \arg\max_k \tau_k$.
4. Cut the dendrogram into c groups (or pass c to another consensus function).

In a nutshell Evidence accumulation

- The chosen number of clusters is that of the partition of the co-association dendrogram that persists over the widest range of thresholds.
- Needs over-segmenting members; beyond that, the exact range of member cluster numbers hardly matters.
- Average link works with soft members; single link can be defeated by chains of intermediate objects.
- Parameters: range of member cluster numbers, linkage, largest k considered.

over-segment: a member with fewer clusters than groups merges some groups, and no amount of evidence separates what no member separates.

12.3 Using it in fuzzyclust

The four sub-packages follow the conventions of the toolbox. The supervised estimators derive from `BaseFuzzyClusterer` and take the supervision as arguments of `fit`; the co-clustering estimators and the ensemble are scikit-learn clusterers with `fit(X)`; the collaborative estimator takes a *list* of data sites. All accept pandas input and have a `random_state`, and their partitions can be compared with the fuzzy Rand indices of `fuzzyclust.validity` (section 4.2.8).

`fuzzyclust.semisupervised.SemiSupervisedFCM`

`n_clusters=3`, `alpha=1.0|"auto"` (N/N_b), `m=2.0` (fixed), `tol`, `max_iter`, `init="labels"|"random"|"kmeans++"|array`, `n_init`, `random_state`, `verbose`.
`fit(X, y)` with `y` a label vector (`-1` = unlabeled) or a label-membership matrix (N, c) whose zero rows mark the unlabeled samples. Attributes `membership_`, `cluster_centers_`, `labels_`, `classes_`, `label_mapping_`

(class of every cluster, `-1` for unsupervised ones), `labelled_mask_`, `label_membership_`, `alpha_`, `covariances_`; methods `predict_classes(X)`, `score_labels(X, y)`.

`fuzzyclust.semisupervised.PairwiseConstrainedFCM`

`n_clusters=3`, `alpha=1.0|"auto"`, `agglomeration=0.0` (η_0), `agglomeration_decay=10.0` (τ), `min_cardinality=1.0`, `m=2.0` (fixed), `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`.
`fit(X, must_link=pairs, cannot_link=pairs)`. Attributes `n_clusters_`, `membership_`, `cluster_centers_`, `must_link_`, `cannot_link_`, `must_link_satisfied_`, `cannot_link_satisfied_`, `alpha_`, `cardinalities_`. Utilities `constraint_satisfaction(labels, must_link, cannot_link)`, `check_pairwise_constraints`; generators `make_partially_labeled_pair`, `make_partial_labels`, `make_pairwise_constraints`; plots `plot_label_propagation`, `plot_pairwise_constraints`.

`fuzzyclust.coclustering.FCCM, FCoDoK`

`n_clusters=3`, `lambda_u=1.0` (λ_u or T_u), `lambda_w=1.0` (λ_w or T_w), `tol`, `max_iter`, `init="random"|"kmeans++"|array`, `n_init` (largest objective kept), `random_state`, `verbose`.
`fit(R)` with a non-negative matrix. Attributes `membership_` (N, c), `item_membership_` (M, c), `labels_`, `item_labels_`, `aggregation_`, `objective_`, `objective_history_`; `transform` returns the aggregations a_{ik}, `predict_proba` the object memberships of new rows. Generator `make_block_matrix`; plots `plot_coclusters` (reordered heat map), `plot_item_memberships`.

`fuzzyclust.collaborative.CollaborativeFCM`

`n_clusters=3`, `alpha=0.5` (scalar or $P \times P$ matrix), `m=2.0` (fixed), `tol`, `max_iter`, `init`, `n_init`, `random_state`, `verbose`.
`fit([X_1, ..., X_P])`. Attributes (lists over the sites) `memberships_`, `cluster_centers_`, `labels_`, `independent_memberships_`, `independent_centers_`; arrays `alpha_`, `objective_`, `objective_history_`, `partition_distances_`, `collaboration_change_`, `consensus_membership_`. Methods `predict_proba`, `predict`, `transform` on lists of sites, `site_agreement(index=fuzzy_rand_index)`. Utility `align_partitions`; generator `make_collaborative_sites`; plots `plot_site_partitions`, `plot_collaboration_trajectory`.

fuzzyclust.ensemble.FuzzyClusterEnsemble

`n_clusters=3|None` (lifetime criterion), `base_estimator=None` (FCM; any clusterer with `predict_proba`), `n_estimators=10`, `base_n_clusters=None|int|sequence`, `sample_fraction=1.0`, `feature_fraction=1.0`, `consensus="fcm"|"spectral"|"hierarchical"|"smcla"|"itk"`, `linkage="average"|"single"|"complete"`, `similarity="cosine"|"product"`, `max_clusters=20`, `random_state`, `verbose`.
Attributes `base_estimators_`, `base_memberships_`, `base_feature_subsets_`, `coassociation_`, `consensus_membership_` (= `membership_`), `labels_`, `n_clusters_`, `consensus_centers_`, `lifetimes_`, `linkage_`, `meta_cluster_labels_`, `base_agreement_`. Functions `coassociation_matrix`, `evidence_accumulation`, `soft_mcla`, `information_theoretic_kmeans`, `cspa`, `mcla`, `stability_curve(X, n_estimators_grid, n_repeats, index)`; generator `make_overlapping_blobs`; plots `plot_coassociation`, `plot_lifetimes`.

Partial labels and pairwise constraints are passed to `fit`:

```
fcm = FCM(n_clusters=2, random_state=0).fit(X)
ssfcm = SemiSupervisedFCM(n_clusters=2, alpha=1.0, random_state=0).fit(X,
    y_partial)
print(f"FCM ARI = {adjusted_rand_score(y, fcm.labels_):.2f}, "
      f"SSFCM ARI = {adjusted_rand_score(y, ssfcm.labels_):.2f}, "
      f"accuracy of the class prediction = {ssfcm.score_labels(X, y):.2f}")
```

```
must_link, cannot_link = make_pairwise_constraints(y, n_must_link=50,
    n_cannot_link=50, random_state=0)
fcm = FCM(n_clusters=2, random_state=0).fit(X)
pcca = PairwiseConstrainedFCM(n_clusters=2, alpha=5.0, random_state=0)
pcca.fit(X, must_link=must_link, cannot_link=cannot_link)
print(f"FCM  ARI = {adjusted_rand_score(y, fcm.labels_):.2f}")
print(f"PCCA ARI = {adjusted_rand_score(y, pcca.labels_):.2f}, satisfied
    must-link "
      f"{pcca.must_link_satisfied_:.2f}, cannot-link
    {pcca.cannot_link_satisfied_:.2f}")
```

A co-clustering returns memberships for the rows and for the columns:

```
fccm = FCCM(n_clusters=4, lambda_u=0.1, lambda_w=20.0, n_init=5,
    random_state=0).fit(R)
print(f"FCCM documents ARI = {adjusted_rand_score(doc_topic,
    fccm.labels_):.2f}, "
      f"topical terms ARI = {adjusted_rand_score(term_topic[topical],
    fccm.item_labels_[topical]):.2f}")
```

The collaborative estimator takes the sites as a list and reports how much every site moved:

```
independent = CollaborativeFCM(n_clusters=3, alpha=0.0,
    random_state=0).fit(Xs)
collab = CollaborativeFCM(n_clusters=3, alpha=3.0, random_state=0).fit(Xs)
for name, model in (("independent", independent), ("alpha = 3", collab)):
    print(f"{name:12s}: ARI per site = "
          + ", ".join(f"{adjusted_rand_score(y, lab):.2f}" for lab in
    model.labels_)
          + f";  membership change =
    {np.round(model.collaboration_change_, 3)}")
```

An ensemble is configured by its library and its consensus function; with `n_clusters=None` it chooses the number of clusters, and `stability_curve` measures its reproducibility:

```
ens = FuzzyClusterEnsemble(n_clusters=4, n_estimators=20,
    sample_fraction=0.3, feature_fraction=0.34,
                           consensus="fcm", random_state=0).fit(X)
members = [adjusted_rand_score(y, U.argmax(axis=1)) for U in
    ens.base_memberships_]
print(f"members: mean ARI = {np.mean(members):.2f}, best =
    {np.max(members):.2f};  "
      f"consensus ARI = {adjusted_rand_score(y, ens.labels_):.2f}")
```

```
ens = FuzzyClusterEnsemble(n_clusters=None, base_n_clusters=range(6, 13),
    n_estimators=30, linkage="average",
                           random_state=0).fit(X)
print(f"lifetime criterion: {ens.n_clusters_} clusters, ARI =
    {adjusted_rand_score(y, ens.labels_):.2f}")
print("lifetimes of k = 1..8:", np.round(ens.lifetimes_[:8], 3))
```

```
table = stability_curve(X10, n_estimators_grid=(1, 3, 10, 20),
    n_repeats=5, n_clusters=10,
                        random_state=0, index=fuzzy_adjusted_rand_index)
print(table.round(3))
```

12.4 Examples

12.4.1 How many labels, how much weight?

The two parallel bars of 150 points each (fig. 12.5) are split by FCM across the bars (ARI 0.39). With 10 % labeled samples and $\alpha = 1$ the prototypes move toward the bars and the ARI rises to 0.60; 87 % of all samples are assigned to their class. Plotted against the position along the bars (fig. 12.6), the memberships are clear in the middle of the bars and drift toward 0.5 at their ends, where both prototypes are almost equally far away. At the ends spherical prototypes cannot tell the bars apart, and that is where the remaining errors lie. The study of fig. 12.7 varies the label fraction from 1 to 30 % and the weight. Up to $\alpha = 3$ more labels always help ($\alpha = 1$: from 0.38 at 1 % to 0.77 at 30 %), while the weight acts non-monotonically:

$\alpha = 10$ never exceeds 0.56, and the paper's rule $\alpha = N/N_b$, which gives a large weight when labels are scarce, does no better than $\alpha = 10$ up to 10 % labels and approaches $\alpha = 3$ only at 30 % (0.69). The reason is the weighting in (12.3): once the labeled memberships match their labels, their supervision weight vanishes, while the unlabeled samples are weighted by $1 + \alpha$.

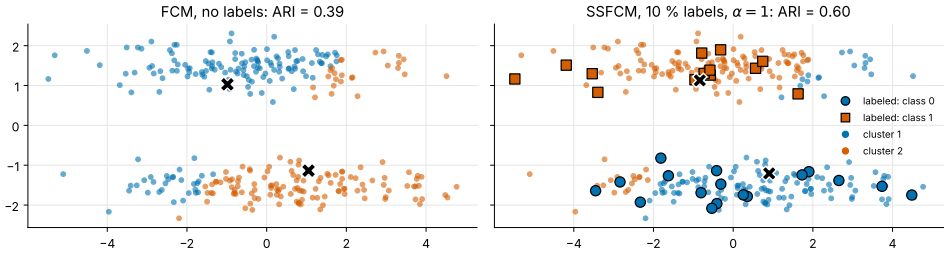


Figure 12.5. FCM splits the two bars across (left); with 10 % labeled samples (large markers) and $\alpha = 1$ the partition turns toward the bars (right).

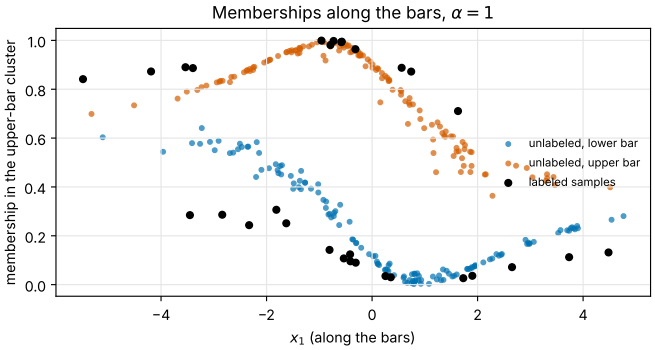


Figure 12.6. Membership in the upper-bar cluster along the bars: clear in the middle, ambiguous at the ends; labeled samples (black) are pulled toward their label.

12.4.2 Must-link, cannot-link, and competitive agglomeration

On the same bars, 50 must-link and 50 cannot-link pairs drawn from the classes are given (fig. 12.8). FCM violates 24 of them; PCCA with $\alpha = 5$ satisfies all and reaches an ARI of 0.95. The penalty study (fig. 12.9) shows that the fewer the pairs, the larger the penalty needed: 100 pairs of each kind reach an ARI of 0.99 at $\alpha = 2$, 25 pairs need $\alpha = 10$, and 10 pairs reach only 0.7 even at $\alpha = 20$, although nine tenths of them are satisfied already at $\alpha = 0.5$, because a handful of pairs can be honored by bending the boundary locally. The automatic penalty, renewed as the FCM term over the number of constraints, ends at $\alpha = 4.3$ for 100 pairs and at 42 for 10 pairs, and gives ARIs of 0.99 and 0.72. Started with eight clusters on three Gaussian groups (fig. 12.10), competitive agglomeration keeps all eight for $\eta_0 \leq 1$, ends with exactly three for η_0 from 5 to 20 at the default decay $\tau = 10$,

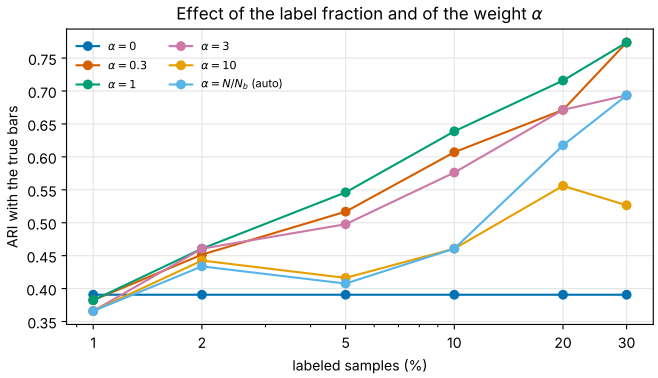


Figure 12.7. Agreement with the true bars against the share of labeled samples for several weights α: more labels help, and moderate weights help most.

and collapses to one cluster at $\eta_0 = 40$; a longer competition ($\tau = 30$) shifts the window toward weaker weights.

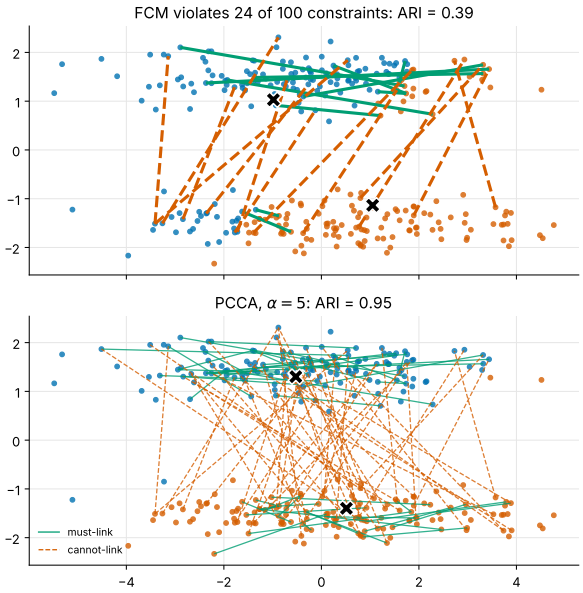


Figure 12.8. The 24 constraints that FCM violates (top) and the partition of PCCA with $\alpha = 5$, which satisfies all 100 constraints (bottom); green: must-link, vermilion dashed: cannot-link.

12.4.3 Co-clustering a document–term matrix

A synthetic corpus of 180 documents in four topics of 30 to 60 documents, with topic vocabularies of 15 to 30 terms and 20 general terms, has Poisson counts of mean 3 inside the topic blocks and 0.3 elsewhere; rows and columns are shuffled (fig. 12.11,

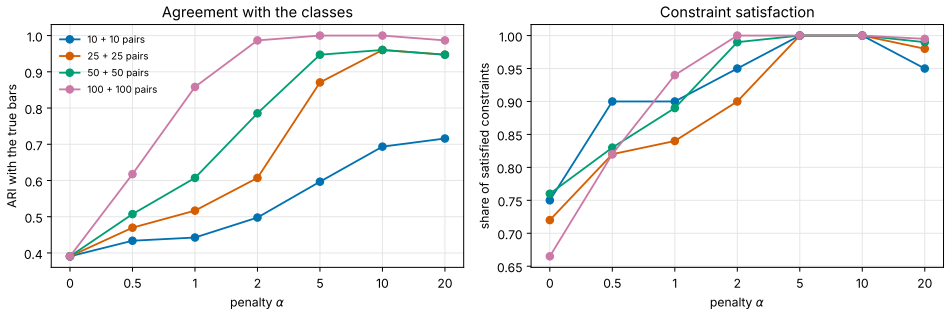


Figure 12.9. Agreement with the classes (left) and share of satisfied constraints (right) against the penalty α for 10 to 100 pairs of each kind.

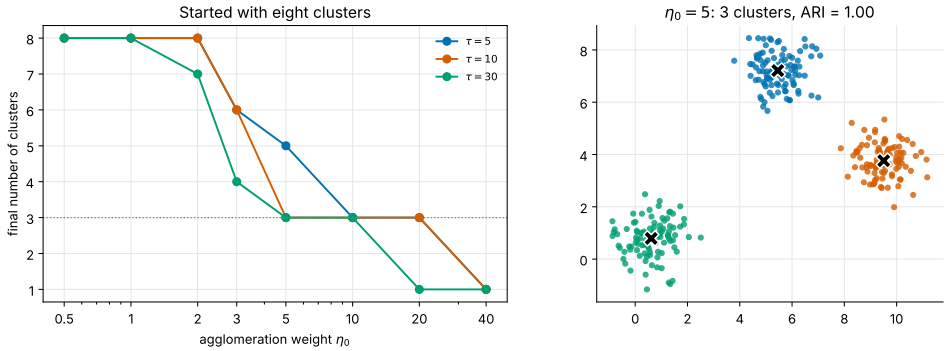


Figure 12.10. Competitive agglomeration from eight clusters: final number of clusters against η_0 for three decays (left) and the three clusters found with $\eta_0 = 5$ (right).

left). FCCM with $\lambda_u = 0.1$ and $\lambda_w = 20$ recovers the topics of the documents and of the topical terms exactly (both ARIs 1.0), and the matrix reordered by the co-clusters shows the four blocks (right). The scaled item memberships (fig. 12.12) rank every topic's own terms by frequency and give the general terms a membership near zero in all co-clusters. The grid study (fig. 12.13) shows the units of the two weights at work. A small λ_u gives nearly crisp document memberships, $\lambda_u \geq 1$ fuzzy ones, and $\lambda_u = 3$ dissolves them. For λ_w there is a window, from 20 to 200 for $\lambda_u \leq 0.1$, below which a co-cluster concentrates on a few frequent terms and above which the term memberships become uniform; the window closes from above as λ_u grows (20–50 at $\lambda_u = 0.3$, only 20 at $\lambda_u = 1$). Fuzzy CoDoK recovers the topics exactly for $T_u \leq 0.3$ and every T_w from 20 to 1000, and its clipped item memberships list the topic vocabularies with exact zeros elsewhere (fig. 12.14).

12.4.4 Three collaborating sites

Three sites describe the same 300 objects in three groups with feature spreads 0.6, 1.2, and 1.8, and at the third site 15 % of the objects are moved to another group (fig. 12.15). Independently, the sites reach ARIs of 1.00, 0.93, and 0.37. With

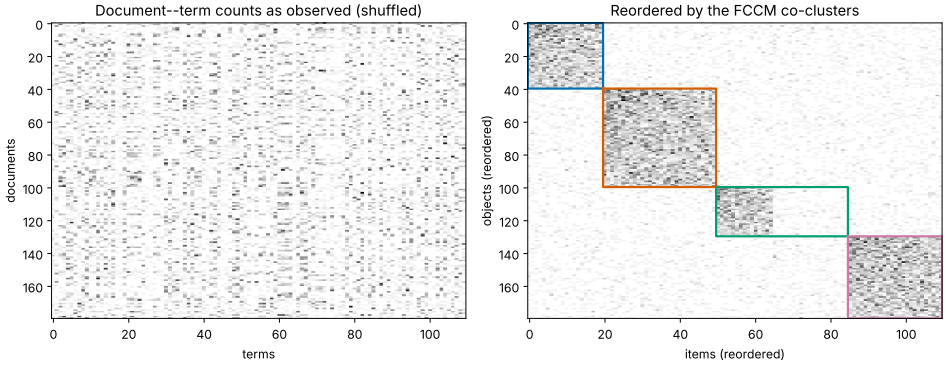


Figure 12.11. A shuffled document–term matrix (left) and the same matrix reordered by the four FCCM co-clusters (right).

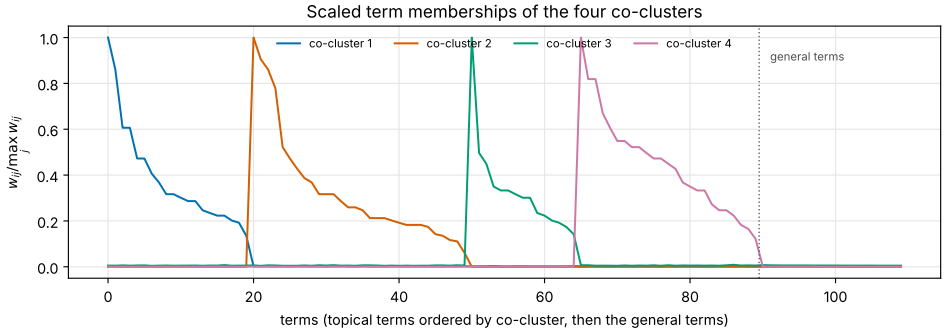


Figure 12.12. Scaled term memberships of the four co-clusters: every topic owns its vocabulary, and the general terms (right of the dotted line) belong to none.

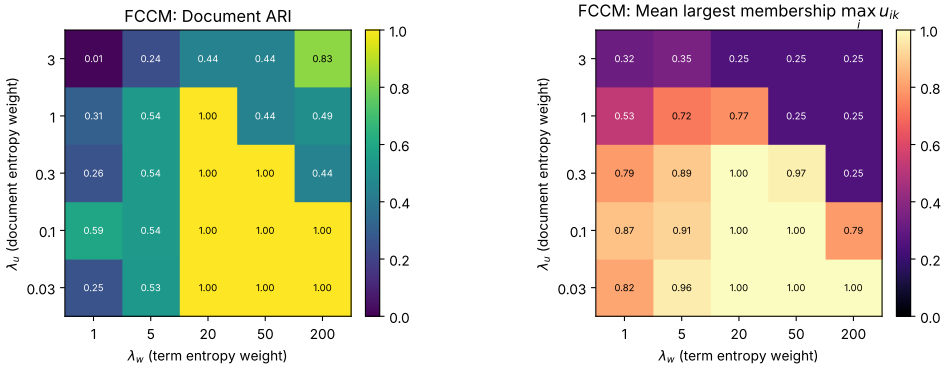


Figure 12.13. FCCM on a grid of the two entropy weights: agreement with the topics (left) and crispness of the document memberships (right).

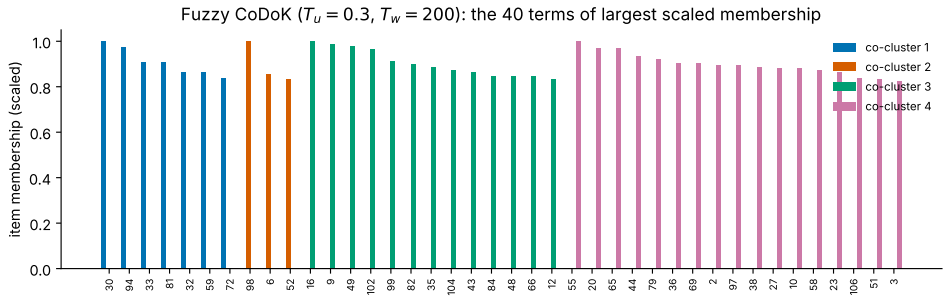


Figure 12.14. Fuzzy CoDoK: the 40 terms of largest scaled membership, grouped by co-cluster.

$\alpha = 3$ the clear site keeps its partition, the second improves to 0.98, and the third follows the majority even on its moved objects (ARI 0.98 with the common groups), so that its clusters now overlap in its own feature space. The study of fig. 12.16 shows the fuzzy Rand index between every pair of sites rising monotonically with α toward one, site 3 joining the majority from $\alpha \approx 1$, and site 3 giving in most (mean membership change 0.13 against 0.10 and 0.07); every site moves, because a membership pulled toward the others is also a softer one. Asymmetric collaboration matrices, whose results the example prints, give trust selectively: when only site 3 listens, sites 1 and 2 keep their independent partitions and site 3 is corrected alone (ARI 1.00); when only the clear site 1 listens, it is pulled toward the noisier sites and drops to 0.91.

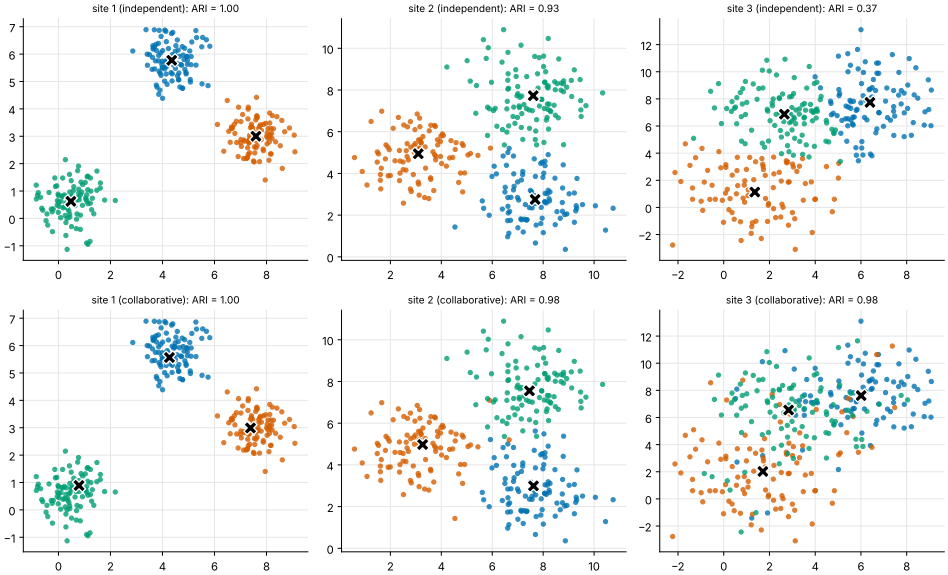


Figure 12.15. The three sites clustered independently (top) and with collaboration $\alpha = 3$ (bottom); colors are the clusters matched across the sites.

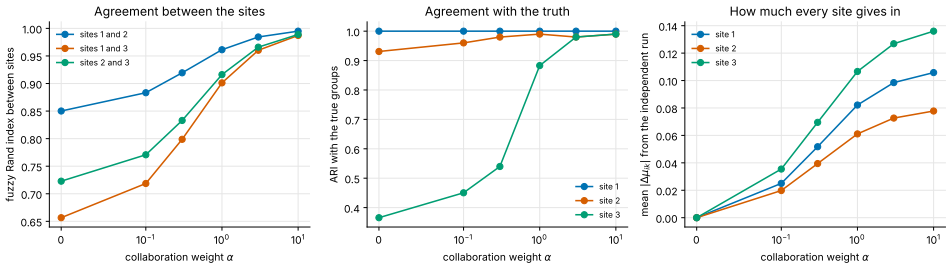


Figure 12.16. Against the collaboration weight: fuzzy Rand index between the sites (left), agreement of every site with the true groups (middle), and mean membership change of every site (right).

12.4.5 A library beats a single run

Four overlapping groups in six dimensions are clustered by an ensemble of 20 FCM members, each seeing two random features and 30 % of the samples (fig. 12.17). The members reach a mean ARI of 0.37 and at best 0.77; their consensus reaches 0.86, better than every member because the members err in different directions, and close to the 0.91 of one FCM run on all data. The co-association matrix ordered by the consensus shows the four blocks. Over the library size (fig. 12.18), a single member is as weak as any, three to ten members bring most of the gain, and the five consensus functions behave alike, with sMCLA a few points behind from ten members on. Accuracy needs ground truth, but reproducibility does not: on ten overlapping groups in the plane, FCM with ten clusters lands in different local optima, and the fuzzy adjusted Rand index between repeated consensus partitions rises from 0.62 ± 0.26 for single runs to 0.86 ± 0.04 for libraries of 20 (fig. 12.19).

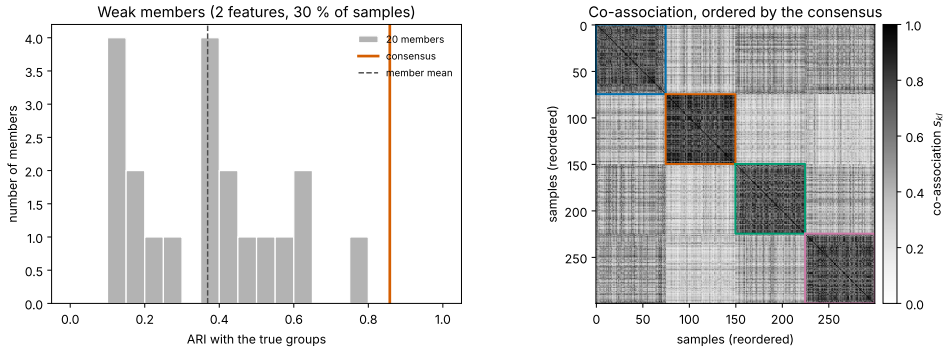


Figure 12.17. Twenty weak members and their consensus: distribution of the member ARIs with the consensus and the member mean (left), co-association matrix ordered by the consensus (right).

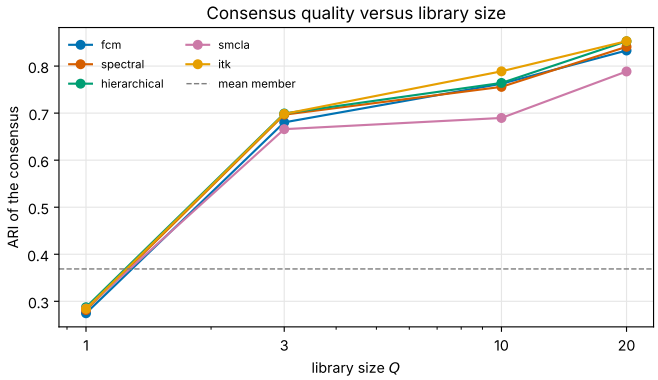


Figure 12.18. ARI of the consensus against the library size for the five consensus functions.

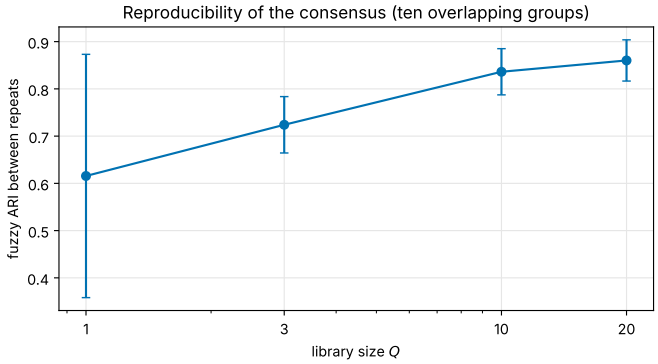


Figure 12.19. Fuzzy adjusted Rand index between repeated consensus partitions against the library size: the ensemble makes an unstable clusterer reproducible.

12.4.6 Choosing the number of clusters by the lifetime

Five groups are clustered by 30 FCM members with 6 to 12 clusters each (fig. 12.20). The average-link dendrogram of $1 - s_{kl}$ gives the five-cluster partition a lifetime of 0.53, against at most 0.06 for any other number, and the consensus recovers the groups exactly. The top of the dendrogram (fig. 12.21) shows where this comes from: below the band the groups fall apart into the fragments the members produced, and above it neighboring groups merge. The study of fig. 12.22 varies the members' numbers of clusters and the linkage. Members with fewer clusters than groups (2 to 4) give three clusters, every over-segmenting range gives five with average link, and single link always returns one cluster, because chains of objects between the groups merge them at a low height. The repetition on four and six groups, printed by the example, finds four groups from every range and six groups from members with 6 to 12 or 10 to 20 clusters (five from members with 4 to 8).

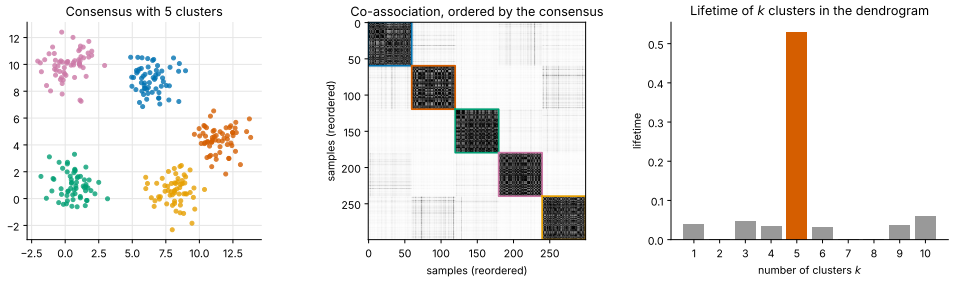


Figure 12.20. Evidence accumulation on five groups: the consensus with the number of clusters chosen by the lifetime (left), the co-association matrix (middle), and the lifetimes of k clusters (right).

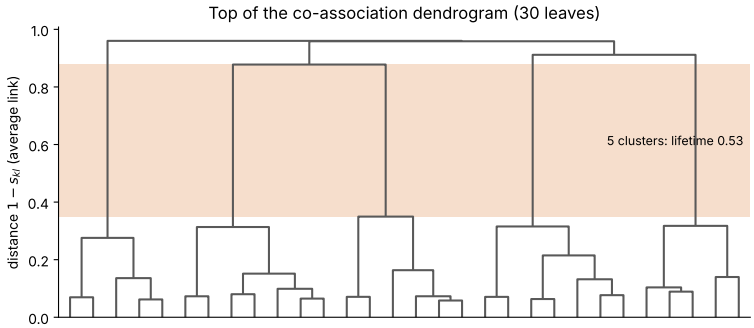


Figure 12.21. Top of the average-link dendrogram of the co-association distances; the band is the range of thresholds that yields five clusters.

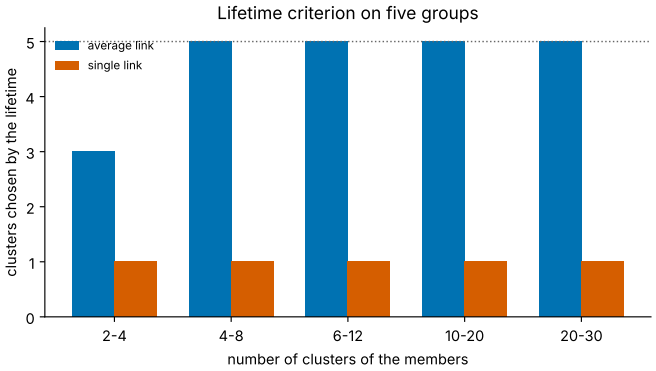


Figure 12.22. Number of clusters chosen by the lifetime against the range of the members' numbers of clusters, for average and single link.

12.5 Further reading

Partial supervision by labels goes back to Pedrycz and Waletzky [165]; pairwise constraints combined with competitive agglomeration are due to Grira, Crucianu, and Boujemaa [166], and competitive agglomeration itself to Frigui and Krishnapuram [167]. Fuzzy co-clustering with entropy regularization was introduced by Oh, Honda, and Ichihashi [168], building on Miyamoto and Mukaidono [23], and the quadratic variant by Kummamuru, Dhawale, and Krishnapuram [169]. Collaborative clustering is due to Pedrycz [170]. Cluster ensembles were framed by Strehl and Ghosh [171] (CSPA, MCLA, and the METIS graph partitioner of Karypis and Kumar [175]), extended to soft partitions by Punera and Ghosh [172] (sCSPA, sMCLA, ITK), and turned into a criterion for the number of clusters by Fred and Jain [173]; the spectral partitioning follows Ng, Jordan, and Weiss [174]. Partitions are compared with the fuzzy Rand indices of Hüllermeier et al. [45] and Campello [46] and the chance correction of Hubert and Arabie [44]; the c-means functional is that of Bezdek [3], and Jain and Dubes [4] cover hierarchical clustering in general.

APPENDIX

A



API reference

This appendix lists every public class and function of `fuzzyclust`, sub-package by sub-package, with the first sentence of its docstring. The lists are generated from the `__all__` attribute of each module by `book/build_api_appendix.py`, so they are complete by construction; the full signatures, parameters, and attributes are in the online documentation and in the docstrings (`help(fuzzyclust.cluster.GK)`). The shared constructor parameters, fitted attributes, and methods of all clustering estimators are described once in section 2.3. The notation of the docstrings is that of table 2.1.

A.1 `fuzzyclust.base`: Base class

Name	Kind	Description
`BaseFuzzyClusterer`	class	Abstract base of the prototype-based clustering algorithms
`EPS`	constant	Small constant added to squared distances before negative powers (10^{-10})

A.2 `fuzzyclust.cluster`: Partitioning algorithms

Name	Kind	Description
`KMeans`	class	Hard c-means (k-means) clustering with the fuzzyclust interface
`KMedoids`	class	Hard k-medoids clustering (prototypes are data points)

Name	Kind	Description
`FCM`	class	Fuzzy c-means clustering (Dunn, Bezdek)
`GK`	class	Gustafson–Kessel (GK) fuzzy clustering
`GG`	class	Gath–Geva (GG) fuzzy clustering
`RegularizedFCM`	class	Fuzzy c-means by entropy or quadratic regularization, with optional deterministic annealing
`PolynomialFuzzifierFCM`	class	Fuzzy c-means with the polynomial fuzzifier of Klawonn and Höppner
`entropy_memberships`	function	Memberships of the entropy-regularized c-means, $\mu_{ik} \propto \exp(-D_{ik}^2/\lambda)$
`quadratic_memberships`	function	Memberships of the quadratically regularized c-means (Miyamoto and Umayahara)
`polynomial_memberships`	function	Memberships of fuzzy c-means with the polynomial fuzzifier (Klawonn and Höppner)
`polynomial_fuzzifier`	function	Polynomial fuzzifier $g(\mu) = \frac{1-\beta}{1+\beta}\mu^2 + \frac{2\beta}{1+\beta}\mu$ of Klawonn and Höppner
`SubtractiveClustering`	class	Subtractive clustering (Chiu, 1994): cluster centers from the potential of the data points
`MountainClustering`	class	Mountain method (Yager and Filev, 1994): cluster centers from a mountain function on a grid
`CompatibleClusterMerging`	class	Compatible cluster merging: find the number of clusters by merging from a large start
`cluster_similarity`	function	Pairwise similarity of fuzzy clusters from their membership columns (fuzzy Jaccard index)
`geometric_compatibility`	function	Compatibility of cluster pairs from centers and covariances (Krishnapuram and Freg)
`reference_fuzziness`	function	Rescale an FCM-type partition obtained with exponent m to the one exponent m_ref would give

A.3 `fuzzyclust.validity`: Validity indices

Name	Kind	Description
`partition_coefficient`	function	Partition coefficient (Bezdek)
`classification_entropy`	function	Classification (partition) entropy (Bezdek)
`modified_partition_coefficient`	function	Modified partition coefficient (Davé, 1996)
`partition_index`	function	Partition index SC (Bensaid et al., 1996)
`separation_index`	function	Separation index S: Xie–Beni in its original form, with squared memberships (Xie and Beni, 1991)

Name	Kind	Description
`xie_beni`	function	Xie–Beni index (Xie and Beni, 1991)
`fukuyama_sugeno`	function	Fukuyama–Sugeno index (1989)
`dunn_index`	function	Dunn's index (1974) on the hardened partition
`alternative_dunn_index`	function	Alternative Dunn index (ADI) of the MATLAB toolbox
`silhouette`	function	Mean silhouette coefficient (Rousseeuw, 1987) of the hardened partition
`fuzzy_hypervolume`	function	Fuzzy hypervolume $FHV = \sum_i \sqrt{\det \mathbf{F}_i}$ (Gath and Geva, 1989)
`partition_density`	function	Partition density $PD = \sum_i S_i / FHV$ (Gath and Geva, 1989)
`average_partition_density`	function	Average partition density (Gath and Geva)
`pbmf_index`	function	PBM-F index (Pakhira, Bandyopadhyay and Maulik, 2004)
`kwon_index`	function	Kwon's index (1998)
`pcaes_index`	function	Partition coefficient and exponential separation, PCAES (Wu and Yang, 2005)
`fuzzy_rand_index`	function	Fuzzy Rand index between two fuzzy partitions of the same samples
`fuzzy_adjusted_rand_index`	function	Fuzzy adjusted Rand index (chance-corrected), from the fuzzy pair counts of fuzzy_rand_index
`fuzzy_jaccard_index`	function	Fuzzy Jaccard index $a/(a+b+c)$ from the fuzzy pair counts of fuzzy_rand_index
`stability_selection`	function	Stability of the partition against resampling, for every number of clusters
`validity_indices`	function	Evaluate several validity indices at once
`scan_n_clusters`	function	Fit the estimator for several numbers of clusters and tabulate validity indices
`suggest_n_clusters`	function	Optimal number of clusters suggested by every index of a scan table
`INDEX_DIRECTION`	constant	For every validity index, whether its optimum is the maximum or the minimum

A.4 `fuzzyclust.extensions`: Robust and shape-oriented variants

Name	Kind	Description
`PCM`	class	Possibilistic c-means clustering (Krishnapuram and Keller)
`PFCM`	class	Possibilistic fuzzy c-means clustering (Pal, Pal, Keller and Bezdek)
`NoiseClustering`	class	Noise clustering: fuzzy c-means with a virtual noise cluster (Davé)

Name	Kind	Description
`KernelFCM`	class	Kernel fuzzy c-means: FCM in the feature space of a Mercer kernel
`rbf_kernel`	function	Gaussian (RBF) kernel matrix $K_{kl} = \exp(-\gamma\|\mathbf{x}_k - \mathbf{y}_l\|^2)$
`KERNELS`	constant	Names of the kernels accepted by `KernelFCM`
`FuzzyCVarieties`	class	Fuzzy c-varieties (FCV) clustering: linear-variety prototypes (Bezdek et al.)
`FuzzyCShells`	class	Fuzzy c-(spherical) shells: circles / hyperspheres as prototypes
`VolumePrototypeFCM`	class	Fuzzy c-means with volume prototypes and optional adaptive cluster merging (Kaymak and Setnes, 2002)
`SCAD`	class	Simultaneous clustering and attribute discrimination (Frigui and Nasraoui, 2004)
`FeatureWeightedFCM`	class	Fuzzy c-means with global attribute weights (after Huang et al., 2005)
`plot_shells`	function	Scatter the data and draw the fitted circles of a FuzzyCShells model
`plot_varieties`	function	Scatter the data and draw the fitted lines of a FuzzyCVarieties model (n_dims=1)
`plot_typicality`	function	Scatter plot colored by the maximal typicality $\max_i t_{ik}$ of every sample
`make_circles`	function	Intersecting circles (shell clusters) with optional arcs and outliers
`make_close_blobs`	function	Two Gaussian clusters of different size and density close to each other
`make_crossing_lines`	function	Line segments that cross at a common point
`make_collinear_segments`	function	Two collinear segments separated by a gap, optionally with a third, oblique segment

A.5 `fuzzyclust.graph`: Graph-based clustering

Name	Kind	Description
`MSTClustering`	class	Hybrid clustering with a minimal spanning tree and Gath–Geva models
`GeodesicFCM`	class	Geodesic-distance-based fuzzy c-medoid clustering (Feil and Abonyi, Algorithm II)
`IsomapFCM`	class	Fuzzy c-means on the Isomap projection (Feil and Abonyi, Algorithm I)
`minimum_spanning_tree`	function	Euclidean minimal spanning tree of the samples
`fuzzy_cluster_similarity`	function	Pairwise similarity of fuzzy clusters seen as fuzzy sets over the samples
`neighbors_graph`	function	Symmetric k-nearest-neighbor (or ε-radius) graph with Euclidean weights

Name	Kind	Description
geodesic_distances	function	Graph (geodesic) distances between all pairs of samples, Isomap style
isomap_embedding	function	Classical multidimensional scaling of a (geodesic) distance matrix (the Isomap projection)
plot_mst	function	Draw the minimal spanning tree of a fitted MSTClustering
plot_neighbors_graph	function	Draw a neighborhood graph (sparse adjacency) over the data
plot_geodesic_path	function	Highlight the shortest graph path between two samples and the Euclidean chord
plot_similarity_dendrogram	function	Dendrogram of fuzzy clusters built from their similarity matrix
make_chained_blobs	function	Two Gaussian blobs connected by a sparse chain of points (the *chaining effect*)
make_density_blobs	function	A diffuse cluster next to two tight ones (very different densities)
make_s_curve	function	Points on the three-dimensional S-curve, a two-dimensional manifold (wraps scikit-learn)

A.6 `fuzzyclust.viz`: Visualization

Name	Kind	Description
pca_projection	function	Project the data and the cluster prototypes onto the first principal components
Sammon	class	Sammon's nonlinear mapping to a low-dimensional space
FuzzySammon	class	Fuzzy Sammon mapping (FUZZSAM) of a fitted fuzzy clustering
FuzzySOM	class	Fuzzy self-organizing map: regularized fuzzy c-means on a grid
grid_laplacian	function	Regularization matrix $\mathbf{L} = \mathbf{G}^T\mathbf{G}$ of a rectangular grid
sammon_stress	function	Sammon's stress of an embedding Y of the data X
recomputed_membership	function	Fuzzy memberships recomputed in the projected space
projection_quality	function	Measures of how well a 2-D (or q-D) projection represents a fuzzy partition
plot_projection	function	Scatter plot of a 2-D projection with projected prototypes and membership contours
plot_stress	function	Stress versus iteration for one or several stress_history_ arrays

Name	Kind	Description
`plot_map`	function	Draw a fuzzy SOM as a colored grid: U-matrix, a component plane or the hit histogram
`plot_som_lattice`	function	Prototypes of a fuzzy SOM in the data space, with grid neighbors connected
`make_ellipsoids`	function	Elongated Gaussian clusters with random orientations in n dimensions
`make_spiral_3d`	function	Noisy points along a part of a helix in 3-D (Migály et al., first example)
`make_hemisphere`	function	Noisy points on the upper half of the unit sphere (Migály et al., second example)

A.7 `fuzzyclust.models`: Model identification

Name	Kind	Description
`ModifiedGG`	class	Modified Gath–Geva clustering: clusters are interpretable Takagi–Sugeno rules
`TakagiSugeno`	class	Takagi–Sugeno (TS) fuzzy model with rules extracted from fuzzy clusters
`FuzzyCRegression`	class	Fuzzy c-regression models (FCRM): clusters are regression hyperplanes
`SupervisedFuzzyClassifier`	class	Fuzzy rule-based classifier identified by supervised Gath–Geva type clustering
`FuzzyPartitionDiscretizer`	class	Supervised fuzzy discretization of continuous features
`HingingHyperplaneTree`	class	Binary regression tree with hinging-hyperplane nodes identified by fuzzy c-regression
`HingeNode`	class	One node of a HingingHyperplaneTree
`simulate_free_run`	function	Free-run (parallel) simulation of a NARX model
`weighted_least_squares`	function	Weighted least-squares estimate of local model parameters
`select_features_by_separability`	function	Backward feature elimination by the Fisher interclass separability of the rules
`supervised_fuzzy_partition`	function	Per-feature fuzzy partitions derived from supervised fuzzy clustering
`ruspini_partition`	function	Ruspini-type (equidistant triangular) partition of every feature, the unsupervised baseline
`membership_1d`	function	Evaluate the fuzzy sets of one feature at the values x
`set_similarity`	function	Jaccard-type similarity $S = \|A \cap B\|/\|A \cup B\|$ of two fuzzy sets sampled on a common grid

Name	Kind	Description
plot_membership_functions	function	Gaussian antecedent membership functions $A_{ij}(x_j)$ of every rule, one panel per input
plot_local_models	function	One-input Takagi–Sugeno model: data, local linear models, model output and firing strengths
plot_fuzzy_partition	function	Fuzzy sets of one feature of a fitted FuzzyPartitionDiscretizer
plot_hinge_tree_surface	function	Prediction surface (filled contours) of a two-input hinging-hyperplane tree with its hinges
plot_decision_regions	function	Decision regions of a two-input classifier, optionally with the rule centers and antecedent ellipses
make_sinc	function	Samples of the sinc function, the classic static benchmark of fuzzy modeling
make_piecewise_linear	function	A continuous piecewise-linear function of two variables (sum of two hinges)
piecewise_linear_function	function	Noise-free value of the function behind make_piecewise_linear
make_switching_lines	function	Switching regression data: samples of several lines over a common input range
make_second_order_plant	function	The second-order nonlinear benchmark plant of Abonyi, Babuška and Szeifert (2002)
second_order_plant_function	function	Nonlinear component $f(y(k-1), y(k-2))$ of make_second_order_plant
make_mimo_plant	function	A two-input two-output nonlinear process with operating-point-dependent gains
make_two_class_overlap	function	Two overlapping classes, each a mixture of two Gaussian blobs (XOR layout)
make_banded_classes	function	Three classes separated by thresholds on two features, plus one irrelevant feature

A.8 `fuzzyclust.timeseries`: Time series

Name	Kind	Description
FuzzySegmentation	class	Fuzzy segmentation of a multivariate time series (modified Gath–Geva clustering)
ChaoticPredictor	class	Operating-regime model of a chaotic series in the reconstructed state space
MixedInputStructureSelection	class	Estimator form of select_structure_mixed
bottom_up_segmentation	function	Bottom-up segmentation with PCA models (Keogh et al. style, multivariate extension)
pca_segment_cost	function	Cost of approximating a segment by one PCA model with n_components components

Name	Kind	Description
`false_nearest_neighbors`	function	Classical false-nearest-neighbor percentages over the model orders (n_a, n_b)
`fnn_model_order`	function	Model order of a NARX model by false nearest neighbors with a cluster-based threshold
`select_model_order`	function	Convenience wrapper of fnn_model_order returning only the selected orders
`cluster_based_threshold`	function	Point-wise FNN thresholds R_k from Gath–Geva clusters in the regression space
`delay_embedding`	function	Delay vectors $\mathbf{y}_k = [y_k, y_{k-\tau}, \dots, y_{k-\tau(d_e-1)}]^T$ of a scalar series
`select_delay`	function	Lag τ of the delay embedding from the series itself
`embedding_scan`	function	Fit ChaoticPredictor for several embedding dimensions and tabulate the criteria
`select_embedding_dimension`	function	Read the embedding and the local dimension off an embedding_scan table
`correlation_dimension`	function	Grassberger–Procaccia correlation dimension of a point cloud (e.g. delay vectors)
`select_structure_mixed`	function	Rank all candidate input structures by their fuzzy variance (Girimonte, Babuška and Abonyi)
`fuzzy_variance`	function	Total fuzzy variance σ_t of y over the fuzzy segments spanned by several inputs
`plot_segmentation`	function	Time plots of the variables with the fuzzy segments underneath
`plot_fnn_table`	function	Heat map of the percentage of false nearest neighbors over the orders (n_a, n_b)
`plot_embedding_scan`	function	One-step prediction error and local-dimension index versus the embedding dimension
`plot_phase_space`	function	Two coordinates of a reconstructed (or true) state space, colored by cluster
`plot_structure_ranking`	function	Variance–complexity trade-off $VCT = \sigma_t n_i$ along the ranking of structures
`make_piecewise_timeseries`	function	Multivariate series with planted regime changes (segmentation benchmark)
`make_mackey_glass`	function	Mackey–Glass delay-differential series (chaotic for $\tau = 17$)
`make_lorenz`	function	Trajectory of the Lorenz system (RK4, fixed step)
`make_rossler`	function	Trajectory of the Rössler system with the parameters of Abonyi et al. (2004)
`make_nonlinear_io_system`	function	Continuous polymerization reactor benchmark of the FNN model-order paper
`make_mixed_input_system`	function	Regression problem with one discrete and several continuous inputs

A.9 `fuzzyclust.relational`: Relational, categorical, and mixed data

Name	Kind	Description
`RelationalFCM`	class	Relational fuzzy c-means (Hathaway, Davenport and Bezdek, 1989)
`NERFCM`	class	Non-Euclidean relational fuzzy c-means (Hathaway and Bezdek, 1994)
`FuzzyCMedoids`	class	Fuzzy c-medoids clustering of relational data (Krishnapuram et al., 2001)
`FuzzyKModes`	class	Fuzzy k-modes clustering of categorical data (Huang and Ng, 1999)
`FuzzyKPrototypes`	class	Fuzzy k-prototypes clustering of mixed numeric and categorical data (Huang, 1998)
`relational_distances`	function	Squared relational distances of the training objects to relational prototypes
`matching_dissimilarity`	function	Simple matching dissimilarity $\delta(\mathbf{a}, \mathbf{b}) = \sum_j [a_j \neq b_j]$ between rows
`pairwise_dissimilarity`	function	Pairwise dissimilarities of feature vectors by any SciPy metric
`check_dissimilarity_matrix`	function	Validate a (square or rectangular) dissimilarity matrix
`dtw_distance_matrix`	function	Dynamic time warping distances between all pairs of (equal-length) series
`dtw_alignment`	function	Optimal warping path between two series (for inspection and plotting)
`autocorrelation_features`	function	Sample autocorrelations $\hat{\rho}(1), \ldots, \hat{\rho}(L)$ of every series
`autocorrelation_distance_matrix`	function	Autocorrelation-based distance between time series (D'Urso and Maharaj, 2009)
`gower_distance_matrix`	function	Gower's general coefficient of dissimilarity for mixed data (Gower, 1971)
`plot_dissimilarity_matrix`	function	Heat map of a dissimilarity matrix with the objects ordered by their labels
`plot_dtw_alignment`	function	Two series with the dynamic-time-warping correspondences between them
`plot_modes`	function	Table of the fitted modes, shaded by how much membership mass the mode carries
`plot_series_clusters`	function	Time series drawn in the color of their cluster, medoid series emphasized
`make_categorical_blobs`	function	Categorical records scattered around planted modes
`make_mixed_blobs`	function	Records with Gaussian numeric attributes and categorical attributes around modes
`make_warped_series`	function	Time series of three shape classes with random time warps

A.10 `fuzzyclust.incomplete`: Incomplete data

Name	Kind	Description
`IncompleteFCM`	class	Fuzzy c-means clustering of incomplete data (Hathaway and Bezdek, 2001)
`STRATEGIES`	constant	Names of the missing-data strategies of `IncompleteFCM` (WDS, PDS, OCS, NPS)
`partial_distances`	function	Squared *partial distances* of (incomplete) samples to prototypes
`make_mcar`	function	Remove values *missing completely at random* (MCAR) from a data matrix
`make_incomplete_blobs`	function	Isotropic Gaussian clusters with values missing completely at random
`plot_incomplete_clusters`	function	Scatter plot of a clustering of incomplete 2-D data at the completed coordinates
`plot_imputation`	function	Imputed against true values of the missing entries, one color per feature

A.11 `fuzzyclust.large`: Very large data and streams

Name	Kind	Description
`WeightedFCM`	class	Weighted fuzzy c-means (wFCM): fuzzy c-means with a weight per sample
`RandomSampleFCM`	class	Random sample and extend FCM (rseFCM) for very large data
`SinglePassFCM`	class	Single-pass fuzzy c-means (spFCM) for very large and streaming data
`OnlineFCM`	class	Online fuzzy c-means (oFCM) for very large and streaming data
`extend_partition`	function	Memberships of (many) samples in fixed prototypes (the *extension* step)
`make_large_blobs`	function	Many samples from a few isotropic Gaussian clusters, in random order
`make_drifting_stream`	function	A stream of chunks whose cluster centers drift from chunk to chunk
`plot_chunk_timeline`	function	Prototype coordinate against the chunk index for a chunked model
`plot_center_trajectories`	function	Trajectories of the prototypes over the chunks on top of (a subsample of) the 2-D data

A.12 `fuzzyclust.image`: Image segmentation

Name	Kind	Description
`SpatialFCM`	class	Spatially constrained fuzzy c-means for image segmentation
`METHODS`	constant	Names of the spatial methods of `SpatialFCM`
`neighborhood_sum`	function	Sum of an image over the neighbors of every pixel, $\sum_{r \in N_k} y_r$ (center excluded)
`weighted_sum_image`	function	Linearly weighted sum image of EnFCM
`local_similarity_image`	function	Locally adaptive weighted image of FGFCM, $\xi_k = \sum_{r \in N_k} S_{kr} x_r / \sum_r S_{kr}$
`make_synthetic_image`	function	A piecewise-constant image of simple shapes with known labels, plus a noisy copy
`add_image_noise`	function	Corrupt an image with Gaussian, salt-and-pepper or both kinds of noise
`segmentation_accuracy`	function	Fraction of correctly labeled pixels after the best matching of the labels
`match_labels`	function	Best one-to-one mapping of predicted labels onto true labels (Hungarian algorithm)
`plot_segmentation`	function	Panels original / noisy / labels / membership maps of a fitted SpatialFCM
`plot_image`	function	Show a gray-level or RGB image without axes ticks
`plot_label_image`	function	Show a label image with the fixed cluster palette (cluster i in color i)
`plot_membership_maps`	function	One panel per cluster with its membership image in [0, 1]

A.13 `fuzzyclust.semisupervised`: Partial supervision

Name	Kind	Description
`SemiSupervisedFCM`	class	Fuzzy c-means with partial supervision (Pedrycz and Waletzky, 1997)
`PairwiseConstrainedFCM`	class	Fuzzy clustering with pairwise constraints, PCCA (Grira, Crucianu, and Boujemaa)
`check_pairwise_constraints`	function	Validate lists of pairwise constraints and return them as integer arrays (n_pairs, 2)
`constraint_satisfaction`	function	Fraction of satisfied must-link and cannot-link pairs under hard labels
`make_partially_labeled_pair`	function	Two long parallel groups whose *natural* split disagrees with the classes
`make_partial_labels`	function	Hide most labels of y: keep a random subset, set the rest to -1

Name	Kind	Description
`make_pairwise_constraints`	function	Draw random must-link and cannot-link pairs consistent with the labels y
`plot_label_propagation`	function	Labeled seeds as large black-edged markers, the rest colored by the cluster they were pulled into
`plot_pairwise_constraints`	function	Overlay must-link (solid green) and cannot-link (dashed vermilion) pairs on a scatter plot

A.14 `fuzzyclust.coclustering`: Co-clustering

Name	Kind	Description
`FCCM`	class	Fuzzy clustering for categorical multivariate data, FCCM (Oh, Honda, and Ichihashi, 2001)
`FCoDoK`	class	Fuzzy co-clustering of documents and keywords, Fuzzy CoDoK (Kummamuru, Dhawale, Krishnapuram, 2003)
`make_block_matrix`	function	A block-structured object–item count matrix with known co-clusters
`plot_coclusters`	function	Heat map of an object–item matrix with rows and columns reordered by co-cluster
`plot_item_memberships`	function	Bar chart of the item memberships of every co-cluster

A.15 `fuzzyclust.collaborative`: Collaborative clustering

Name	Kind	Description
`CollaborativeFCM`	class	Horizontal collaborative fuzzy clustering (Pedrycz, 2002)
`align_partitions`	function	Permute the clusters of U to match those of U_ref (maximal membership overlap)
`make_collaborative_sites`	function	The same objects described at several data sites by different features
`plot_site_partitions`	function	Scatter plot of every site's (two-dimensional) data colored by its own partition
`plot_collaboration_trajectory`	function	Agreement between the sites as a function of the collaboration weight α

A.16 `fuzzyclust.ensemble`: Cluster ensembles

Name	Kind	Description
`FuzzyClusterEnsemble`	class	Consensus of a library of soft clusterings (Punera and Ghosh, 2008)
`coassociation_matrix`	function	Soft co-association (similarity) matrix of a library of fuzzy partitions
`evidence_accumulation`	function	Evidence accumulation clustering of a co-association matrix (Fred and Jain, 2005)
`soft_mcla`	function	Soft meta-clustering algorithm, sMCLA (Punera and Ghosh, 2008)
`information_theoretic_kmeans`	function	Information-theoretic k-means consensus, ITK (Punera and Ghosh, 2008)
`cspa`	function	Cluster-based similarity partitioning, CSPA (Strehl and Ghosh, 2002), for hard partitions
`mcla`	function	Meta-clustering algorithm, MCLA (Strehl and Ghosh, 2002), for hard partitions
`stability_curve`	function	Reproducibility of the consensus partition as the library grows
`make_overlapping_blobs`	function	Gaussian groups that overlap enough for single clustering runs to disagree
`plot_coassociation`	function	Heat map of a co-association matrix with the samples reordered by (consensus) label
`plot_lifetimes`	function	Bar chart of the lifetime of every number of clusters in the co-association dendrogram

A.17 `fuzzyclust.datasets`: Data sets

Name	Kind	Description
`make_fuzzy_blobs`	function	Isotropic Gaussian clusters with random centers (MATLAB nDexample)
`make_ellipses`	function	Elongated 2-D Gaussian clusters with different orientations
`make_different_densities`	function	Clusters of very different size (density) and spread
`make_rings`	function	Concentric rings (hollow clusters) in 2-D
`make_lines`	function	Noisy line segments in 2-D (linear sub-space prototypes)
`make_noisy_blobs`	function	Gaussian clusters plus uniformly distributed outliers
`make_curved_manifold`	function	Points along two interleaved spiral arms (nonconvex clusters)
`make_two_moons`	function	Two interleaving half circles (thin wrapper around scikit-learn)

Name	Kind	Description
`load_motorcycle`	function	Motorcycle impact data (Silverman, 1985) used throughout the MATLAB toolbox
`load_iris`	function	Fisher's iris data (150 × 4, 3 classes) via scikit-learn
`load_wine`	function	Wine recognition data (178 × 13, 3 classes) via scikit-learn
`load_breast_cancer`	function	Wisconsin breast cancer data (569 × 30, 2 classes) via scikit-learn

A.18 `fuzzyclust.preprocessing`: Preprocessing

Name	Kind	Description
`normalize`	function	Scale the data and return the scaler for inverse transformation
`make_regressors`	function	Regression matrix and target of a NARX model
`lagged_matrix`	function	Delay-embedding of a univariate series

A.19 `fuzzyclust.plotting`: Plotting helpers

Name	Kind	Description
`PALETTE`	constant	Fixed categorical color palette; cluster i always gets `PALETTE[i]`
`plot_clusters`	function	Scatter plot of 2-D data colored by (hard) cluster label with prototypes
`plot_membership_contours`	function	Contour map of the maximal membership $\max_i \mu_i(\mathbf{x})$ over the plane
`plot_covariance_ellipses`	function	Draw the n_{std}-sigma ellipses of (fuzzy) covariance matrices
`plot_validity_scan`	function	Plot validity indices versus the number of clusters
`plot_objective`	function	Objective J versus iteration for one or several fitted estimators
`plot_membership_profiles`	function	Membership degrees μ_{ik} of every cluster along the sample index

APPENDIX

B



Migrating from the MATLAB toolbox

This appendix is for readers who know the *Fuzzy Clustering and Data Analysis Toolbox for MATLAB* [12]. It maps every MATLAB function, parameter, and result field to its counterpart in `fuzzyclust` and lists the places where the Python toolbox deliberately behaves differently. The chapter column of the first table points to the full treatment of each method.

B.1 Functions

The MATLAB toolbox passed a `data` structure and a `param` structure to every function and returned a nested `result` structure. In `fuzzyclust` every algorithm is a scikit-learn estimator: the parameters are constructor arguments, the data matrix is the argument of `fit`, and the results are attributes of the fitted object. Table B.1 maps the functions.

B.2 Parameters

Table B.2 maps the fields of the MATLAB `param` structure to the constructor arguments of the estimators.

B.3 Results

Table B.3 maps the fields of the MATLAB `result` structure to the attributes of a fitted estimator.

Table B.1. MATLAB toolbox functions and their `fuzzyclust` counterparts.

MATLAB	`fuzzyclust`	Chapter
`Kmeans(data,param)`	`cluster.KMeans(n_clusters).fit(X)`	3
`Kmedoid(data,param)`	`cluster.KMedoids(n_clusters).fit(X)`	3
`FCMclust(data,param)`	`cluster.FCM(n_clusters, m).fit(X)`	3
`GKclust(data,param)`	`cluster.GK(n_clusters, m, rho, gamma, beta).fit(X)`	3
`GGclust(data,param)`	`cluster.GG(n_clusters, m).fit(X)`	3
`clusteval(new,result, param)`	`est.predict_proba(X_new)`, `est.transform(X_new)`, `plotting.plot_membership_contours`	3
`validity(result,data, param)`	`validity.validity_indices(X, U, V, m)`	4
`optnumber` (demo)	`validity.scan_n_clusters`, `suggest_n_clusters`	4
`clust_normalize(data, 'range'|'var')`	`preprocessing.normalize(X, 'range'|'var')` or scikit-learn scalers	2
`clust_denormalize(data)`	`scaler.inverse_transform(Z)`	2
`PCA(data,param,result)`	`viz.pca_projection(X, n_components, centers)`	5
`Sammon(proj,data, result,param)`	`viz.Sammon(n_components).fit_transform(X)`	5
`FuzSam(proj,result, param)`	`viz.FuzzySammon().fit(est, X)`	5
`projeval(result,param)`	`viz.projection_quality(U, Y, Z, m)`	5
`samstr(data,result)` (stress)	`viz.sammon_stress(X, Y)`	5
`nDexample(cx,N,n,seed)`	`datasets.make_fuzzy_blobs(...)`	2
`MotorCycle.txt`	`datasets.load_motorcycle()`	3

B.4 Behavioral differences

Initialization.
The fuzzy algorithms of the MATLAB toolbox drew random prototypes uniformly from the box centered at the data mean whose half-widths are the largest absolute deviations from it, with a fixed seed (`rand('state',0)`); `Kmeans` and `Kmedoid` started from randomly chosen samples. `fuzzyclust` offers the same start (`init="random"`, the default) and k-means++ seeding for every estimator, controls the seed through `random_state`, and can run several restarts (`n_init`), keeping the partition with the lowest J (section 2.2.6).

Termination.
MATLAB iterated while the largest membership *increase*, $\max_{ik}(\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)})$ without the absolute value, exceeded ϵ, and had no iteration cap; `fuzzyclust` tests $\max_{ik} |\Delta\mu_{ik}| < \epsilon$ and adds `max_iter`.

Optional parameters.
`FCMclust`, `GKclust`, and `GGclust` tested their optional fields with

Table B.2. Fields of the MATLAB `param` structure and the corresponding constructor arguments.

`param.*`	constructor argument	remark
`param.c` (scalar)	`n_clusters`	number of clusters
`param.c` (matrix $N \times c$)	`init=U0`	initial partition matrix
`param.c` (matrix $c \times n$, k-means)	`init=V0`	initial centers
`param.m`	`m`	fuzziness exponent (default 2)
`param.e`	`tol`	termination tolerance on $\max \lvert\Delta\mu_{ik}\rvert$ (default 10^{-4})
`param.ro`	`rho`	GK cluster volumes (default all 1); MATLAB's $(\det \mathbf{F}_i/\rho_i)^{1/n}\mathbf{F}_i^{-1}$ gave $\det \mathbf{A}_i = 1/\rho_i$, `fuzzyclust` follows $\det \mathbf{A}_i = \rho_i$
`param.gamma`	`gamma`	covariance regularization weight of GK, here also of GG (default 0)
`param.beta`	`beta`	condition-number bound of GK, here also of GG (default 10^{15})
`param.val` (1/2/3)	`indices=[...]` of `validity_indices`	1: PC, CE; 2: SC, S, XB; 3: DI, ADI
`param.vis`	`plotting.*` after fitting	plotting is separated from fitting
`param.q`	`n_components`	projection dimension (PCA; Sammon and FuzSam took it from `proj.P`)
`param.max`	`max_iter`	iteration count of Sammon and FuzSam (default 100)
`param.alpha`	`alpha`	gradient step size of Sammon and FuzSam
—	`init`, `n_init`, `max_iter`, `random_state`	new in the clustering estimators: initialization strategy, restarts, iteration cap, seed

`exist('param.m')==1`. `exist` looks for a variable or a file of that name, never for a structure field, so the test was always false: `m` and `e` (and, in `GKclust`, `ro`, `gamma`, and `beta`) always took their default values, and `clusteval` and `validity` always used $m = 2$ for the same reason. The tolerance line also read `param.m` where `param.e` was meant. Results computed in MATLAB with other settings therefore correspond to the defaults; `fuzzyclust` honors every constructor argument.

GK covariance regularization.
The MATLAB code shrank the covariance toward `A0.^(1/n)`, and since `A0` already contained $\det(\mathbf{F}_0)^{1/n}$, it computed $\mathbf{F}_i \leftarrow (1-\gamma)\mathbf{F}_i + \gamma \det(\mathbf{F}_0)^{1/n^2}\mathbf{I}$; the Python implementation follows the published formula $\mathbf{F}_i \leftarrow (1-\gamma)\mathbf{F}_i + \gamma \det(\mathbf{F}_0)^{1/n}\mathbf{I}$ of Babuška et al. [19] exactly (section 3.2.4). Since `gamma` was always zero in MATLAB (see above), the difference never showed in MATLAB results.

Table B.3. Fields of the MATLAB `result` structure and the fitted attributes.

`result.*`	attribute	shape
`result.data.f`	`membership_`	(N, c) — same orientation as MATLAB
`result.data.d`	`distances_`	(N, c), D_{ik} (not squared)
`result.cluster.v`	`cluster_centers_`	(c, n)
`result.cluster.P`	`covariances_`	(c, n, n) (MATLAB: $n \times n \times c$)
`result.cluster.M`	`norm_matrices_`	(c, n, n), GK only
`result.cluster.V`	`eigenvectors_`	(c, n, n), all eigenvectors as columns, in ascending order of the eigenvalues (MATLAB kept only that of the smallest eigenvalue)
`result.cluster.D`	`eigenvalues_`	(c, n), ascending
`result.cluster.Pi` (GG)	`priors_`	$(c,)$
`result.iter`	`n_iter_`	
`result.cost`	`objective_history_`, `objective_`	J per iteration, final J; MATLAB's `cost` summed $\mu_{ik} D_{ik}^2$ without the exponent m (`GGclust`: after the distances were transformed), so its values differ
`result.validity.PC` …	`validity_indices(...)['PC']`	dictionary
`eval.f`, `eval.d` (`clusteval`)	`predict_proba(X)`, `transform(X)`	(N_{new}, c)
`result.proj.P`	`embedding_`; first output `Y` of `pca_projection`	(N, q)
`result.proj.vp`	`embedded_centers_` (FuzzySammon); second output `Z` of `pca_projection`	(c, q)
`result.proj.e`	`stress_` (Sammon) or `sammon_stress(X, Y)`	float; the function scores any projection

Gath–Geva.
The exponential distance is evaluated in the logarithmic domain, so far-away samples no longer overflow. The covariance weighting exponent equals m, as in MATLAB, but the priors are computed from the memberships themselves, $\alpha_i = \frac{1}{N} \sum_k \mu_{ik}$, where MATLAB used μ_{ik}^m. The regularization of GK (`gamma`, `beta`) is also available.

Validity.
All indices are computed at once from the data, the partition, the centers, and m, with Euclidean distances whatever the clustering algorithm (for SC, S, XB, and ADI, MATLAB used the algorithm's distance matrix `result.data.d`). The Fukuyama–Sugeno, PBM-F, Kwon, and PCAES indices, the volume and density indices of Gath and Geva (FHV, PD, APD), MPC, and the silhouette were added. The `param.val` groups are reproduced by passing the index names.

Plotting.
`param.vis` and the contour map of `clusteval` are replaced by explicit plotting helpers that draw on a matplotlib `Axes`; fitting never opens a figure.

Data.
The data matrix is a NumPy array or a pandas `DataFrame`; no wrapper structure is needed. Feature names are kept in `feature_names_in_`.

Normalization.
`clust_normalize` stored the scaling inside the data structure; here the scaler is a separate object (or a step in a scikit-learn `Pipeline`) whose `inverse_transform` plays the role of `clust_denormalize`. The MATLAB `Kmeans` range-normalized its input internally; `KMeans` does not scale the data.

A minimal translation of a typical MATLAB session,

```
data = clust_normalize(data,'range');
param.c = 3; param.m = 2; param.e = 1e-4;
result = GKclust(data,param);
param.val = 2;
result = validity(result,data,param);
new.X = (new.X - repmat(data.min,size(new.X,1),1)) ./ ...
        repmat(data.max - data.min,size(new.X,1),1);
eval = clusteval(new,result,param);
```

reads in Python as

```python
from fuzzyclust.cluster import GK
from fuzzyclust.preprocessing import normalize
from fuzzyclust.validity import validity_indices

Xs, scaler = normalize(X, "range")
gk = GK(n_clusters=3, m=2.0, tol=1e-4, random_state=0).fit(Xs)
indices = validity_indices(Xs, gk.membership_, gk.cluster_centers_, m=gk.m,
                           indices=["SC", "S", "XB"])
U_new = gk.predict_proba(scaler.transform(X_new))
```

APPENDIX

C



The example gallery

Every algorithm of the toolbox has at least one runnable example script. The scripts generate their own data, explain the method in their comments, and end with a study of the main parameters. The same scripts produce the gallery of the online documentation, the Jupyter notebooks, and every computed figure and code listing of this book. The tables below list them by folder of `examples/`, with the first figure of the book that each script produced.

C.1 Core partitioning algorithms and validity

Folder `examples/01_clustering/`. The methods are described in chapter 3.

Script	Topic	Figure
`plot_01_fcm_basics.py`	Fuzzy c-means: memberships and the fuzziness exponent	1.3
`plot_02_hard_vs_fuzzy.py`	Hard versus fuzzy partitions: k-means, k-medoids, and FCM	3.7
`plot_03_gk_ellipses.py`	Gustafson–Kessel: clusters with their own shape	3.8
`plot_04_gg_densities.py`	Gath–Geva: clusters of different size and density	3.12
`plot_05_validity_scan.py`	Choosing the number of clusters with validity indices	4.3
`plot_06_predict_new_data.py`	Applying a fitted model to new data	3.14
`plot_07_motorcycle_scaling.py`	Feature scaling on real data: the motorcycle data set	2.6

Script	Topic	Figure
`plot_08_sklearn_pandas_workflow.py`	Working with pandas and scikit-learn: pipelines, DataFrames, and model selection	2.8
`plot_09_fuzziness_exponent.py`	The fuzziness exponent m: from crisp cells to uniform memberships	3.5
`plot_10_initialisation_restarts.py`	Local optima and restarts: initializing Gustafson–Kessel on crossing lines	2.7
`plot_11_distance_norms.py`	Distance norms: Euclidean, diagonal, and Mahalanobis fuzzy c-means	2.5
`plot_12_fuzzifiers_annealing.py`	What makes a partition fuzzy: fuzzifiers, regularization, and annealing	3.16
`plot_13_subtractive_clustering.py`	Subtractive clustering: the number and location of clusters from density	3.20
`plot_14_merging_stability.py`	Three routes to the number of clusters: merging, validity scan, and stability	4.4

C.2 Visualization of fuzzy clusters

Folder `examples/02_visualization/`. The methods are described in chapter 5.

Script	Topic	Figure
`plot_01_pca_vs_sammon.py`	PCA versus Sammon mapping: preserving variance or distances	5.3
`plot_02_fuzzy_sammon.py`	Fuzzy Sammon mapping: projecting clusters, not just data	5.5
`plot_03_fuzzy_som.py`	Fuzzy self-organizing map: ordered prototypes by regularized FCM	5.8
`plot_04_cluster_number_validation.py`	Visual validation of the number of clusters with FUZZSAM	5.14

C.3 Model identification

Folder `examples/03_models/`. The methods are described in chapter 8.

Script	Topic	Figure
`plot_01_ts_sinc.py`	Takagi–Sugeno model of a static nonlinearity: clusters become rules	8.5
`plot_02_ts_dynamic_plant.py`	Identification of a nonlinear dynamic system with free-run validation	8.9

Script	Topic	Figure
`plot_03_ts_mimo.py`	MIMO Takagi–Sugeno model with local MIMO ARX consequents	8.14
`plot_04_fcrm_switching.py`	Fuzzy c-regression models: switching regression with crossing lines	8.17
`plot_05_supervised_classifier.py`	Supervised fuzzy clustering: classifier rules that carry class probabilities	8.21
`plot_06_fuzzy_partition_tree.py`	Supervised fuzzy partitions as a priori discretization for decision trees	8.26
`plot_07_hinging_tree.py`	Hinging-hyperplane regression tree identified by fuzzy c-regression	8.29

C.4 Time-series analysis

Folder `examples/04_timeseries/`. The methods are described in chapter 9.

Script	Topic	Figure
`plot_01_fuzzy_segmentation.py`	Fuzzy segmentation of a multivariate time series	9.4
`plot_02_bottom_up_vs_fuzzy.py`	Bottom-up segmentation with PCA models versus fuzzy segmentation	9.9
`plot_03_fnn_model_order.py`	Model-order selection by false nearest neighbors with a cluster-based threshold	9.11
`plot_04_chaotic_reconstruction.py`	State-space reconstruction and prediction of a chaotic time series	9.16
`plot_05_mixed_input_structure.py`	Model-free structure selection with mixed discrete and continuous inputs	9.21

C.5 Graph-based clustering

Folder `examples/05_graph/`. The methods are described in chapter 7.

Script	Topic	Figure
`plot_01_mst_clustering.py`	Minimal spanning tree clustering: cutting, splitting, and Gath–Geva refinement	7.3
`plot_02_mst_cut_criteria.py`	Inconsistent edges: Zahn's criteria and the neighborhood depth	7.7
`plot_03_mst_validation.py`	Number of clusters: volume-based validity and fuzzy cluster similarity	7.10

Script	Topic	Figure
`plot_04_geodesic_fcm.py`	Geodesic-distance-based fuzzy clustering versus fuzzy c-means	7.13
`plot_05_geodesic_path_isomap.py`	Geodesic paths, the S-curve, and clustering of the Isomap projection	7.17

C.6 Possibilistic, noise, kernel, and shape-prototype extensions

Folder `examples/06_extensions/`. The methods are described in chapter 6.

Script	Topic	Figure
`plot_01_pcm_typicality.py`	Possibilistic c-means: typicalities, scales and coincident clusters	6.4
`plot_02_pfcm_trade_off.py`	Possibilistic fuzzy c-means: memberships and typicalities together	6.8
`plot_03_noise_clustering.py`	Noise clustering: a virtual cluster at constant distance δ	6.11
`plot_04_kernel_fcm.py`	Kernel fuzzy c-means: rings and moons through the kernel trick	6.15
`plot_05_fuzzy_c_varieties.py`	Fuzzy c-varieties: lines as prototypes, and the elliptotype compromise	6.19
`plot_06_fuzzy_c_shells.py`	Fuzzy c-shells: circles as prototypes	6.22
`plot_07_feature_weighting.py`	Learning which features matter: SCAD and the weighted FCM	6.30
`plot_08_volume_prototypes.py`	Volume prototypes and adaptive cluster merging	6.26

C.7 Relational, categorical, and mixed data

Folder `examples/07_relational/`. The methods are described in chapter 10.

Script	Topic	Figure
`plot_01_rfcm_fcm_duality.py`	Relational fuzzy c-means and NERF for non-Euclidean data	10.4
`plot_02_fcmdd_dtw_series.py`	Fuzzy c-medoids on dynamic-time-warping dissimilarities of time series	10.7
`plot_03_kmodes_survey.py`	Fuzzy k-modes on categorical survey data	10.12

Script	Topic	Figure
`plot_04_kprototypes_mixed.py`	Fuzzy k-prototypes on mixed numeric and categorical data	10.16
`plot_05_gower_fcmdd_vs_onehot.py`	Gower's dissimilarity versus one-hot encoding for mixed data	10.19
`plot_06_autocorrelation_series.py`	Autocorrelation-based fuzzy clustering of time series	10.22

C.8 Incomplete, very large, and image data

Folder `examples/08_data_realities/`. The methods are described in chapter 11.

Script	Topic	Figure
`plot_01_missing_strategies.py`	Clustering incomplete data: the four strategies of Hathaway and Bezdek	11.4
`plot_02_ocs_imputation.py`	Imputation by clustering: accuracy of the completed values	11.7
`plot_03_large_data.py`	Fuzzy c-means for very large data: rseFCM, spFCM and oFCM	11.10
`plot_04_stream_drift.py`	Clustering a drifting stream chunk by chunk with `partial_fit`	11.13
`plot_05_image_noise.py`	Image segmentation under noise: six spatially constrained FCM variants	11.16
`plot_06_spatial_alpha_window.py`	Spatial FCM: the neighborhood weight and the window size	11.19

C.9 Partial supervision, co-clustering, collaboration, and ensembles

Folder `examples/09_beyond_unsupervised/`. The methods are described in chapter 12.

Script	Topic	Figure
`plot_01_ssfcm_partial_labels.py`	Fuzzy c-means with partial supervision: label fraction and weight	12.5
`plot_02_pcca_pairwise_constraints.py`	Pairwise constraints: must-link, cannot-link and competitive agglomeration	12.8

Script	Topic	Figure
`plot_03_fccm_document_term.py`	Fuzzy co-clustering of a document–term matrix	12.11
`plot_04_collaborative_sites.py`	Collaborative fuzzy clustering across three data sites	12.15
`plot_05_ensemble_consensus.py`	Consensus of soft clusterings: a library beats a single run	12.17
`plot_06_ensemble_lifetime.py`	The lifetime criterion of evidence accumulation	12.20

Bibliography

[1] Enrique H. Ruspini. “A new approach to clustering”. In: *Information and Control* 15.1 (1969), pp. 22–32. DOI: 10.1016/S0019-9958(69)90591-9.

[2] Joseph C. Dunn. “A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters”. In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57. DOI: 10.1080/01969727308546046.

[3] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum Press, 1981. DOI: 10.1007/978-1-4757-0450-1.

[4] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[5] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: Wiley, 1990. DOI: 10.1002/9780470316801.

[6] Frank Höppner, Frank Klawonn, Rudolf Kruse, and Thomas Runkler. *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*. Chichester: John Wiley & Sons, 1999.

[7] Robert Babuška. *Fuzzy Modeling for Control*. Vol. 12. International Series in Intelligent Technologies. Boston: Kluwer Academic Publishers, 1998. DOI: 10.1007/978-94-011-4868-9.

[8] János Abonyi and Balázs Feil. *Cluster Analysis for Data Mining and System Identification*. Basel: Birkhäuser, 2007. DOI: 10.1007/978-3-7643-7988-9.

[9] Lotfi A. Zadeh. “Fuzzy sets”. In: *Information and Control* 8.3 (1965), pp. 338–353. DOI: 10.1016/S0019-9958(65)90241-X.

[10] James C. Bezdek. “A convergence theorem for the fuzzy ISODATA clustering algorithms”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2.1 (1980), pp. 1–8. DOI: 10.1109/TPAMI.1980.4766964.

[11] James C. Bezdek, Richard J. Hathaway, Michael J. Sabin, and William T. Tucker. “Convergence theory for fuzzy c-means: counterexamples and repairs”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 17.5 (1987), pp. 873–877. DOI: 10.1109/TSMC.1987.6499296.

[12] Balázs Balaskó, János Abonyi, and Balázs Feil. *Fuzzy Clustering and Data Analysis Toolbox for Use with MATLAB*. Toolbox manual. Veszprém, Hungary: Department of Process Engineering, University of Veszprém, 2005.

[13] Fabian Pedregosa et al. “Scikit-learn: machine learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[14] Raghu Krishnapuram and James M. Keller. "A possibilistic approach to clustering". In: *IEEE Transactions on Fuzzy Systems* 1.2 (1993), pp. 98–110. DOI: 10.1109/91.227387.

[15] David Arthur and Sergei Vassilvitskii. "k-means++: the advantages of careful seeding". In: *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. New Orleans: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[16] James C. Bezdek, Robert Ehrlich, and William Full. "FCM: the fuzzy c-means clustering algorithm". In: *Computers & Geosciences* 10.2–3 (1984), pp. 191–203. DOI: 10.1016/0098-3004(84)90020-7.

[17] James MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*. Ed. by Lucien M. Le Cam and Jerzy Neyman. Vol. 1. Berkeley: University of California Press, 1967, pp. 281–297.

[18] Donald E. Gustafson and William C. Kessel. "Fuzzy clustering with a fuzzy covariance matrix". In: *Proceedings of the 1978 IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes*. San Diego, CA, 1979, pp. 761–766. DOI: 10.1109/CDC.1978.268028.

[19] Robert Babuška, Pieter J. van der Veen, and Uzay Kaymak. "Improved covariance estimation for Gustafson–Kessel clustering". In: *Proceedings of the 2002 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2002)*. Vol. 2. IEEE, 2002, pp. 1081–1085. DOI: 10.1109/FUZZ.2002.1006654.

[20] James C. Bezdek and Joseph C. Dunn. "Optimal fuzzy partitions: a heuristic for estimating the parameters in a mixture of normal distributions". In: *IEEE Transactions on Computers* 24.8 (1975), pp. 835–838. DOI: 10.1109/T-C.1975.224317.

[21] Isak Gath and Amir B. Geva. "Unsupervised optimal fuzzy clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.7 (1989), pp. 773–780. DOI: 10.1109/34.192473.

[22] Frank Klawonn and Frank Höppner. "What is fuzzy about fuzzy clustering? Understanding and improving the concept of the fuzzifier". In: *Advances in Intelligent Data Analysis V: 5th International Symposium on Intelligent Data Analysis, IDA 2003*. Vol. 2810. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 254–264. DOI: 10.1007/978-3-540-45231-7_24.

[23] Sadaaki Miyamoto and Masao Mukaidono. "Fuzzy c-means as a regularization and maximum entropy approach". In: *Proceedings of the 7th International Fuzzy Systems Association World Congress (IFSA'97)*. Vol. 2. Prague, 1997, pp. 86–92.

[24] Rui-Ping Li and Masao Mukaidono. "A maximum-entropy approach to fuzzy clustering". In: *Proceedings of the 4th IEEE International Conference on Fuzzy Systems and the 2nd International Fuzzy Engineering Symposium (FUZZ-IEEE/IFES 1995)*. Vol. 4. Yokohama: IEEE, 1995, pp. 2227–2232. DOI: 10.1109/FUZZY.1995.409989.

[25] Kenneth Rose. "Deterministic annealing for clustering, compression, classification, regression, and related optimization problems". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2210–2239. DOI: 10.1109/5.726788.

[26] Sadaaki Miyamoto and Kazutaka Umayahara. "Fuzzy clustering by quadratic regularization". In: *Proceedings of the 1998 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 1998), IEEE World Congress on Computational Intelligence*. Vol. 2. Anchorage: IEEE, 1998, pp. 1394–1399. DOI: 10.1109/FUZZY.1998.686323.

[27] Ronald R. Yager and Dimitar P. Filev. "Approximate clustering via the mountain method". In: *IEEE Transactions on Systems, Man, and Cybernetics* 24.8 (1994), pp. 1279–1284. DOI: 10.1109/21.299710.

[28] Stephen L. Chiu. "Fuzzy model identification based on cluster estimation". In: *Journal of Intelligent & Fuzzy Systems* 2.3 (1994), pp. 267–278. DOI: 10.3233/IFS-1994-2306.

[29] Rajesh N. Davé. "Validating fuzzy partitions obtained through c-shells clustering". In: *Pattern Recognition Letters* 17.6 (1996), pp. 613–623. DOI: 10.1016/0167-8655(96)00026-8.

[30] Amine M. Bensaid et al. "Validity-guided (re)clustering with applications to image segmentation". In: *IEEE Transactions on Fuzzy Systems* 4.2 (1996), pp. 112–123. DOI: 10.1109/91.493905.

[31] Xuanli Lisa Xie and Gerardo Beni. "A validity measure for fuzzy clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.8 (1991), pp. 841–847. DOI: 10.1109/34.85677.

[32] Nikhil R. Pal and James C. Bezdek. "On cluster validity for the fuzzy c-means model". In: *IEEE Transactions on Fuzzy Systems* 3.3 (1995), pp. 370–379. DOI: 10.1109/91.413225.

[33] Yoshiki Fukuyama and Michio Sugeno. "A new method of choosing the number of clusters for the fuzzy c-means method". In: *Proceedings of the 5th Fuzzy Systems Symposium*. 1989, pp. 247–250.

[34] Malay K. Pakhira, Sanghamitra Bandyopadhyay, and Ujjwal Maulik. "Validity index for crisp and fuzzy clusters". In: *Pattern Recognition* 37.3 (2004), pp. 487–501. DOI: 10.1016/j.patcog.2003.06.005.

[35] Joseph C. Dunn. "Well-separated clusters and optimal fuzzy partitions". In: *Journal of Cybernetics* 4.1 (1974), pp. 95–104. DOI: 10.1080/01969727408546059.

[36] Peter J. Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. DOI: 10.1016/0377-0427(87)90125-7.

[37] Soon H. Kwon. "Cluster validity index for fuzzy clustering". In: *Electronics Letters* 34.22 (1998), pp. 2176–2177. DOI: 10.1049/el:19981523.

[38] Kuo-Lung Wu and Miin-Shen Yang. "A cluster validity index for fuzzy clustering". In: *Pattern Recognition Letters* 26.9 (2005), pp. 1275–1291. DOI: 10.1016/j.patrec.2004.11.022.

[39] Raghu Krishnapuram and Chih-Pin Freg. "Fitting an unknown number of lines and planes to image data through compatible cluster merging". In: *Pattern Recognition* 25.4 (1992), pp. 385–400. DOI: 10.1016/0031-3203(92)90087-Y.

[40] Uzay Kaymak and Robert Babuška. "Compatible cluster merging for fuzzy modelling". In: *Proceedings of the 4th IEEE International Conference on Fuzzy Systems and the 2nd International Fuzzy Engineering Symposium (FUZZ-IEEE/IFES 1995)*. Vol. 2. IEEE, 1995, pp. 897–904. DOI: 10.1109/FUZZY.1995.409789.

[41] P. M. Kelly. *An algorithm for merging hyperellipsoidal clusters*. Tech. rep. LA-UR-94-3306. Los Alamos National Laboratory, 1994.

[42] Magne Setnes, Robert Babuška, Uzay Kaymak, and Hans R. van Nauta Lemke. "Similarity measures in fuzzy rule base simplification". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28.3 (1998), pp. 376–386. DOI: 10.1109/3477.678632.

[43] Uzay Kaymak and Magne Setnes. "Fuzzy clustering with volume prototypes and adaptive cluster merging". In: *IEEE Transactions on Fuzzy Systems* 10.6 (2002), pp. 705–712. DOI: 10.1109/TFUZZ.2002.805901.

[44] Lawrence Hubert and Phipps Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (1985), pp. 193–218. DOI: 10.1007/BF01908075.

[45] Eyke Hüllermeier, Maria Rifqi, Sascha Henzgen, and Robin Senge. "Comparing fuzzy partitions: a generalization of the Rand index and related measures". In: *IEEE Transactions on Fuzzy Systems* 20.3 (2012), pp. 546–556. DOI: 10.1109/TFUZZ.2011.2179303.

[46] Ricardo J. G. B. Campello. "A fuzzy extension of the Rand index and other related indexes for clustering and classification assessment". In: *Pattern Recognition Letters* 28.7 (2007), pp. 833–841. DOI: 10.1016/j.patrec.2006.11.010.

[47] Asa Ben-Hur, André Elisseeff, and Isabelle Guyon. "A stability based method for discovering structure in clustered data". In: *Biocomputing 2002: Proceedings of the Pacific Symposium*. Singapore: World Scientific, 2002, pp. 6–17. DOI: 10.1142/9789812799623_0002.

[48] Tilman Lange, Volker Roth, Mikio L. Braun, and Joachim M. Buhmann. "Stability-based validation of clustering solutions". In: *Neural Computation* 16.6 (2004), pp. 1299–1323. DOI: 10.1162/089976604773717621.

[49] Annamária Kovács and János Abonyi. "Visualization of fuzzy clustering results by modified Sammon mapping". In: *Proceedings of the 3rd International Symposium of Hungarian Researchers on Computational Intelligence*. Budapest, 2002, pp. 177–188.

[50] János Abonyi and Robert Babuška. "FUZZSAM – visualization of fuzzy clustering results by modified Sammon mapping". In: *Proceedings of the 2004 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2004)*. Vol. 1. Budapest: IEEE, 2004, pp. 365–370. DOI: 10.1109/FUZZY.2004.1375750.

[51] Balázs Feil, Balázs Balaskó, and János Abonyi. "Visualization of fuzzy clusters by fuzzy Sammon mapping projection: application to the analysis of phase space trajectories". In: *Soft Computing* 11.5 (2007), pp. 479–488. DOI: 10.1007/s00500-006-0111-5.

[52] John W. Sammon. "A nonlinear mapping for data structure analysis". In: *IEEE Transactions on Computers* 18.5 (1969), pp. 401–409. DOI: 10.1109/T-C.1969.222678.

[53] Ian T. Jolliffe. *Principal Component Analysis*. 2nd ed. Springer Series in Statistics. New York: Springer, 2002. DOI: 10.1007/b98835.

[54] Jianchang Mao and Anil K. Jain. "Artificial neural networks for feature extraction and multivariate data projection". In: *IEEE Transactions on Neural Networks* 6.2 (1995), pp. 296–317. DOI: 10.1109/72.363467.

[55] Dick de Ridder and Robert P. W. Duin. "Sammon's mapping using neural networks: a comparison". In: *Pattern Recognition Letters* 18.11–13 (1997), pp. 1307–1316. DOI: 10.1016/S0167-8655(97)00093-7.

[56] Nikhil R. Pal and Vijay Kumar Eluri. "Two efficient connectionist schemes for structure preserving dimensionality reduction". In: *IEEE Transactions on Neural Networks* 9.6 (1998), pp. 1142–1154. DOI: 10.1109/72.728358.

[57] Teuvo Kohonen. "The self-organizing map". In: *Proceedings of the IEEE* 78.9 (1990), pp. 1464–1480. DOI: 10.1109/5.58325.

[58] Eric Chen-Kuo Tsao, James C. Bezdek, and Nikhil R. Pal. "Fuzzy Kohonen clustering networks". In: *Pattern Recognition* 27.5 (1994), pp. 757–764. DOI: 10.1016/0031-3203(94)90052-3.

[59] Roberto D. Pascual-Marqui, Alberto D. Pascual-Montano, Kieko Kochi, and José M. Carazo. "Smoothly distributed fuzzy c-means: a new self-organizing map". In: *Pattern Recognition* 34.12 (2001), pp. 2395–2402. DOI: 10.1016/S0031-3203(00)00167-9.

[60] János Abonyi, Sándor Migály, and Ferenc Szeifert. "Fuzzy self-organizing map based on regularized fuzzy c-means clustering". In: *Advances in Soft Computing: Engineering Design and Manufacturing*. Ed. by José Manuel Benítez, Oscar Cordón, Frank Hoffmann, and Rajkumar Roy. London: Springer, 2003, pp. 99–108. DOI: 10.1007/978-1-4471-3744-3_10.

[61] Hong Yan. "Fuzzy curve-tracing algorithm". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 31.5 (2001), pp. 768–780. DOI: 10.1109/3477.956038.

[62] Juha Vesanto. "Neural network tool for data mining: SOM Toolbox". In: *Proceedings of the 5th International Symposium on Tool Environments and Development Methods for Intelligent Systems (TOOLMET 2000)*. Oulu, Finland: Oulun yliopistopaino, 2000, pp. 184–196.

[63] Nikhil R. Pal, Kuhu Pal, James M. Keller, and James C. Bezdek. "A possibilistic fuzzy c-means clustering algorithm". In: *IEEE Transactions on Fuzzy Systems* 13.4 (2005), pp. 517–530. DOI: 10.1109/TFUZZ.2004.840099.

[64] Rajesh N. Davé. "Characterization and detection of noise in clustering". In: *Pattern Recognition Letters* 12.11 (1991), pp. 657–664. DOI: 10.1016/0167-8655(91)90002-4.

[65] Raghu Krishnapuram and James M. Keller. "The possibilistic c-means algorithm: insights and recommendations". In: *IEEE Transactions on Fuzzy Systems* 4.3 (1996), pp. 385–393. DOI: 10.1109/91.531779.

[66] Rajesh N. Davé and Raghu Krishnapuram. "Robust clustering methods: a unified view". In: *IEEE Transactions on Fuzzy Systems* 5.2 (1997), pp. 270–293. DOI: 10.1109/91.580801.

[67] Peter J. Huber. *Robust Statistics*. New York: Wiley, 1981. DOI: 10.1002/0471725250.

[68] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, 2002. DOI: 10.7551/mitpress/4175.001.0001.

[69] Daniel Graves and Witold Pedrycz. "Kernel-based fuzzy clustering and fuzzy clustering: a comparative experimental study". In: *Fuzzy Sets and Systems* 161.4 (2010), pp. 522–543. DOI: 10.1016/j.fss.2009.10.021.

[70] Mark Girolami. "Mercer kernel-based clustering in feature space". In: *IEEE Transactions on Neural Networks* 13.3 (2002), pp. 780–784. DOI: 10.1109/TNN.2002.1000150.

[71] Dao-Qiang Zhang and Song-Can Chen. "A novel kernelized fuzzy c-means algorithm with application in medical image segmentation". In: *Artificial Intelligence in Medicine* 32.1 (2004), pp. 37–50. DOI: 10.1016/j.artmed.2004.01.012.

[72] James C. Bezdek, Chris Coray, Robert Gunderson, and James Watson. "Detection and characterization of cluster substructure. I. Linear structure: fuzzy c-lines". In: *SIAM Journal on Applied Mathematics* 40.2 (1981), pp. 339–357. DOI: 10.1137/0140029.

[73] James C. Bezdek, Chris Coray, Robert Gunderson, and James Watson. "Detection and characterization of cluster substructure. II. Fuzzy c-varieties and convex combinations thereof". In: *SIAM Journal on Applied Mathematics* 40.2 (1981), pp. 358–372. DOI: 10.1137/0140030.

[74] Rajesh N. Davé. "Fuzzy shell-clustering and applications to circle detection in digital images". In: *International Journal of General Systems* 16.4 (1990), pp. 343–355. DOI: 10.1080/03081079008935087.

[75] Raghu Krishnapuram, Olfa Nasraoui, and Hichem Frigui. "The fuzzy c spherical shells algorithm: a new approach". In: *IEEE Transactions on Neural Networks* 3.5 (1992), pp. 663–671. DOI: 10.1109/72.159056.

[76] Joshua Zhexue Huang, Michael K. Ng, Hongqiang Rong, and Zichen Li. "Automated variable weighting in k-means type clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.5 (2005), pp. 657–668. DOI: 10.1109/TPAMI.2005.95.

[77] Annette Keller and Frank Klawonn. "Fuzzy clustering with weighting of data variables". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 8.6 (2000), pp. 735–746. DOI: 10.1142/S0218488500000538.

[78] Hichem Frigui and Olfa Nasraoui. "Unsupervised learning of prototypes and attribute weights". In: *Pattern Recognition* 37.3 (2004), pp. 567–581. DOI: 10.1016/j.patcog.2003.08.002.

[79] Dao-Qiang Zhang and Song-Can Chen. "Clustering incomplete data using kernel-based fuzzy c-means algorithm". In: *Neural Processing Letters* 18.3 (2003), pp. 155–162. DOI: 10.1023/B:NEPL.0000011135.19145.1b.

[80] Raghu Krishnapuram, Hichem Frigui, and Olfa Nasraoui. "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation—Part I". In: *IEEE Transactions on Fuzzy Systems* 3.1 (1995), pp. 29–43. DOI: 10.1109/91.366564.

[81] Charles T. Zahn. “Graph-theoretical methods for detecting and describing gestalt clusters”. In: *IEEE Transactions on Computers* C-20.1 (1971), pp. 68–86. DOI: 10.1109/T-C.1971.223083.

[82] John C. Gower and Gavin J. S. Ross. “Minimum spanning trees and single linkage cluster analysis”. In: *Journal of the Royal Statistical Society, Series C (Applied Statistics)* 18.1 (1969), pp. 54–64. DOI: 10.2307/2346439.

[83] Ágnes Vathy-Fogarassy, Balázs Feil, and János Abonyi. “Minimal spanning tree based fuzzy clustering”. In: *International Journal of Computer and Information Engineering* 1.8 (2007), pp. 2505–2510. DOI: 10.5281/zenodo.1055914.

[84] Balázs Feil and János Abonyi. “Geodesic distance based fuzzy clustering”. In: *Soft Computing in Industrial Applications*. Ed. by Ashraf Saad, Keshav Dahal, Muhammad Sarfraz, and Rajkumar Roy. Vol. 39. Advances in Soft Computing. Berlin, Heidelberg: Springer, 2007, pp. 50–59. DOI: 10.1007/978-3-540-70706-6_5.

[85] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. “A global geometric framework for nonlinear dimensionality reduction”. In: *Science* 290.5500 (2000), pp. 2319–2323. DOI: 10.1126/science.290.5500.2319.

[86] Mira Bernstein, Vin de Silva, John C. Langford, and Joshua B. Tenenbaum. *Graph approximations to geodesics on embedded manifolds*. Tech. rep. Stanford, CA: Department of Psychology, Stanford University, 2000.

[87] Robert C. Prim. “Shortest connection networks and some generalizations”. In: *Bell System Technical Journal* 36.6 (1957), pp. 1389–1401. DOI: 10.1002/j.1538-7305.1957.tb01515.x.

[88] Joseph B. Kruskal. “On the shortest spanning subtree of a graph and the traveling salesman problem”. In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50. DOI: 10.1090/S0002-9939-1956-0078686-7.

[89] Sudhir Varma and Richard Simon. “Iterative class discovery and feature selection using minimal spanning trees”. In: *BMC Bioinformatics* 5 (2004), p. 126. DOI: 10.1186/1471-2105-5-126.

[90] Edsger W. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische Mathematik* 1.1 (1959), pp. 269–271. DOI: 10.1007/BF01386390.

[91] Robert W. Floyd. “Algorithm 97: Shortest path”. In: *Communications of the ACM* 5.6 (1962), p. 345. DOI: 10.1145/367766.368168.

[92] Warren S. Torgerson. “Multidimensional scaling: I. Theory and method”. In: *Psychometrika* 17.4 (1952), pp. 401–419. DOI: 10.1007/BF02288916.

[93] Sam T. Roweis and Lawrence K. Saul. “Nonlinear dimensionality reduction by locally linear embedding”. In: *Science* 290.5500 (2000), pp. 2323–2326. DOI: 10.1126/science.290.5500.2323.

[94] Tomohiro Takagi and Michio Sugeno. “Fuzzy identification of systems and its applications to modeling and control”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-15.1 (1985), pp. 116–132. DOI: 10.1109/TSMC.1985.6313399.

[95] Robert Babuška and Henk B. Verbruggen. “Constructing fuzzy models by product space clustering”. In: *Fuzzy Model Identification: Selected Approaches*. Ed. by Hans Hellendoorn and Dimiter Driankov. Berlin, Heidelberg: Springer, 1997, pp. 53–90. DOI: 10.1007/978-3-642-60767-7_2.

[96] János Abonyi, Robert Babuška, and Ferenc Szeifert. "Modified Gath–Geva fuzzy clustering for identification of Takagi–Sugeno fuzzy models". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 32.5 (2002), pp. 612–621. DOI: 10.1109/TSMCB.2002.1033180.

[97] Richard J. Hathaway and James C. Bezdek. "Switching regression models and fuzzy clustering". In: *IEEE Transactions on Fuzzy Systems* 1.3 (1993), pp. 195–204. DOI: 10.1109/91.236552.

[98] János Abonyi and Ferenc Szeifert. "Supervised fuzzy clustering for the identification of fuzzy classifiers". In: *Pattern Recognition Letters* 24.14 (2003), pp. 2195–2207. DOI: 10.1016/S0167-8655(03)00047-3.

[99] János Abonyi. "Supervised fuzzy clustering based initialization of fuzzy partitions for decision tree induction". In: *Soft Computing in Industrial Applications*. Ed. by Xiao-Zhi Gao, António Gaspar-Cunha, Mario Köppen, Gerald Schaefer, and Jun Wang. Vol. 75. Advances in Intelligent and Soft Computing. Berlin, Heidelberg: Springer, 2010, pp. 31–39. DOI: 10.1007/978-3-642-11282-9_4.

[100] Tamás Kenesei and János Abonyi. "Hinging hyperplane based regression tree identified by fuzzy clustering and its application". In: *Applied Soft Computing* 13.2 (2013), pp. 782–792. DOI: 10.1016/j.asoc.2012.09.027.

[101] Neil Gershenfeld, Bernd Schoner, and Eric Métois. "Cluster-weighted modelling for time-series analysis". In: *Nature* 397.6717 (1999), pp. 329–332. DOI: 10.1038/16873.

[102] János Abonyi and Robert Babuška. "Local and global identification and interpretation of parameters in Takagi–Sugeno fuzzy models". In: *Proceedings of the 9th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2000)*. Vol. 2. San Antonio, TX, 2000, pp. 835–840. DOI: 10.1109/FUZZY.2000.839140.

[103] Euntai Kim, Minkee Park, Seungwoo Kim, and Mignon Park. "A transformed input-domain approach to fuzzy modeling". In: *IEEE Transactions on Fuzzy Systems* 6.4 (1998), pp. 596–604. DOI: 10.1109/91.728458.

[104] János Abonyi and Ferenc Szeifert. "Fuzzy clustering for the identification of Takagi–Sugeno fuzzy models of MIMO dynamical systems". Department of Process Engineering, University of Veszprém, Hungary. 2002.

[105] Balázs Feil, János Abonyi, Sándor Németh, Péter Árva, and János Madár. "Identification and analysis of MIMO systems based on clustering algorithm". In: *Acta Agraria Kaposváriensis* 8.3 (2004), pp. 191–203.

[106] Tor A. Johansen, Robert Shorten, and Roderick Murray-Smith. "On the interpretation and identification of dynamic Takagi–Sugeno fuzzy models". In: *IEEE Transactions on Fuzzy Systems* 8.3 (2000), pp. 297–313. DOI: 10.1109/91.855918.

[107] J. Ross Quinlan. "Induction of decision trees". In: *Machine Learning* 1.1 (1986), pp. 81–106. DOI: 10.1007/BF00116251.

[108] Cezary Z. Janikow. "Fuzzy decision trees: issues and methods". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28.1 (1998), pp. 1–14. DOI: 10.1109/3477.658573.

[109] James Dougherty, Ron Kohavi, and Mehran Sahami. “Supervised and unsupervised discretization of continuous features”. In: *Machine Learning: Proceedings of the Twelfth International Conference*. Ed. by Armand Prieditis and Stuart Russell. San Francisco, CA: Morgan Kaufmann, 1995, pp. 194–202. DOI: 10.1016/B978-1-55860-377-6.50032-3.

[110] Leo Breiman. “Hinging hyperplanes for regression, classification, and function approximation”. In: *IEEE Transactions on Information Theory* 39.3 (1993), pp. 999–1013. DOI: 10.1109/18.256506.

[111] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984.

[112] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. “An online algorithm for segmenting time series”. In: *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001)*. IEEE Computer Society, 2001, pp. 289–296. DOI: 10.1109/ICDM.2001.989531.

[113] Johan Himberg, Kalle Korpiaho, Heikki Mannila, Johanna Tikanmäki, and Hannu T. T. Toivonen. “Time series segmentation for context recognition in mobile devices”. In: *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001)*. IEEE Computer Society, 2001, pp. 203–210. DOI: 10.1109/ICDM.2001.989520.

[114] János Abonyi, Balázs Feil, Sándor Németh, and Péter Árva. “Modified Gath–Geva clustering for fuzzy segmentation of multivariate time-series”. In: *Fuzzy Sets and Systems* 149.1 (2005), pp. 39–56. DOI: 10.1016/j.fss.2004.07.008.

[115] János Abonyi, Balázs Feil, Sándor Németh, and Péter Árva. “Fuzzy clustering based segmentation of time-series”. In: *Advances in Intelligent Data Analysis V: 5th International Symposium on Intelligent Data Analysis, IDA 2003*. Ed. by Michael R. Berthold, Hans-Joachim Lenz, Elizabeth Bradley, Rudolf Kruse, and Christian Borgelt. Vol. 2810. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 275–285. DOI: 10.1007/978-3-540-45231-7_26.

[116] Matthew B. Kennel, Reggie Brown, and Henry D. I. Abarbanel. “Determining embedding dimension for phase-space reconstruction using a geometrical construction”. In: *Physical Review A* 45.6 (1992), pp. 3403–3411. DOI: 10.1103/PhysRevA.45.3403.

[117] Carl Rhodes and Manfred Morari. “Determining the model order of nonlinear input/output systems”. In: *AIChE Journal* 44.1 (1998), pp. 151–163. DOI: 10.1002/aic.690440116.

[118] John D. Bomberger and Dale E. Seborg. “Determination of model order for NARX models directly from input-output data”. In: *Journal of Process Control* 8.5–6 (1998), pp. 459–468. DOI: 10.1016/S0959-1524(98)00013-4.

[119] Balázs Feil, János Abonyi, and Ferenc Szeifert. “Determining the model order of nonlinear input-output systems by fuzzy clustering”. In: *Advances in Soft Computing: Engineering Design and Manufacturing*. Ed. by José Manuel Benítez, Oscar Cordón, Frank Hoffmann, and Rajkumar Roy. London: Springer, 2003, pp. 89–98. DOI: 10.1007/978-1-4471-3744-3_9.

[120] Balázs Feil, János Abonyi, and Ferenc Szeifert. "Model order selection of nonlinear input–output models—a clustering based approach". In: *Journal of Process Control* 14.6 (2004), pp. 593–602. DOI: 10.1016/j.jprocont.2004.01.005.

[121] Floris Takens. "Detecting strange attractors in turbulence". In: *Dynamical Systems and Turbulence, Warwick 1980*. Ed. by David A. Rand and Lai-Sang Young. Vol. 898. Lecture Notes in Mathematics. Berlin, Heidelberg: Springer, 1981, pp. 366–381. DOI: 10.1007/BFb0091924.

[122] Tim Sauer, James A. Yorke, and Martin Casdagli. "Embedology". In: *Journal of Statistical Physics* 65.3–4 (1991), pp. 579–616. DOI: 10.1007/BF01053745.

[123] János Abonyi, Balázs Feil, Sándor Németh, Péter Árva, and Robert Babuška. "State-space reconstruction and prediction of chaotic time series based on fuzzy clustering". In: *Proceedings of the 2004 IEEE International Conference on Systems, Man and Cybernetics*. Vol. 3. IEEE, 2004, pp. 2374–2380. DOI: 10.1109/ICSMC.2004.1400684.

[124] Daniela Girimonte, Robert Babuška, and János Abonyi. "Fuzzy clustering for selecting structure of nonlinear models with mixed discrete and continuous inputs". In: *Proceedings of the 2004 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2004)*. Vol. 1. IEEE, 2004, pp. 383–387. DOI: 10.1109/FUZZY.2004.1375754.

[125] Michael E. Tipping and Christopher M. Bishop. "Mixtures of probabilistic principal component analyzers". In: *Neural Computation* 11.2 (1999), pp. 443–482. DOI: 10.1162/089976699300016728.

[126] Wojtek J. Krzanowski. "Between-groups comparison of principal components". In: *Journal of the American Statistical Association* 74.367 (1979), pp. 703–707. DOI: 10.1080/01621459.1979.10481674.

[127] Andrew M. Fraser and Harry L. Swinney. "Independent coordinates for strange attractors from mutual information". In: *Physical Review A* 33.2 (1986), pp. 1134–1140. DOI: 10.1103/PhysRevA.33.1134.

[128] J. Doyne Farmer and John J. Sidorowich. "Predicting chaotic time series". In: *Physical Review Letters* 59.8 (1987), pp. 845–848. DOI: 10.1103/PhysRevLett.59.845.

[129] Martin Casdagli. "Nonlinear prediction of chaotic time series". In: *Physica D: Nonlinear Phenomena* 35.3 (1989), pp. 335–356. DOI: 10.1016/0167-2789(89)90074-2.

[130] Peter Grassberger and Itamar Procaccia. "Measuring the strangeness of strange attractors". In: *Physica D: Nonlinear Phenomena* 9.1–2 (1983), pp. 189–208. DOI: 10.1016/0167-2789(83)90298-1.

[131] Xiangdong He and Haruhiko Asada. "A new method for identifying orders of input-output models for nonlinear dynamic systems". In: *Proceedings of the 1993 American Control Conference*. San Francisco, CA: IEEE, 1993, pp. 2520–2523. DOI: 10.23919/ACC.1993.4793346.

[132] Francis J. Doyle III, Babatunde A. Ogunnaike, and Ronald K. Pearson. "Nonlinear model-based control using second-order Volterra models". In: *Automatica* 31.5 (1995), pp. 697–714. DOI: 10.1016/0005-1098(94)00150-H.

[133] Lennart Ljung. *System Identification: Theory for the User*. 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

[134] Liangyue Cao. "Practical method for determining the minimum embedding dimension of a scalar time series". In: *Physica D: Nonlinear Phenomena* 110.1–2 (1997), pp. 43–50. DOI: 10.1016/S0167-2789(97)00118-8.

[135] Xiaomo Jiang and Hojjat Adeli. "Fuzzy clustering approach for accurate embedding dimension identification in chaotic time series". In: *Integrated Computer-Aided Engineering* 10.3 (2003), pp. 287–302. DOI: 10.3233/ICA-2003-10305.

[136] Roderick Murray-Smith and Tor Arne Johansen, eds. *Multiple Model Approaches to Nonlinear Modelling and Control*. London: Taylor & Francis, 1997.

[137] Michael C. Mackey and Leon Glass. "Oscillation and chaos in physiological control systems". In: *Science* 197.4300 (1977), pp. 287–289. DOI: 10.1126/science.267326.

[138] Edward N. Lorenz. "Deterministic nonperiodic flow". In: *Journal of the Atmospheric Sciences* 20.2 (1963), pp. 130–141. DOI: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[139] Otto E. Rössler. "An equation for continuous chaos". In: *Physics Letters A* 57.5 (1976), pp. 397–398. DOI: 10.1016/0375-9601(76)90101-8.

[140] Jerome H. Friedman. "Multivariate adaptive regression splines". In: *The Annals of Statistics* 19.1 (1991), pp. 1–67. DOI: 10.1214/aos/1176347963.

[141] Richard J. Hathaway, John W. Davenport, and James C. Bezdek. "Relational duals of the c-means clustering algorithms". In: *Pattern Recognition* 22.2 (1989), pp. 205–212. DOI: 10.1016/0031-3203(89)90066-6.

[142] Richard J. Hathaway and James C. Bezdek. "NERF c-means: non-Euclidean relational fuzzy clustering". In: *Pattern Recognition* 27.3 (1994), pp. 429–437. DOI: 10.1016/0031-3203(94)90119-8.

[143] Raghu Krishnapuram, Anupam Joshi, Olfa Nasraoui, and Liyu Yi. "Low-complexity fuzzy relational clustering algorithms for Web mining". In: *IEEE Transactions on Fuzzy Systems* 9.4 (2001), pp. 595–607. DOI: 10.1109/91.940971.

[144] Zhexue Huang. "Extensions to the k-means algorithm for clustering large data sets with categorical values". In: *Data Mining and Knowledge Discovery* 2.3 (1998), pp. 283–304. DOI: 10.1023/A:1009769707641.

[145] Zhexue Huang and Michael K. Ng. "A fuzzy k-modes algorithm for clustering categorical data". In: *IEEE Transactions on Fuzzy Systems* 7.4 (1999), pp. 446–452. DOI: 10.1109/91.784206.

[146] John C. Gower and Pierre Legendre. "Metric and Euclidean properties of dissimilarity coefficients". In: *Journal of Classification* 3.1 (1986), pp. 5–48. DOI: 10.1007/BF01896809.

[147] Hiroaki Sakoe and Seibi Chiba. "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49. DOI: 10.1109/TASSP.1978.1163055.

[148] Pierpaolo D'Urso and Elizabeth Ann Maharaj. "Autocorrelation-based fuzzy clustering of time series". In: *Fuzzy Sets and Systems* 160.24 (2009), pp. 3565–3589. DOI: 10.1016/j.fss.2009.04.013.

[149] John C. Gower. "A general coefficient of similarity and some of its properties". In: *Biometrics* 27.4 (1971), pp. 857–871. DOI: 10.2307/2528823.

[150] James C. Bezdek, James M. Keller, Raghu Krishnapuram, and Nikhil R. Pal. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Vol. 4. The Handbooks of Fuzzy Sets. Boston: Kluwer Academic Publishers, 1999. DOI: 10.1007/b106267.

[151] Richard J. Hathaway and James C. Bezdek. "Fuzzy c-means clustering of incomplete data". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 31.5 (2001), pp. 735–744. DOI: 10.1109/3477.956035.

[152] John K. Dixon. "Pattern recognition with partly missing data". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.10 (1979), pp. 617–621. DOI: 10.1109/TSMC.1979.4310090.

[153] Roderick J. A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data*. 2nd ed. Hoboken, NJ: Wiley, 2002. DOI: 10.1002/9781119013563.

[154] Robert L. Cannon, Jitendra V. Dave, and James C. Bezdek. "Efficient implementation of the fuzzy c-means clustering algorithms". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8.2 (1986), pp. 248–255. DOI: 10.1109/TPAMI.1986.4767778.

[155] John F. Kolen and Tim Hutcheson. "Reducing the time complexity of the fuzzy c-means algorithm". In: *IEEE Transactions on Fuzzy Systems* 10.2 (2002), pp. 263–267. DOI: 10.1109/91.995126.

[156] Timothy C. Havens, James C. Bezdek, Christopher Leckie, Lawrence O. Hall, and Marimuthu Palaniswami. "Fuzzy c-means algorithms for very large data". In: *IEEE Transactions on Fuzzy Systems* 20.6 (2012), pp. 1130–1146. DOI: 10.1109/TFUZZ.2012.2201485.

[157] Prodip Hore, Lawrence O. Hall, and Dmitry B. Goldgof. "Single pass fuzzy c means". In: *Proceedings of the 2007 IEEE International Fuzzy Systems Conference (FUZZ-IEEE 2007)*. London, UK, 2007, pp. 1–7. DOI: 10.1109/FUZZY.2007.4295372.

[158] Prodip Hore, Lawrence O. Hall, Dmitry B. Goldgof, Yuhua Gu, Andrew A. Maudsley, and Ammar Darkazanli. "A scalable framework for segmenting magnetic resonance images". In: *Journal of Signal Processing Systems* 54.1–3 (2009), pp. 183–203. DOI: 10.1007/s11265-008-0243-1.

[159] Mohamed N. Ahmed, Sameh M. Yamany, Nevin Mohamed, Aly A. Farag, and Thomas Moriarty. "A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data". In: *IEEE Transactions on Medical Imaging* 21.3 (2002), pp. 193–199. DOI: 10.1109/42.996338.

[160] Songcan Chen and Daoqiang Zhang. "Robust image segmentation using FCM with spatial constraints based on new kernel-induced distance measure". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.4 (2004), pp. 1907–1916. DOI: 10.1109/TSMCB.2004.831165.

[161] László Szilágyi, Zoltán Benyó, Sándor M. Szilágyi, and H. S. Adam. "MR brain image segmentation using an enhanced fuzzy c-means algorithm". In: *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. Cancún, Mexico, 2003, pp. 724–726. DOI: 10.1109/IEMBS.2003.1279866.

[162] Weiling Cai, Songcan Chen, and Daoqiang Zhang. “Fast and robust fuzzy c-means clustering algorithms incorporating local information for image segmentation”. In: *Pattern Recognition* 40.3 (2007), pp. 825–838. DOI: 10.1016/j.patcog.2006.07.011.

[163] Stelios Krinidis and Vassilios Chatzis. “A robust fuzzy local information c-means clustering algorithm”. In: *IEEE Transactions on Image Processing* 19.5 (2010), pp. 1328–1337. DOI: 10.1109/TIP.2010.2040763.

[164] Dzung L. Pham and Jerry L. Prince. “Adaptive fuzzy segmentation of magnetic resonance images”. In: *IEEE Transactions on Medical Imaging* 18.9 (1999), pp. 737–752. DOI: 10.1109/42.802752.

[165] Witold Pedrycz and James Waletzky. “Fuzzy clustering with partial supervision”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.5 (1997), pp. 787–795. DOI: 10.1109/3477.623232.

[166] Nizar Grira, Michel Crucianu, and Nozha Boujemaa. “Fuzzy clustering with pairwise constraints for knowledge-driven image categorisation”. In: *IEE Proceedings – Vision, Image and Signal Processing* 153.3 (2006), pp. 299–304. DOI: 10.1049/ip-vis:20050060.

[167] Hichem Frigui and Raghu Krishnapuram. “Clustering by competitive agglomeration”. In: *Pattern Recognition* 30.7 (1997), pp. 1109–1119. DOI: 10.1016/S0031-3203(96)00140-9.

[168] Chi-Hyon Oh, Katsuhiro Honda, and Hidetomo Ichihashi. “Fuzzy clustering for categorical multivariate data”. In: *Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference*. Vol. 4. Vancouver, BC, Canada, 2001, pp. 2154–2159. DOI: 10.1109/NAFIPS.2001.944403.

[169] Krishna Kummamuru, Ajay Dhawale, and Raghu Krishnapuram. “Fuzzy co-clustering of documents and keywords”. In: *Proceedings of the 12th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2003)*. Vol. 2. St. Louis, MO, USA, 2003, pp. 772–777. DOI: 10.1109/FUZZ.2003.1206527.

[170] Witold Pedrycz. “Collaborative fuzzy clustering”. In: *Pattern Recognition Letters* 23.14 (2002), pp. 1675–1686. DOI: 10.1016/S0167-8655(02)00130-7.

[171] Alexander Strehl and Joydeep Ghosh. “Cluster ensembles—a knowledge reuse framework for combining multiple partitions”. In: *Journal of Machine Learning Research* 3 (2002), pp. 583–617.

[172] Kunal Punera and Joydeep Ghosh. “Consensus-based ensembles of soft clusterings”. In: *Applied Artificial Intelligence* 22.7–8 (2008), pp. 780–810. DOI: 10.1080/08839510802170546.

[173] Ana L. N. Fred and Anil K. Jain. “Combining multiple clusterings using evidence accumulation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.6 (2005), pp. 835–850. DOI: 10.1109/TPAMI.2005.113.

[174] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. “On spectral clustering: analysis and an algorithm”. In: *Advances in Neural Information Processing Systems 14*. Ed. by Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani. Cambridge, MA: MIT Press, 2002, pp. 849–856.

[175] George Karypis and Vipin Kumar. "A fast and high quality multilevel scheme for partitioning irregular graphs". In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392. DOI: 10.1137/S1064827595287997.

Index

A-FCM, 244
A-FCMd, 245
adaptive cluster merging, 121, 122
aggregation degree, 295
alternating optimization, 16
autocorrelation-based fuzzy clustering, 244
`autocorrelation_distance_matrix`, 245

`BaseFuzzyClusterer`, 16
β-spread, 239
bottom-up segmentation, 210
`bottom_up_segmentation`, 210

c-means functional, 14
cannot-link constraint, 293
chaotic time series, 216
`ChaoticPredictor`, 216
cluster ensemble, 299
cluster merging, *see* compatible cluster merging
co-association matrix, 299
co-clustering, 295
coincident clusters, 110
collaborative fuzzy clustering, 297
`CollaborativeFCM`, 297
comparing partitions, 72
compatible cluster merging, 70, 73
`CompatibleClusterMerging`, 73
competitive agglomeration, 293
consensus clustering, 299
covariance regularization, 35

data sites, 297
decomposition error, 170
delay embedding, 216
deterministic annealing, 39, 42
discretization
  supervised, 181
DTW, 243
duality of FCM and RFCM, 238
dynamic time warping, 243

eigenstructure of a cluster, 34
EnFCM, 272
entropy regularization, 39, 42
Euclidean dissimilarity matrix, 236
evidence accumulation, 301
`evidence_accumulation`, 301

false nearest neighbors (FNN), 213
`FCCM`, 295
FCM, *see* fuzzy c-means
`FCM`, 33
FCM_S, 271
FCM_S1, FCM_S2, 272
FCMdd, 241
`FCoDoK`, 296
FCTMdd, 241
feature weighting, 121, 124
  cluster-specific, 125
feature-weighted fuzzy c-means, 123, 124
`FeatureWeightedFCM`, 124
FGFCM, 273
FLICM, 274

fnn_model_order, 213
forgetting factor, 269
fuzzifier, 38
    polynomial, 43
fuzziness exponent, 14
    role of, 32
FUZZSAM, *see* fuzzy Sammon mapping
fuzzy adjusted Rand index, 74
fuzzy c-elliptotypes, 118
fuzzy c-lines, 118
fuzzy c-means (FCM), 33
    incomplete data, 266
fuzzy c-medoids, 241
fuzzy c-regression models (FCRM), 177
fuzzy c-shells (FCS), 120
fuzzy c-spherical shells (FCSS), 120
fuzzy c-trimmed medoids, 241
fuzzy c-varieties (FCV), 118
fuzzy classifier, 179
Fuzzy CoDoK, 296
fuzzy hypervolume, 68
    as splitting criterion, 152
fuzzy k-modes, 242
fuzzy k-prototypes, 243
fuzzy local information c-means, 274
fuzzy maximum likelihood estimation, 37
fuzzy Rand index, 74
fuzzy Sammon mapping (FUZZSAM), 93
fuzzy segmentation, 209
fuzzy self-organizing map, 96
fuzzy variance decomposition, 218
FuzzyClusterEnsemble, 299
FuzzyCMedoids, 241
FuzzyCRegression, 177
FuzzyCShells, 120
FuzzyCVarieties, 118
FuzzyKModes, 242
FuzzyKPrototypes, 243
FuzzyPartitionDiscretizer, 181
FuzzySammon, 93
FuzzySegmentation, 209
FuzzySOM, 96

Gath–Geva algorithm (GG), 37
geodesic distance, 154
geodesic fuzzy c-medoids, 154
GeodesicFCM, 154
GG, *see* Gath–Geva algorithm
GG, 37
GK, *see* Gustafson–Kessel algorithm
GK, 35
Gower's coefficient, 245
gower_distance_matrix, 245
Gustafson–Kessel algorithm (GK), 35

hard c-means, *see* k-means
hinging hyperplane, 183
HingingHyperplaneTree, 183
histogram-based fuzzy c-means, 272
hybrid MST–GG clustering, 152

IncompleteFCM, 266
information-theoretic k-means, 299
initialization, 15
Isomap, 155
IsomapFCM, 155

k-means, 29
k-medoids, 30
k-modes, 242
k-prototypes, 243
kernel fuzzy c-means, 116
kernel trick, 116
KernelFCM, 116
KMeans, 29
KMedoids, 30

lifetime criterion, 301
linear fuzzy c-medoids, 241

medoid, 30
minimal spanning tree clustering, 152
mixed data, 243
MixedInputStructureSelection, 218

mode of a fuzzy cluster, 242
model-order selection, 213
modified Gath–Geva algorithm, 175
ModifiedGG, 175
mountain method, 43, 46
MountainClustering, 46
MSTClustering, 152
must-link constraint, 293

NERF c-means, 239
NERFCM, 239
noise clustering, 114
noise prototype, 114
NoiseClustering, 114
non-Euclidean relational fuzzy c-means, 239
norm-inducing matrix, 11
number of clusters
  density-based estimate, 41
  lifetime criterion, 301
  stability selection, 75
  validity scan, 72

OCS, *see* optimal completion strategy
oFCM, *see* online fuzzy c-means
online fuzzy c-means, 270
OnlineFCM, 270
optimal completion strategy, 266

pairwise constraints, 293
PairwiseConstrainedFCM, 293
partial distance, 266
partial supervision, 292
partition
  fuzzy, 13
  hard, 12
  possibilistic, 13
partitioning space, 12
pca_projection, 90
PCCA, 293
PCM, 111
PDS, *see* partial distance
PFCM, 113
Picard iteration, *see* alternating optimization
polynomial fuzzifier, 40, 43
PolynomialFuzzifierFCM, 43
possibilistic c-means (PCM), 111
possibilistic fuzzy c-means (PFCM), 113
principal component analysis (PCA), 90
probabilistic PCA, 209
projection quality, 94

quadratic regularization, 40, 42

random sample and extend FCM, 268
RandomSampleFCM, 268
regression tree
  hinging-hyperplane, 183
RegularizedFCM, 42
relational fuzzy c-means, 238
RelationalFCM, 238
restarts, 15
RFCM, 238
rseFCM, *see* random sample and extend FCM

Sammon, 91
Sammon mapping, 91
SCAD, 125
scaling, 18
scan_n_clusters, 72
self-organizing map
  fuzzy, 96
semi-supervised fuzzy c-means, 292
SemiSupervisedFCM, 292
simple matching dissimilarity, 242
simultaneous clustering and attribute discrimination (SCAD), 123, 125
single-pass fuzzy c-means, 269
SinglePassFCM, 269
sMCLA, 299
SpatialFCM, 271
spatially constrained fuzzy c-means, 271

spFCM, *see* single-pass fuzzy c-means
SSFCM, *see* semi-supervised fuzzy c-means
stability selection, 75, 76
`stability_selection`, 76
state-space reconstruction, 216
stress (Sammon), 91
structure selection, 218
subtractive clustering, 44, 45
`SubtractiveClustering`, 45
supervised fuzzy clustering, 179
supervised fuzzy partition, 181
`SupervisedFuzzyClassifier`, 179
switching regression, 177

Takagi–Sugeno model, 169
  identification by clustering, 175
`TakagiSugeno`, 175
time series
  segmentation, 209
typicality, 111

U-matrix, 95

validity index
  alternative Dunn index (ADI), 68
  average partition density (APD), 69
  classification entropy (CE), 66
  Dunn's index (DI), 68
  Fukuyama–Sugeno (FS), 67
  fuzzy hypervolume (FHV), 68
  Kwon (KWON), 69
  modified partition coefficient (MPC), 66
  partition coefficient (PC), 66
  partition density (PD), 69
  partition index (SC), 66
  PBM-F, 67
  PCAES, 69
  separation index (S), 67
  silhouette (SIL), 68
  Xie–Beni (XB), 67
validity scan, 72
volume prototype, 120, 122
`VolumePrototypeFCM`, 122

weighted fuzzy c-means (wFCM), 267
`WeightedFCM`, 267
wFCM, *see* weighted fuzzy c-means (wFCM)

Zahn's criteria, 152



From the c-means functional to model identification

Fuzzy clustering assigns every observation a degree of membership in every group instead of forcing it into one. This book develops the main families of fuzzy clustering algorithms in one notation and shows, for each of them, how to run it in Python and how its parameters change the result.

The core chapters treat fuzzy c-means and the Gustafson–Kessel and Gath–Geva algorithms, cluster validity, visualization by fuzzy Sammon mapping, and possibilistic, kernel, shell, and graph-based variants. Later chapters cover Takagi–Sugeno model identification, time-series segmentation, relational and mixed data, missing values, very large data, image segmentation, partial supervision, co-clustering, collaborative clustering, and cluster ensembles.

Every method is implemented in `fuzzyclust`, an open-source Python package with a scikit-learn interface and the successor of the *Fuzzy Clustering and Data Analysis Toolbox* for MATLAB. Each figure in the book is produced by an example script that generates its own data; the same scripts run as Jupyter notebooks.

```
pip install fuzzyclust
```

github.com/ComplexSystemsIntelligence/fuzzyclust