Fuzzy Clustering of Sampled Functions*

Frank Höppner** Frank Klawonn

University of Applied Sciences, Emden
Department of Electrical Engineering and Computer Science
Constantiaplatz 4
D-26723 Emden, Germany

Abstract

Fuzzy clustering algorithms perform cluster analysis on a data set that consists of feature attribute vectors. In the context of multiple sampled functions, a set of samples (e.g. a time series) becomes a single datum. We show how the already known algorithms can be used to perform fuzzy cluster analysis on this kind of data sets by replacing the conventional prototypes with sets of prototypes. This approach allows reusing the known algorithms and works also with other data than sampled functions. Furthermore, to reduce the computational costs in case of single-input/single-output functions we present a new fuzzy clustering algorithm, which uses for the first time a more complex input data type (data points aggregated to data-lines instead of raw data). The new alternating optimisation algorithm performs cluster analysis directly on this more compact representation of the sampled functions.

Keywords: fuzzy clustering, function clustering, sets-of-features analysis, continuous piecewise linear function approximation, aggregated data, linear features

1 Introduction

Fuzzy clustering algorithms like the algorithm by Gustafson and Kessel (GK) [6], Gath and Geva (GG) [5], or the fuzzy c-varieties (FCV) algorithm [1] are capable of detecting linear substructures (clusters) in a set of feature attribute vectors (see [9] for a detailed discussion). These algorithms therefore have been used intensively to construct fuzzy models automatically from data, see [10] for instance. This data may come from multiple observations of system behaviour versus time. However, it is not clear whether the data (which comes from *several* observations) should be used to build a *single* model. For example, if the system has multiple internal states it is possible that we obtain (partially) completely different observations. These observations should not be mixed up if we want to develop precise models. How can we find out how many models are necessary (or how many different states we have observed)? How can we identify multiple models? These questions lead us to the problem of "function clustering", where a single datum is a time series or sampled function (characterising a systems behaviour).

In the next section, we shortly review objective-function based fuzzy clustering methods. In section 3 we show how conventional fuzzy clustering algorithms can be used to solve this problem. In case of single-input single-output (SI-SO) functions we can use the Hard c-Connected Lines (HCCL) algorithm [8] instead of the GK/GG/FCV algorithm, to generate piecewise linear approximations of the data. The method we propose in section 3 considers each datum of every sampled function individually and therefore has to process a large number of data vectors. In section 4 we therefore generalise the HCCL algorithm to work on line segments (aggregated points) instead of single points only. This allows preprocessing the observations in order to obtain smaller data sets which speeds up the clustering process.

2 Objective Function-Based Fuzzy Clustering

The process of subdividing a data set $X = \{x_1, ..., x_n\} \subset \mathbb{R}^{\mathrm{DIM}}$, $\mathrm{DIM} \in \mathbb{N}$, in distinct, meaningful subsets is called clustering. With fuzzy clustering each datum belongs to all clusters simul-

*This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant no. Kl 648/1-1.

** e-mail alias: frank.hoeppner@ieee.org

taneously, but to different degrees. Many fuzzy clustering algorithms minimise the sum of weighted (squared) distances between data objects and cluster representatives $\{p_1, ..., p_c\}$

$$J_m(X; U, P) = \sum_{j=1}^{n} \sum_{i=1}^{c} u_{i,j}^m d_{i,j}^2 \qquad (1)$$

taking the constraints

$$\forall i \in \mathbb{N}_{\leq c} \quad : \quad \sum_{j=1}^{n} u_{i,j} > 0 \qquad (2)$$

$$\forall j \in \mathbb{N}_{\leq n} \quad : \quad \sum_{i=1}^{c} u_{i,j} = 1 \qquad (3)$$

into account. The membership degree of datum x_j to cluster p_i is denoted by $u_{i,j} \in [0, 1]$. The distance or similarity of datum x_j and cluster prototype p_i is denoted by $d_{i,j}$. The parameter $m > 1$ is called fuzziness index and influences the "fuzziness" of the obtained partition. Constraint (2) makes sure that none of the clusters is empty and thus we really have a partition into c clusters. Constraint (3) assures that every datum has the same overall weight in the data set. Fuzzy clustering under constraints (2) and (3) is often called "probabilistic clustering". Other fuzzy clustering techniques, using a relaxed constraint (3), are noise clustering [2] and possibilistic clustering [11]. The latter approaches are especially useful when dealing with very noisy data.

The most popular fuzzy clustering algorithm is the fuzzy c-means algorithm [1]. There, a cluster is represented by a "typical datum" $p_i \in \mathbb{R}^{\mathrm{DIM}}$. The Euclidean distance between data vector x_j and prototype p_i is used as the distance measure. This "cluster shape model" searches for spherical clusters of approximately the same size.

Most of the objective function based fuzzy clustering algorithms minimise (1) by alternatingly optimising the membership degrees and cluster shapes. From the membership model (e.g. "probabilistic") and the cluster shape model (e.g. "point-like") one can develop necessary conditions for a local minimiser of J from $\frac{\partial J}{\partial U} = 0$ and $\frac{\partial J}{\partial P} = 0$. Due to space limitations we refer to the literature [1] for the update equations of FCM.

3 Clustering of Sampled Functions

Let $\mathcal{S}$ be a set of time series or sampled functions $\mathcal{S} = \{S_1, ..., S_n\}$. Each S_j is a set of feature attributes, $S_j = \{x_{j,1}, x_{j,2}, ..., x_{j,n_j}\} \subset \mathbb{R}^{\mathrm{DIM}}$. Let A be an objective function-based clustering algorithm that is capable of analysing a weighted data set: We can easily extend any of the known fuzzy clustering algorithms to involve an additional weight w_j for each datum $x_j \in X$, leading us from (1) to the extended objective function

$$J_m(X, W; U, P) = \sum_{j=1}^{n} \sum_{i=1}^{c} u_{i,j}^m w_j d_{i,j}^2 \qquad (4)$$

Since the weights are fixed and not subject to optimisation, we obtain similar update equations as before. A weight of $w_j = 2$ has the same effect as considering data object x_j in the sum (1) twice.

As already mentioned, fuzzy clustering algorithms that detect linear substructures in the data are often used to construct fuzzy models automatically from data. During clustering each prototype approximates the system more or less locally. The complete model is characterized by the "set of prototypes", the set becomes a "model prototype". Let $\mathcal{P}$ be the set of all possible prototypes of algorithm A ($\mathcal{P} = \mathbb{R}^{\mathrm{DIM}}$ for the fuzzy c-means algorithm). In order to "approximate" a sampled function S_j we use a set of prototypes $P_i = \{p_{i,1}, p_{i,2}, ..., p_{i,c_i}\} \subset \mathcal{P}$. To define a distance measure that reflects the distance of a data series S_j to a model prototype P_i it is straightforward to use

$$d^2(S_j, P_i) := \sum_{k=1}^{n_j} \min\{d_A^2(x_{j,k}, p_{i,l}) \mid l \in \{1, .., c_i\}\} \qquad (5)$$

where d_A denotes the distance measure used by algorithm A. Due to the minimum function it is impossible to solve for the prototype parameters directly.

To avoid expensive numerical techniques, let us shortly review the development of the first fuzzy clustering algorithm. The hard ISODATA algorithm [3] uses no fuzzy partition and assigns each datum unambiguously to a cluster. A datum therefore belongs always to the cluster with the closest distance. The hard ISODATA algorithm minimises the objective function

$$J_{HCM}(X; P) = \sum_{j=1}^{n} \min_{i=1..c} \quad d^2(x_j, p_i) \qquad (6)$$

ISODATA has been "fuzzified" [4] to the fuzzy c-means algorithm. When comparing (6) with (1) we can see that with the fuzzification the $\min d_{i,j}^2$ term has been replaced by the weighted sum $\sum_{i=1}^{c} u_{i,j}^m d_{i,j}^2$. In analogy we replace the minimum term in the distance function (5). Thus the

objective function (1) of algorithm A becomes the distance of the sampled function S_j to the model prototype P_i.

We obtain the objective function of a fuzzy clustering algorithm that partitions data series S_j into sets of prototypes P_i from

$$J_m(X;U,P) = \sum_{j=1}^{n}\sum_{i=1}^{c} u_{i,j}^m d_{i,j}^2$$

$$= \sum_{j=1}^{n}\sum_{i=1}^{c} u_{i,j}^m \sum_{k=1}^{n_j}\sum_{l=1}^{c_i} \hat{u}_{l,k}^{\hat{m}} d_A^2(x_{j,k}, p_{i,l}) \qquad (7)$$

where $\hat{m}$ and $\hat{u}$ are the corresponding fuzziness index and membership matrix of algorithm A. From this objective function we obtain membership and prototype update equations as follows:

- *Membership Update:* Given prototypes P_i and data series S_j we can calculate the memberships $\hat{u}$ of algorithm A using S_j as input data set and P_i as prototypes. From d_A and $\hat{u}$ we can calculate the objective function of algorithm A, which becomes the distance in the new algorithm. From the distances we obtain the membership update equations as usually (depending on the membership model).
- *Prototype Update:* In order to update a model (or set of prototypes) P_i we simply call algorithm A with the weighted data set $X = \cup_{j=1..n} S_j$, assigning the weight $u_{i,j}$ to every element of data series S_j. Using P_i as the prototypes the extended objective function (4) of algorithm A becomes (7).

Example

Fifty data series, each consisting of fifty samples, are plotted simultaneously in figure 1(a). All we may realise from this plot is that the noise seems to vary along time. Examining all the noisy time series one by one may reveal some other information, however, it is difficult for the human observer to compare 50 plots of similar time series with each other (figure 1(b) shows a single datum as an example). The figures 1(c) - 1(d) show two out of four clusters (resp. models) identified by the algorithm. In this case, fuzzy clustering revealed that each sampled function has two peaks, one on the negative and one on the positive x-axis. In our example each model consists of six GK clusters. However, it is also possible to use an unsupervised clustering algorithm A, in order to automatically adjust the number of prototypes that are necessary for each model (we will do this in section 4).

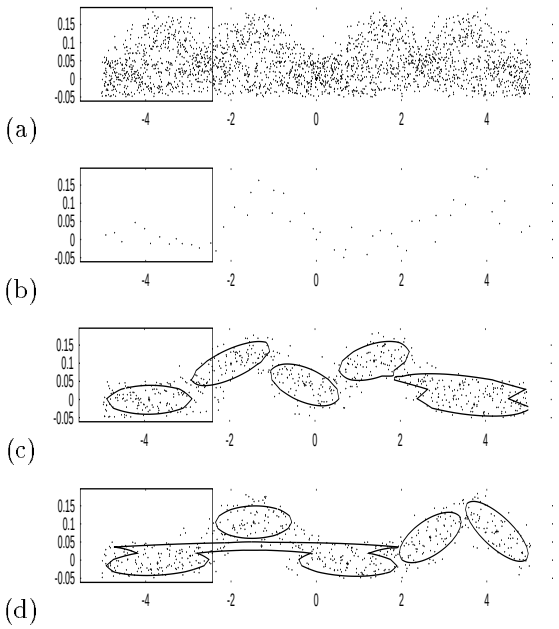

Figure 1: Clustering of time series using the extended GK algorithm. From top to bottom: all time series, a single time series, two out of four identified models together with the associated data.

The approach proposed so far works with sampled functions as well as with any other application where each datum is a set of features. For instance it is possible to cluster different geometric shapes, each of it described by a set of contour points, with a number of point-like or linear clusters. But if it is clear from the application, as in our example, that we deal with sampled single-input single-output functions, other algorithms might be more appropriate for this task than GK. For example, in figure 1(d) we can observe that there is a long stretched cluster on the left which forces the neighbouring clusters to re-orientate, too. If we consider the resulting partition only, we are satisfied with the result, but if we want to use the prototypes to develop a fuzzy model of the function we are not. In [8] we introduced the unsupervised Hard c-Connected Lines (HCCL) algorithm which finds a continuous piecewise linear approximation of the function with optimised number and location of knot points. It therefore suits the purposes of this special case better than the GK algorithm, classifications as in figure 1(d) cannot occur.

However, regardless which algorithm A we finally use, since the algorithm is called for a very large data set $\cup S_j$ several times within the prototype update step – and unsupervised versions of A call A itself multiple times – this approach suffers from

high computational costs. In the following section we try to overcome this problem in case of real valued (SISO) functions.

4 HCCL Algorithm with Linear Features

In this section we modify the unsupervised HCCL algorithm [8] to work not only with single points but with line segments (that may represent aggregated points). The HCCL algorithm can be considered as a hard c-varieties algorithm with coupled line prototypes (HCCL is restricted to 2D case), e.g. one linear cluster starts where another has ended. The prototypes represent the sequence of knot points that define the line segments. The (squared) HCCL distance of a datum $(x_j\ \hat{x}_j)^\top$ to the line segment between prototype $(k_i\ \hat{k}_i)^\top$ and $(k_{i+1}\ \hat{k}_{i+1})^\top$ is given by the dot product of point difference and normal vector [8]:

$$d_{i,j}^2 = \left(\left(\begin{pmatrix} x_j \\ \hat{x}_j \end{pmatrix} - \begin{pmatrix} k_i \\ \hat{k}_i \end{pmatrix} \right)^\top \begin{pmatrix} \hat{k}_i - \hat{k}_{i+1} \\ k_{i+1} - k_i \end{pmatrix} \right)^2 \quad (8)$$

The distance of a datum to the piecewise linear function that is defined by the HCCL prototypes is obtained from $\sum_{i=1}^{k} u_{i,j} d_{i,j}^2$, where $u_{i,j}$ denotes the (hard) membership degree of the jth datum to the ith line segment, that is

$$u_{i,j} = 1 \Leftrightarrow x_j \in I_i, \text{ with } I_i := [k_i, k_{i+1}[\quad (9)$$

Thus the hard membership degrees are parameterised by the knots k_i such that we always obtain a continuous piecewise linear approximation of the data. (Optimisation with respect to memberships is therefore an optimisation with respect to knot values k_i.) The HCCL algorithm minimises the overall approximation error, that is, the objective function $\sum_{j=1}^{n} \sum_{i=1}^{c-1} u_{i,j} d_{i,j}^2$.

For each time series S_j we can apply the unsupervised HCCL algorithm and obtain prototypes representing a piecewise linear approximation of the data. If we *replace* S_j with these prototypes, the original data is approximated with much less storage needs (data compression). Let us therefore consider a data set that contains line segments leading from $(x_j\ \hat{x}_j)^\top$ to $(x_{j+1}\ \hat{x}_{j+1})^\top$. If we assume that we can find an $i \in \mathbb{N}_{<c}$ for each $j \in \mathbb{N}_{\leq n}$ such that $[x_j, x_{j+1}] \subset [k_i, k_{i+1}]$ (ranges do not overlap), we can generalise the point distance (8) to

$$d_{i,j}^2 = \int_0^1 \left(\Delta_{i,j}(\lambda)^\top \begin{pmatrix} \hat{k}_i - \hat{k}_{i+1} \\ k_{i+1} - k_i \end{pmatrix} \right)^2 d\lambda \quad (10)$$

with

$$\Delta_{i,j}(\lambda) = \begin{pmatrix} x_j \\ \hat{x}_j \end{pmatrix} + \lambda \begin{pmatrix} x_{j+1} - x_j \\ \hat{x}_{j+1} - \hat{x}_j \end{pmatrix} - \begin{pmatrix} k_i \\ \hat{k}_i \end{pmatrix}$$

This leads us to

Theorem 1 *If J, given by (4), (9) and (10), is minimised with respect to knot-values $k = (k_1, ..., k_c)$ (resp. $\hat{k} = (\hat{k}_1, ..., \hat{k}_c)$), the equation $k = -T^{-1}s$ (resp. $\hat{k} = -T^{-1}s$) holds with $T \in \mathbb{R}^{c \times c}$, $s \in \mathbb{R}^c$ and $s_i = \sum_{j=1}^{n-1} \alpha_j$, $T_{i,i-1} = \sum_{j=1}^{n-1} \beta_j$, $T_{i,i} = \sum_{j=1}^{n-1} \gamma_j$, $T_{i,i+1} = \sum_{j=1}^{n-1} \delta_j$ ($T_{i,j} = 0$ otherwise), where*

$$\begin{aligned}
\alpha_j &= u_{i-1,j}(\hat{k}_i - \hat{k}_{i-1}) \left((3\hat{k}_{i-1} - 2\hat{x}_{j+1} - \hat{x}_j)x_{j+1} \right. \\
&\quad \left. + (3\hat{k}_{i-1} - 2\hat{x}_j - \hat{x}_{j+1})x_j \right) \\
&\quad - u_{i,j}(\hat{k}_{i+1} - \hat{k}_i) \left((3\hat{k}_{i+1} - 2\hat{x}_{j+1} - \hat{x}_j)x_{j+1} \right. \\
&\quad \left. + (3\hat{k}_{i+1} - 2\hat{x}_j - \hat{x}_{j+1})x_j \right) \\
\beta_j &= u_{i-1,j} \left((\hat{x}_{j+1} - \hat{x}_j)(3\hat{k}_i - 2\hat{x}_{j+1} - \hat{x}_j) \right. \\
&\quad \left. - 3(\hat{k}_{i-1} - \hat{x}_j)(2\hat{k}_i - \hat{x}_{j+1} - \hat{x}_j) \right) \\
\gamma_j &= u_{i-1,j} \left(-(\hat{x}_{j+1} - \hat{x}_j)(3\hat{k}_{i-1} - 2\hat{x}_{j+1} - \hat{x}_j) \right. \\
&\quad \left. + 3(\hat{k}_{i-1} - \hat{x}_j)(2\hat{k}_{i-1} - \hat{x}_{j+1} - \hat{x}_j) \right) \\
&\quad + u_{i,j} \left(-(\hat{x}_{j+1} - \hat{x}_j)(3\hat{k}_{i+1} - 2\hat{x}_{j+1} - \hat{x}_j) \right. \\
&\quad \left. + 3(\hat{k}_{i+1} - \hat{x}_j)(2\hat{k}_{i+1} - \hat{x}_{j+1} - \hat{x}_j) \right) \\
\delta_j &= u_{i,j} \left((\hat{x}_{j+1} - \hat{x}_j)(3\hat{k}_i - 2\hat{x}_{j+1} - \hat{x}_j) \right. \\
&\quad \left. - (\hat{k}_{i+1} - \hat{x}_j)(6\hat{k}_i - 3\hat{x}_{j+1} - 3\hat{x}_j) \right)
\end{aligned}$$

and $\hat{l}_i = \hat{k}_i$, $z_j = x_j$ and $\hat{z}_j = \hat{x}_j$ (resp. $\hat{l}_i = k_i$, $z_j = \hat{x}_j$, and $\hat{z}_j = x_j$).

The assumption that each data line range $[x_j, x_{j+1}]$ is "enclosed" by a prototype line range $[k_i, k_{i+1}]$ does not hold in the general case. But we can dynamically subdivide any data line that crosses several intervals I_i such that the resulting line segments do not overlap multiple prototype ranges. For further implementational issues (and for an implementation) see [7]. The cluster merging procedure [8] of the unsupervised HCCL algorithm can be applied to this modified algorithm, too. We call the resulting algorithm the fuzzy c-piecewise linear functions (FCPLF) algorithm.

Example

To demonstrate the FCPLF algorithm, we show its performance on an artificially generated data set. We have chosen three similar functions as shown in figure 2(a):

$$y = \sin(x) \cdot \cos(x^2) \quad (11)$$

$$y = \sin(x + 0.3) \cdot \cos(x^2 + 0.2) \quad (12)$$

$$y = \sin(x + 0.1) \cdot \cos(0.9 * x^2 + 0.5) \quad (13)$$

For each of the three functions we created 20 time series, each consisting of 200 samples; the x values of the samples differed from series to series. For each time series, we added noise to the summands within the (co)sine and to the resulting input value x and output value y (figure 2(b)). The time series were transformed into continuous, piecewise linear approximations using the unsupervised HCCL algorithm. Then, the FCPLF algorithm was applied to the resulting data set. Figures 2(c) - 2(d) show two out of three resulting models, together with the corresponding original function. The three models approximate the original functions pretty well, all the time series have been assigned to the correct model.

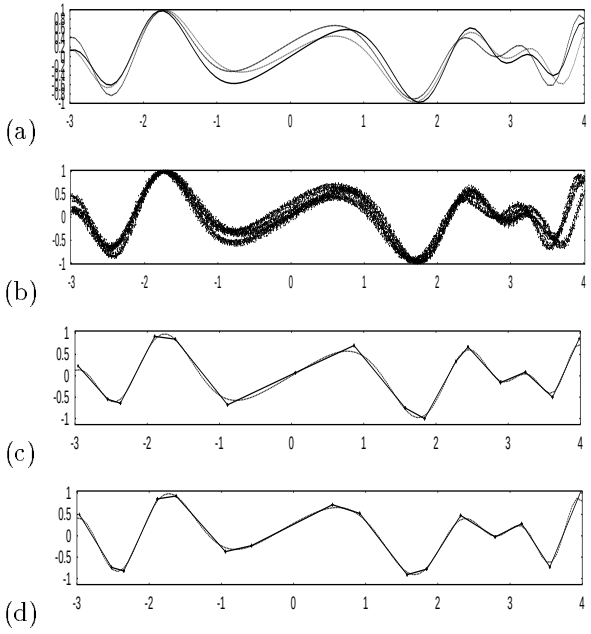

Figure 2: Clustering of noisy time series (11)-(13) using the FCPLF algorithm. From top to bottom: original functions, the 60 time series, the identified piecewise linear functions together with the original function.

5 Conclusions

In many applications a single *datum* consists of a *set of feature vectors*, for example with time series analysis. In certain cases one can take the set of features as a single big tuple of attribute values, but this is not possible if the features in the sets do not *correspond* to each other or some values are missing. In this paper we proposed a method to apply the well-known fuzzy clustering algorithms to sets of features in general, obtaining sets of prototypes as cluster representatives. This includes the case of time series analysis, but may be useful for other domains like shape clustering, too.

For the special case of single-input single-output time series we have modified the Hard c-Connected Lines algorithm to process sets of line segments instead of sets of features. Using line segments as "aggregated features" corresponds to a data compression which speeds up the clustering process. We have demonstrated that the resulting fuzzy c-piecewise linear functions algorithm is capable of identifying different models within the set of sampled functions.

References

[1] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, New York, 1981.

[2] R. N. Davé. Characterization and detection of noise in clustering. *Pattern Recognition Letters*, 12:657–664, Nov. 1991.

[3] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[4] J. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact, well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[5] I. Gath and A. B. Geva. Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):773–781, July 1989.

[6] D. E. Gustafson and W. C. Kessel. Fuzzy clustering with a fuzzy covariance matrix. In *Proc. of the IEEE Conference on Decision and Control*, pages 761–766, Jan. 1979.

[7] F. Höppner. Fuzzy clustering algorithms – a tool library. Open Source Project, `http://www.et-inf.fho-emden.de/~dmlab`.

[8] F. Höppner. Piecewise linear function approximation by alternating optimization. In *Proc. of the 8th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU)*, Madrid, Spain, July 2000.

[9] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis.* John Wiley & Sons, Chichester, England, 1999. `http://fuzzy.cs.uni-magdeburg.de/clusterbook`.

[10] F. Klawonn and R. Kruse. Constructing a fuzzy controller from data. *Fuzzy Sets and Systems*, 85:177–193, 1997.

[11] R. Krishnapuram and J. M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110, May 1993.