Adaptive Optimization of the Number of Clusters in Fuzzy Clustering

Jürgen Beringer
Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany

Eyke Hüllermeier
Philipps-Universität Marburg
Hans-Meerwein-Str., Marburg, Germany

Abstract

In this paper, we present a local, adaptive optimization scheme for adjusting the number of clusters in fuzzy C-means clustering. This method is especially motivated by online applications in which a potentially changing clustering structure must be maintained over time, though it turns out to be useful in the static case as well. As part of the method, we propose a new validity measure for fuzzy partitions which is a modification of the commonly used Xie-Beni index and overcomes some deficiencies thereof.

1 Introduction

Learning in online scenarios has been a focus of research in the data mining field in recent years. Corresponding algorithms must be able to learn in an incremental way and to adapt to a changing environment. For example, clustering algorithms should be able to properly react to changes of the underlying clustering structure and, in particular, to a changing number of clusters [1, 2]. Obviously, this hampers the application of standard clustering algorithms such as K-means and fuzzy C-means, as these algorithms assume the number of clusters to be a fixed (user-specified) parameter which is known in advance.

Several methods for determining an optimal number of clusters have been proposed in the literature [10, 8]. Assuming a lower bound K_{min} and upper bound K_{max} to be given, the simplest approach simply tries all potential numbers $K \in \{K_{min} \ldots K_{max}\}$: A clustering structure is derived for every K and evaluated in terms of a quality measure such as the well-known Xie-Beni index [11]. Finally, the result which appears to be optimal according to this measure is adopted.

This enumeration strategy as well as more sophisticated variants thereof are computationally quite complex, as they have to test every K independently. This drawback is further amplified by the problem of local optima, which is commonly encountered for clustering algorithms like fuzzy C-means: Running the algorithm twice will typically produce two different results. Thus, in order to evaluate a fixed cluster number K, it is not enough to apply the algorithm to the data once. Instead, it must be applied repeatedly, say, N times, in order to produce a reasonably good and confident approximation to the best quality that can be obtained for K. Consequently, the overall number of runs of the clustering algorithm amounts to $N \times (K_{max} - K_{min} + 1)$.

Needless to say, the above optimization strategy cannot be used in an online context where an optimal K has to be found repeatedly, and possibly under hard time constraints. In this paper, we propose a method for determining an optimal number of clusters in an *adaptive* and efficient way, focusing on fuzzy C-means (FCM) as an underlying clustering algorithm.

The remainder of the paper is organized as follows: In the next section, we outline the problem of clustering data streams as an interesting application and important motivation of adaptive clustering. Our method for optimizing the cluster number will be introduced in section 3. An empirical evaluation covering both the static case and the clustering of data streams is given in section 4.

2 Adaptive Clustering of Data Streams

In recent years, so-called *data streams* have attracted considerable attention in different fields of computer science, such as database systems, distributed systems and, in particular, data mining [5, 9, 7]. As the notion suggests, a data stream can roughly be thought of as an ordered sequence of data items, where the input arrives more or less continuously as time progresses. There are various applications in which streams of this type are produced, such as network monitoring, telecommunication systems, customer click streams, stock markets, or any type of multi-sensor system. As a data stream system may constantly produce huge amounts of data, it is usually not feasible to simply store the arriving data in order to analyze it offline later on. Instead, stream data must be processed in an online manner so as to guarantee that results are up-to-date.

In [2], the problem of continuously clustering a fixed number of parallel data streams is considered, with a focus on time-series data streams (i.e., individual data items are real numbers). More specifically, the problem is to maintain groups of data streams such that streams within one class are similar to each other in the sense of evolving similarly over time; as a suitable distance function, a weighted correlation measure on a sliding window was used (see Fig. 1). We refer to [2] for technical details.

The basic algorithm for clustering data streams is given in pseudo-code in algorithm 2.1. As can be seen, the algorithm performs FCM clustering of the streams on the current time window until time progresses (in a discrete step) and new data arrives. Then, the streams are updated by sliding the windows, and the pairwise distances between streams are re-computed (which can be done incrementally). From a clustering point of view, the challenge derives from the fact that the objects to be clustered, namely the data streams, do change in the course of time. Roughly, these objects can be imagined as "moving points" in a high-dimensional Euclidean space (the dimension corresponds to the length of the time window). Consequently, an *adaptive* clustering approach is needed in order to maintain an up-to-date clustering structure.

Algorithm 2.1: Adaptive clustering of data streams

```
  Input: streams of data points
1 Initialize K cluster centers at random;
2 while streams are still alive do
3     repeat
4         Assign membership degrees for each stream to the cluster centers;
5         Replace each center by the center of its associated fuzzy cluster;
6     until new data has arrived ;
7     Adjust the cluster number K by ±1;
8     Update data streams (by sliding windows), distances between streams, and
      cluster centers;
```

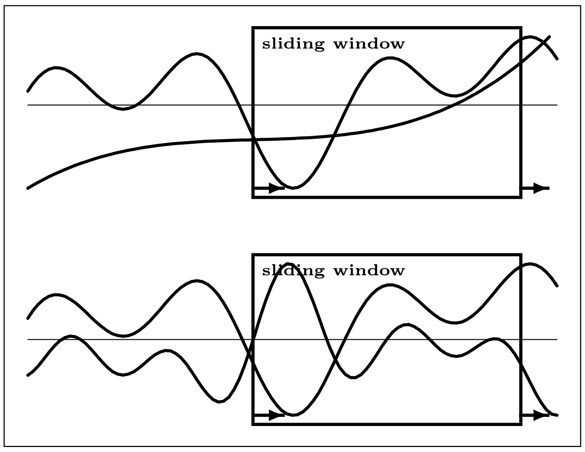

Figure 1: Data streams are compared within a sliding window of fixed size. In the example above, the behavior of the two streams is obviously quite different. In the example below, the two streams are similar to some extent.

3 Local Optimization of the Cluster Number

Recall that fuzzy C-means seeks to minimize the following objective function [3]:

$$\sum_{\imath=1}^{n}\sum_{\jmath=1}^{K} \|x_\imath - c_\jmath\|^2 \, (u_{\imath\jmath})^m, \tag{1}$$

where $u_{\imath\jmath} = u_\jmath(x_\imath)$ is the membership of the $\imath$-th object $x_\imath$ in the $\jmath$-th cluster, and $c_\imath$ is the $\jmath$-th center. In the commonly employed *probabilistic* version of fuzzy C-means, it is required that

$$\sum_{\jmath=1}^{K} u_{\imath\jmath} = \sum_{\jmath=1}^{K} u_\jmath(x_\imath) = 1 \tag{2}$$

for all $x_\imath$ [6]. The constant $m > 1$ in (1) is called the *fuzzifier* and controls the overlap ("smoothness") of the clusters (a common choice is $m = 2$).

As mentioned before, the simple enumeration strategy for optimizing the cluster number, as outlined in the introduction, is not practicable in an online setting as it requires the consideration of too large a number of candidate values and, hence, applications of the clustering algorithm. To minimize the effort, the idea of this paper is to pursue a *local* adaptation process that tries to adapt the cluster number K on the basis of a starting point K_0 in the style of a hill-climbing procedure. This strategy appears particularly appealing in an online setting where the optimal cluster number, K^*, may "smoothly" change in the course of time. In fact, assuming that K^* does not make big jumps, the optimal number at time t, $K^*(t)$, will provide a good initialization for finding the optimal number at time $t+1$. In other words, a local search is likely to succeed without getting trapped in local optima.

Thus, starting with $K = K_0$, each iteration of our method consists of a test that checks whether the cluster model can be improved by increasing or decreasing K. To this end, we make use of a suitable quality measure (validity function) $Q(\cdot)$. Let $Q(K)$ denote the quality of the cluster number K, that is, of the cluster model obtained for this number. In each iteration, K is then updated as follows:

$$K \leftarrow \arg\max \{Q(K-1), Q(K), Q(K+1)\}$$

This is repeated until K remains unchanged, i.e., $Q(K) > \max \{Q(K-1), Q(K+1)\}$. Essentially, this approach requires two elements: Firstly, a suitable validity function $Q(\cdot)$, and secondly, a means for going from a clustering structure with K clusters to structures with $K-1$ and $K+1$ clusters, respectively. We shall address the first issue in section 3.1 and the second one in section 3.2.

3.1 Fuzzy Validity Function

Regarding the evaluation of a cluster model (partition of the data) in terms of a measure $Q(\cdot)$, several proposals can be found in the literature. Unfortunately, most of these measures have been developed for the non-fuzzy case. Indeed, validity functions of that kind might still be (and in fact often are) employed, namely by mapping a fuzzy cluster model to a crisp one first (i.e., assigning each object to the cluster in which it has the highest degree of membership) and deriving the measure for this latter structure afterwards. However, this approach can of course be criticized as it comes along with a considerable loss of information. On the other hand, many of the non-fuzzy measures can be adapted to the fuzzy case in a natural way.

Validity functions typically suggest finding a trade-off between intra-cluster and inter-cluster variability, which is of course a reasonable principle. Besides, our approach gives rise to a number of additional requirements, notably the following:

(a) *Concavity:* Since the number K of clusters is only changed locally by ± 1, i.e., in the style of hill-climbing, our adaptation procedure might get stuck in local

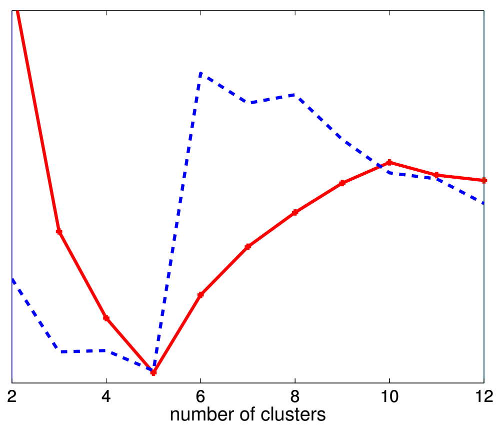

Figure 2: Example of the Xie-Beni validity measure (dashed line) and its modified version (5); the data was generated as described in section 4.1 (2 dimensions, 5 clusters).

optima. Consequently, the convexity (resp. concavity) of the validity function is highly desirable. That is, $Q(K)$ should be maximal (resp. minimal) for the optimal number K^* of clusters and decrease (resp. increase) in a monotone way for smaller and larger values (at least within a certain range around K^*). Unfortunately, most existing measures do not have this property and instead show a rather irregular behavior (see Fig. 2 for a typical example).

(b) *FCM-conformity:* As already explained above, to adapt the cluster number K, we provisionally consider two alternative structures that we obtain, respectively, by removing and adding a cluster. However, as will become clear later on, both candidate structures might not be fully optimized with regard to the FCM objective function (1). Instead, in an online setting, this optimization might take place only *after* the apparently best structure (cluster number) has been selected. In order to avoid this optimization to invalidate the previous selection, the validity measure $Q(\cdot)$ should well harmonize with the objective function (1).

(c) *Efficiency:* Since the validity function is frequently evaluated in our application, its computation should be efficient. This disqualifies measures with a quadratic complexity, such as the maximal distance between two objects within a cluster.

A widely used validity function is the so-called Xie-Beni index or separation [11], which is defined as

$$\frac{\frac{1}{n}\sum_{\imath=1}^{n}\sum_{k=1}^{K} u_{\imath k}^{m}\,\|x_\imath - c_k\|^2}{\min_{k,\ell}\|c_k - c_\ell\|^2}. \tag{3}$$

As most validity measures do, (3) puts the intra-cluster variability (numerator) in relation to the inter-cluster variability (denominator). In this case, the latter is simply determined by the minimal distance between two cluster centers. Obviously, the smaller the separation, the better the cluster model.

Since the nominator of (3) just corresponds to the objective function (1), the Xie-Beni index looks quite appealing with regard to point (b) above. Moreover, it is also efficient from a computational point of view. Still, point (a) remains problematic, mainly due to the minimum in the denominator.

To remedy this problem, we suggest replacing the minimum by a summation over all (pairwise) cluster dissimilarities, with smaller dissimilarities having a higher weight than larger ones. Simply defining the dissimilarity between two clusters by the distance between the corresponding centers is critical, however, since it neglects the variability (size) of these clusters. Therefore, we define the variability of a cluster in terms of the average (squared) distance from the center,

$$V_k \overset{\text{df}}{=} \frac{\sum_\imath u_{\imath k}\|x_\imath - c_k\|^2}{\sum_\imath u_{\imath k}}$$

and the dissimilarity between two clusters as

$$D(C_k, C_\ell) \overset{\text{df}}{=} \frac{\|c_k - c_\ell\|^2}{V_k \times V_\ell}.$$

These dissimilarities are aggregated by means of

$$\sum_{1\le k<\ell\le K} \frac{1}{D(C_k, C_\ell)}, \tag{4}$$

thereby putting higher weight on smaller dissimilarities. Replacing the denominator in (3) by (4), we thus obtain

$$\frac{1}{n}\sum_{\imath=1}^{n}\sum_{k=1}^{K}\|x_\imath - c_k\|^2\,u_{\imath k}^{m} \times \sum_{1\le k<\ell\le K}\frac{1}{D(C_k, C_\ell)}. \tag{5}$$

It is of course not possible to prove the concavity of (5) in a formal way. Still, our practical experience so far has shown that it satisfies our requirements in this regard very well and compares favorably with alternative measures ((see Fig. 2 for a typical example). Corresponding experimental results will be presented in section 4 below.

Algorithm 3.1: Optimization of the cluster number

```
Input: data, initial cluster number $K$
Output: optimal cluster number with associated partition
Data: partition $P$
1  initialize $P$ by $K$ cluster centers selected at random;
2  count(·) $\leftarrow 0$ ;
3  best $\leftarrow P$ ;
4  repeat
5      repeat
6          for every object, derive the membership degrees in the clusters in $P$;
7          derive the centers of the new clusters in $P$;
8      until $\Delta J < \varepsilon$ ;
9      find the best $K$-1 partition $\rightarrow P_{-1}$;
10     find the best $K$+1 partition $\rightarrow P_{+1}$;
11     $P \leftarrow$ best partition among $\{P, P_{-1}, P_{+1}\}$;
12     $K \leftarrow$ number of clusters in $P$;
13     count($K$) $\leftarrow$ count($K$)+1;
14     if $P$ better than best then best $\leftarrow P$;
15 until $K$ unchanged or count($K$)>$K$ ;
16 return best;
```

3.2 Adapting the Clustering Structure

The second element of our adaptive optimization scheme is a means for replacing the current clustering structure, consisting of K clusters, by structures with $K-1$ and $K+1$ clusters, respectively. Intuitively, decreasing the cluster number by 1 means that one of the current clusters has disappeared. Thus, we propose to derive a cluster model with $K-1$ clusters as follows: One of the current clusters is tentatively removed, and M iterations of the fuzzy C-means algorithm are executed in order to re-assign the elements of this cluster to the remaining cluster centers (and to adjust these centers correspondingly). The quality of the cluster model thus obtained is then computed. This is repeated K times, i.e., each of the current clusters is removed by way of trial. The best cluster model is then chosen, i.e., $Q(K-1)$ is defined by the quality of the best model (see Algorithm 3.2).

Algorithm 3.2: Optimal reduction of cluster size

```
Input: $T$ is partition with $K$ clusters
Output: $N$ is partition with $K-1$ clusters
1 foreach Cluster $c$ do
2     $S_c$=$T$ without cluster c;
3     for $i = 1$ to $M$ do
4         for every object, derive the membership degrees in the $K-1$ clusters in $S_c$;
5         derive the centers of the new clusters in $S_c$;
6     compute the validity measure for $S_c$;
7 set $N$ = best partition according to validity measure $S_c$;
```

Going from K to $K+1$ assumes that an additional cluster has emerged. To create this cluster we complement the existing K centers by one center that is defined by a randomly chosen element. The probability of an element to be selected is reasonably defined as an increasing function of the elements's distance from its associated cluster center. More specifically, we define the probability of an element x_i in proportion to $\sum_{k=1}^{K} ||x_i - c_k||(u_{ik})^m$. In order to compute $Q(K+1)$, we try out a fixed number of randomly chosen elements and select the one that gives the best cluster model.

4 Empirical Evaluation

Our evaluation consists of four experimental studies. First, we investigated the suitability of our approach for determining the optimal cluster number in the case of static data. The second study is concerned with discovering a sudden change in the number of clusters. The last two experiments evaluate our adaptive clustering scheme in the streaming environment.

4.1 Experiment 1

In the static case we compared our local optimization approach with the simple strategy which tries every cluster number in the range $\{2 \ldots 19\}$. In the latter case, we called the fuzzy C-means algorithm 10 times for each K in order to avoid local minima and then took the best result [4].

As test data, we used randomly generated synthetic data with rather simple clustering structures, each consisting of 1,000 data points located in the d-dimensional space $[0, 200]^d$. To generate a data set, we first determine the cluster number K at random, using a uniform distribution over $\{2 \ldots 19\}$. The cluster centers are generated analogously (a distribution of the centers in $[0, 200]^d$ is accepted only if the distance between every pair of centers is at least 6 times as large as the standard deviation of the clusters). For every cluster, $1000/K$ data points are generated using a normal distribution with standard deviation 5.

50 data sets have been generated for each dimension $d \in \{2, 5, 10, 25\}$ and each cluster number K. Our local approach is initialized with $K_0 = 10$ while the global strategy searches the complete range $\{2 \ldots 19\}$. To obtain a reference measure, we initialized the standard fuzzy C-means algorithm with the correct cluster centers and derived the validity index for the clustering structure thus obtained. For both approaches (local and global) we then checked whether the respective solution is at least as good as the reference result and derived a corresponding hit rate. Additionally, we measured the runtimes of both approaches.

Regarding the hit rate, it was to be expected that the global strategy outperforms the local one, as it tries all potential cluster numbers. However, as can be seen in Fig. 3, our local strategy is quite competitive and in some cases even superior. As a potential explanation, we suspect that our strategy of adding and removing clusters

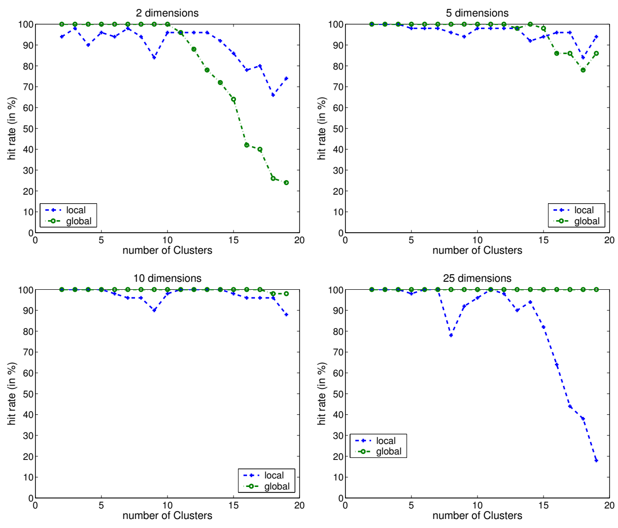

Figure 3: Comparison of the hit rates

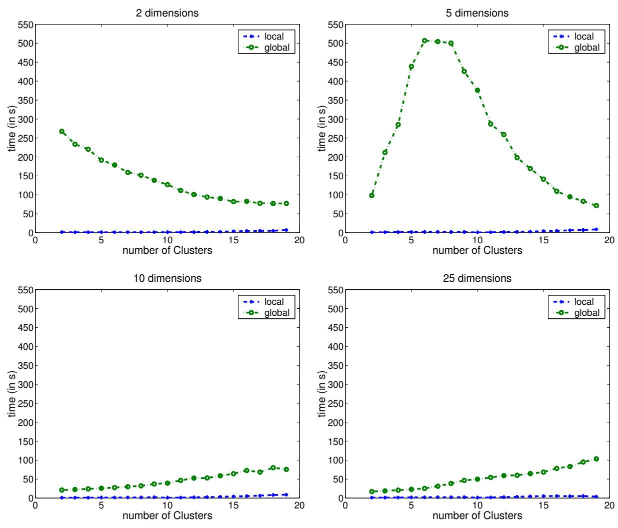

Figure 4: Comparison of the runtimes

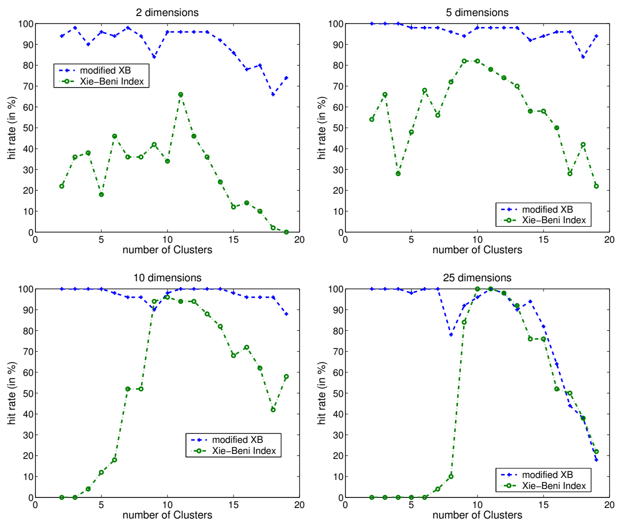

Figure 5: Comparison of the validity measure

in a systematic way is more effective in escaping local minima of the fuzzy C-means objective function than restarting the algorithm at random. On the other hand, the local approach obviously runs into trouble if the true cluster number is much higher than the initialization, especially in the case of high-dimensional data.

With regard to the runtimes, our local approach clearly outperforms the global one (see Fig. 4). In fact, even though the runtimes slightly increase with the number of clusters, they remain on a comparably low level for the local approach, independently of the dimension. In contrast, the runtimes of the global method are much higher.

Fig. 5 shows a comparison with the Xie-Beni index. As expected, the results strongly depend on the initialization: the closer the true cluster number to the initialization $K_0 = 10$, the higher the hit rate becomes. More importantly, however, the figure clearly shows the positive influence of our modified version of the Xie-Beni index. In fact, using this measure significantly improves the hit rate.

4.2 Experiment 2

In this experiment, we tested our method's ability to discover the emergence of a new or disappearance of an existing cluster. To this end, we generated coupled data sets as follows: First, a single data set with $K \in \{2 \ldots 18\}$ is generated as described above.

Table 1: Detection rate

dimension	$K \to K-1$	$K \to K+1$	rate
2	100%	98.7%	99.3%
5	99.9%	98.9%	99.4%
10	100%	99.9%	100%
25	100%	99.9%	100%

Then, two variants of this data set are produced, one by generating additional data for the last cluster, and the other one by generating data for an additional (random) cluster center. Thus, the second variant contains $K+1$ clusters, one more than the first variant.

To test the ability of detecting the increase of the cluster number, the first variant is clustered by initializing fuzzy C-means with the correct cluster centers. Then, we replaced the data set by the second variant and checked whether or not the additional cluster was discovered by the adaptation step outlined in section 3.2. To test the ability of detecting the decrease of the cluster number, the two variants of the data set are simply exchanged.

Table 1 shows average detection rates for 5,000 tests. As can be seen, the rates are close to 100%. As expected, the removal of a cluster is even somewhat easier to detect than the emergence of an additional cluster. In fact, in the latter case we only tried 10 data points as additional cluster centers, which makes it is quite likely that no improvement can be achieved. As in the first experimental study, the results seem to be even better for higher dimensions. This can be explained by the fact that local minima of the FCM objective function occur more frequently in the low-dimensional case.

4.3 Experiment 3

In the third experiment, we investigated the ability of our algorithm to adapt to a changing number of clusters in the data stream environment outlined in section 2. To this end, synthetic data streams were generated in the follow way: First, a *prototype* $p(\cdot)$ is predefined in terms of a specific (deterministic) function of time. The elements that (should) belong to the cluster are then generated by "distorting" the prototype, both horizontally (by stretching the time axis) and vertically (by adding noise). More precisely, a data stream $x(\cdot)$ is defined by

$$x(t) \;=\; p\left(t + h(t)\right) + g(t),$$

where $h(\cdot)$ and $g(\cdot)$ are stochastic processes that are generated by means of a second-order difference equation:

$$\begin{aligned} h(t+\Delta t) \;&=\; h(t) + h'(t+\Delta t) \qquad (6)\\ h'(t+\Delta t) \;&=\; h'(t) + u(t), \end{aligned}$$

$t = 0, \Delta t, 2\Delta t \ldots$ The $u(t)$ are independent random variables, uniformly distributed in an interval $[-a, a]$; obviously, the smaller the constant a is, the smoother the stochastic process will be.

In the experiment, we varied the number of artificial clusters in the data generating process: Starting with two clusters, the number of clusters was repeatedly doubled to four (in a "smooth" way) and later again reduced to two. Technically, this was accomplished as follows: The overall number of 100 data streams is divided into four groups. The first and second group are represented by the prototype $p_1(t) = \sin(t)$ and $p_2(t) = 1 - \sin(t)$, respectively. The third group is characterized by the prototype $p_3(t) = (1 - \lambda)p_1(t) + \lambda \sin(t + \pi/2)$, where $\lambda \in [0, 1]$ is a parameter. Likewise, the fourth group is characterized by the prototype $p_4(t) = (1-\lambda)p_2(t)+\lambda(1-\sin(t+\pi/2))$. As explained above, all streams were generated as distortions of their corresponding prototypes.

As can be seen, for $\lambda = 0$, the third (fourth) and the first (second) group form a single cluster, whereas the former moves away from the latter for larger values of λ, and finally constitutes a completely distinct cluster for $\lambda = 1$. The parameter λ is changed from 0 to 1 and back from 1 to 0 in a smooth way within a range of 8 time steps (during one time step 512 new data points are generated).

Fig. 6 shows the value of λ and the number of clusters generated by the algorithm as a function of time, that is, for each block number. As can be seen, our approach correctly adapts the number of clusters, but of course with a small delay. We obtained qualitatively very similar results with other numbers of clusters and other data generating processes.

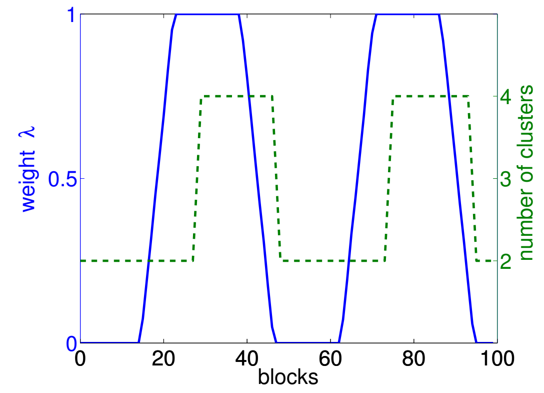

Figure 6: Weight of the parameter λ (solid line) and number of clusters (dashed line) in the third experiment.

4.4 Experiment 4

This experiment is quite similar to the recent one. This time, we simulated a scenario in which some data streams move between two clusters. Again, these two clusters are represented, respectively, by the prototypes $p_1(t) = \sin(t)$ and $p_2(t) = 1 - \sin(t)$. Additionally, there are two streams that are generated as distortions of the convex combination $(1 - \lambda)p_1 + \lambda p_2$, where $\lambda \in [0, 1]$.

Fig. 7 shows the value of λ and the (average) membership degrees of the two streams in the second cluster and the intermediate cluster. As can be seen, the membership degrees are again correctly adapted with a small delay of time. The algorithm creates an additional cluster in-between. This cluster suddenly emerges when the streams are relatively far away from the first cluster and disappears when they come close enough to the second cluster. The degree of membership in the intermediate cluster, again averaged over the moving elements, is shown by the additional solid line in the figure.

5 Conclusions

This paper has introduced a local, adaptive optimization scheme for adjusting the number of clusters in fuzzy C-means clustering. Even though this method is especially motivated by online scenarios in which a potentially changing clustering structure must

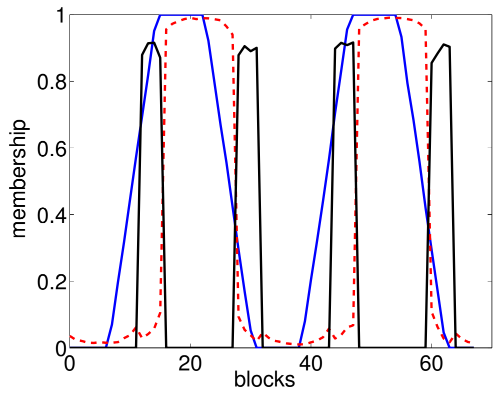

Figure 7: Weight of the parameter λ and degree of membership of the moving streams in the second cluster (dashed line) and the intermediate cluster (solid line).

be maintained over time, our experiments have shown that it is likewise useful for analyzing static data sets.

References

[1] Jürgen Beringer and Eyke Hüllermeier. Online clustering of parallel data streams. *Data Knowl. Eng.*, 58(2):180–204, 2006.

[2] Jürgen Beringer and Eyke Hüllermeier. Fuzzy clustering of parallel data streams. In Jose Valente de Oliveira and Witold Pedrycz, editors, *Advances in Fuzzy Clustering and Its Applications*. John Wiley and Sons, 2007.

[3] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[4] Paul S. Bradley and Usama M. Fayyad. Refining initial points for k-means clustering. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 91–99, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[5] P. Domingos and G. Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.

[6] R. Kruse F. Höppner, F. Klawonn and T. Runkler. *Fuzzy Cluster Analysis*. Wiley, 1999.

[7] L. Golab and M.T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[8] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *NIPS*, 2003.

[9] J. Gehrke M. Garofalakis and R. Rastogi. Querying and mining data streams: you only get one look a tutorial. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 635–635, New York, NY, USA, 2002. ACM Press.

[10] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a dataset via the gap statistic. *JRSSB*, 2000.

[11] Xuanli Lisa Xie and Gerardo Beni. A validity measure for fuzzy clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(8):841–847, 1991.