Fuzzy Clustering of Parallel Data Streams

Jürgen Beringer and Eyke Hüllermeier
Fakultät für Informatik
Otto-von-Guericke-Universität
Magdeburg, Germany
{juergen.beringer,eyke.huellermeier}@iti.cs.uni-magdeburg.de

Abstract

The management and processing of so-called data streams has recently become a topic of active research in several fields of computer science, notably database systems and data mining. A data stream can roughly be thought of as a transient, continuously increasing sequence of time-stamped data. In this paper, we consider the problem of clustering parallel streams of real-valued data, that is to say, continuously evolving time series. More specifically, we are interested in grouping data streams the evolution over time of which is similar in a specific sense. In order to maintain an up-to-date clustering structure, it is necessary to analyze the incoming data in an online manner, tolerating not more than a constant time delay. For this purpose, we develop an efficient online version of the fuzzy C-means clustering algorithm. A fuzzy approach appears to be particularly useful for this type of application, in which the clustering structure is subject to continuous changes.
Keywords: clustering, fuzzy sets, K-means, fuzzy C-means, data streams, Fourier transformation

1 Introduction

In recent years, so-called *data streams* have attracted considerable attention in different fields of computer science such as, e.g., database systems, data mining, or distributed systems. 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 [16, 13, 6]. There are various applications in which streams of this type are produced such as, e.g., network monitoring, telecommunication systems, customer click streams, stock markets, or any type of multi-sensor system.

A data stream system may constantly produce huge amounts of data. To illustrate, imagine a multi-sensor system with 10,000 sensors each of which sends a measurement every second of time. Regarding aspects of data storage, management and processing, the continuous arrival of data items in multiple, rapid, time-varying, and potentially unbounded streams raises

new challenges and research problems. Indeed, it is usually not feasible to simply store the arriving data in a traditional database management system in order to perform operations on that data later on. Rather, stream data must generally be processed in an online manner in order to guarantee that results are up-to-date and that queries can be answered with small time delay. The development of corresponding *stream processing systems* is a topic of active research [3].

In this paper, we consider the problem of clustering data streams. Clustering is one of the most important and frequently used data analysis techniques. It refers to the grouping of objects into homogeneous classes or groups and is commonly seen as a tool for discovering structure in data. In our context, the goal is to maintain classes of data streams such that streams within one class are similar to each other in a sense to be specified below. Roughly speaking, we assume a large number of evolving data streams to be given, and we are looking for groups of data streams that evolve similarly over time. Our focus is on time-series data streams, which means that individual data items are real numbers that can be thought of as a kind of measurement. There are numerous applications for this type of data analysis such as e.g. clustering of stock rates.

Apart from its practical relevance, this problem is also interesting from a methodological point of view. Especially, the aspect of efficiency plays an important role: Firstly, data streams are complex, extremely high-dimensional objects making the computation of similarity measures costly. Secondly, clustering algorithms for data streams should be adaptive in the sense that up-to-date clusters are offered at any time, taking new data items into consideration as soon as they arrive. In this paper, we develop techniques for clustering data streams that meet these requirements. More specifically, we develop an efficient online version of the fuzzy C-means clustering algorithm. The efficiency of our approach is mainly due to a scalable online transformation of the original data which allows for a fast computation of approximate distances between streams.

The remainder of the paper is organized as follows: Section 2 provides some background information, both on data streams and on clustering. The maintenance and adequate preprocessing of data streams is addressed in Section 3. Section 4 is given to the clustering of data streams and introduces an online version of the fuzzy C-means algorithm. Section 5 discusses quality and distance measures for (fuzzy) cluster models appropriate for the streaming setting. Finally, experimental results are presented in Section 6.

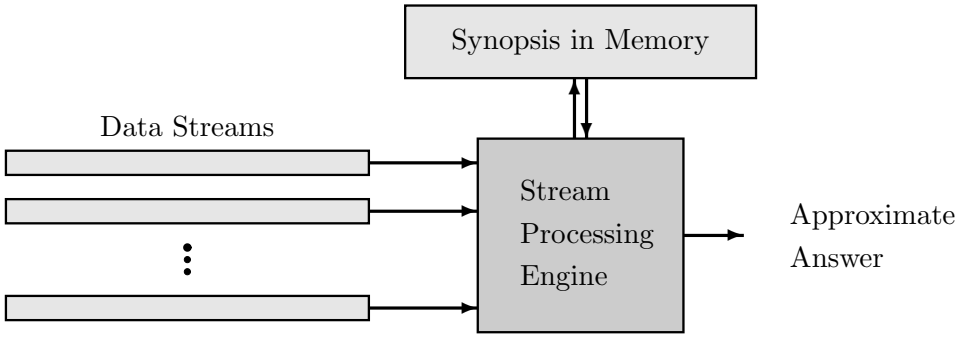

Figure 1: Basic structure of a data stream model.

2 Background

2.1 The Data Stream Model

The *data stream model* assumes that input data are not available for random access from disk or memory, such as relations in standard relational databases, but rather arrive in the form of one or more continuous data streams. The stream model differs from the standard relational model in the following ways [1]:

- The elements of a stream arrive incrementally in an "online" manner. That is, the stream is "active" in the sense that the incoming items trigger operations on the data rather than being sent on request.
- The order in which elements of a stream arrive are not under the control of the system.
- Data streams are potentially of unbounded size.
- Data stream elements that have been processed are either discarded or archived. They cannot be retrieved easily unless being stored in memory, which is typically small relative to the size of the stream. (Stored/condensed information about past data is often referred to as a *synopsis*, see Fig. 1).
- Due to limited resources (memory) and strict time constraints, the computation of exact results will usually not be possible. Therefore, the processing of stream data does commonly produce *approximate* results [4].

2.2 Clustering

Clustering refers to the process of grouping a collection of objects into classes or "clusters" such that objects within the same class are *similar* in a certain sense, and objects from

different classes are dissimilar. Clustering algorithms proceed from given information about the similarity between objects, e.g., in the form of a *proximity matrix*. Usually, objects are described in terms of a set of measurements from which similarity degrees between pairs of objects are derived, using a kind of similarity or distance measure.

One of the most popular clustering methods is the so-called K-means algorithm [19]. This algorithm starts by guessing K cluster centers and then iterates the following steps until convergence is achieved:

- clusters are built by assigning each element to the closest cluster center;
- each cluster center is replaced by the mean of the elements belonging to that cluster.

K-means usually assumes that objects are described in terms of quantitative attributes, i.e., that an object is a vector $x \in \Re^n$. Dissimilarity between objects is defined by the Euclidean distance, and the above procedure actually implements an iterative descent method that seeks to minimize the variance measure ("within cluster" point scatter)

$$\sum_{k=1}^{K} \sum_{x_\imath, x_\jmath \in C_k} \|x_\imath - x_\jmath\|^2, \tag{1}$$

where C_k is the k-th cluster. In each iteration, the criterion (1) is indeed improved, which means that convergence is assured. Still, it is not guaranteed that the global minimum will be found, i.e., the final result may represent a suboptimal local minimum of (1).

Fuzzy (C-means) clustering is a generalization of standard (K-means) clustering that has proved to be useful in many practical applications.[1] In standard clustering, each object is assigned to one cluster in an unequivocal way. As opposed to this, in fuzzy clustering an object x may belong to different clusters at the same time, and the degree to which it belongs to the $\imath$-th cluster is expressed in terms of a *membership degree* $u_\imath(x)$. Consequently, the boundary of single clusters and the transition between different clusters are usually "smooth" rather than abrupt.

The fuzzy variant of K-means clustering seeks to minimize the following objective function [2]:

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

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 (FCM), it is

[1] We adhere to the common practice of using the term fuzzy C-means (FCM) instead of fuzzy K-means. However, for reasons of coherence, we will go on denoting the number of clusters by K, also in the fuzzy case.

assumed that

$$\sum_{j=1}^{K} u_{ij} = \sum_{j=1}^{K} u_j(x_i) = 1 \tag{3}$$

for all x_i [20]. The constant $m > 1$ in (2) is called the *fuzzifier* and controls the overlap ("smoothness") of the clusters (a common choice is $m = 2$).

Minimizing (2) subject to (3) defines a constrained optimization problem.[2] The clustering algorithm approximates an optimal (or at least locally optimal) solution by means of an iterative scheme that alternates between recomputing the optimal centers according to

$$c_j = \frac{\sum_{i=1}^{n} x_i \, (u_{ij})^m}{\sum_{i=1}^{n} (u_{ij})^m}$$

and membership degrees according to

$$u_{ij} = \left(\sum_{\ell=1}^{K} \left(\frac{\|x_i - c_j\|}{\|x_i - c_\ell\|} \right)^{2/(m-1)} \right)^{-1}.$$

2.3 Related Work

Stream data mining [10] is a topic of active research, and several adaptations of standard statistical and data analysis methods to data streams or related models have been developed recently (e.g. [7, 28]). Likewise, several online data mining methods have been proposed (e.g. [21, 8, 25, 5, 15, 12]). In particular, the problem of clustering in connection with data streams has been considered in [17, 9, 24]. In these works, however, the problem is to cluster the elements of one individual data stream, which is clearly different from our problem, where the objects to be clustered are the streams themselves rather than single data items thereof. To the best of our knowledge, the problem in this form has not been addressed in the literature so far.

There is a bunch of work on time series data mining in general and on clustering time series in particular [23]. Even though time series data mining is of course related to stream data mining, one should not overlook important differences between these fields. Particularly, time series are still static objects that can be analyzed offline, whereas the focus in the context of data streams is on dynamic adaptation and online data mining.

3 Preprocessing and Maintaining Data Streams

The first question in connection with the clustering of (active) data streams concerns the concept of distance or, alternatively, similarity between streams. What does similarity of two streams mean, and why should they, hence, fall into one cluster?

[2] As most clustering problems, the standard K-means problem is known to be NP-hard (see e.g. [11]).

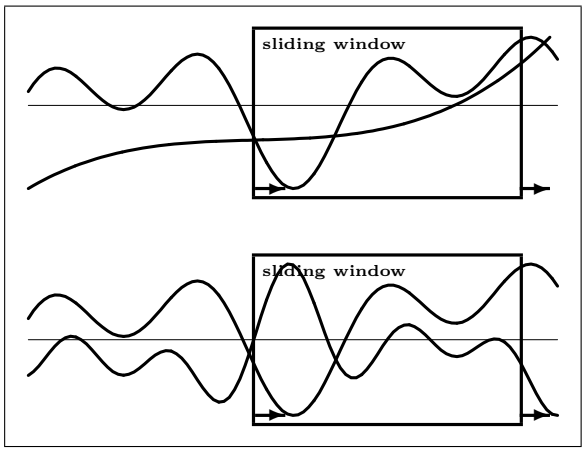

Figure 2: Data streams are compared within a sliding window of fixed size. In the example above, the behavior of the two streams is obviously quite different. In the example below, the two streams are similar to some extent.

Here, we are first of all interested in the qualitative, time-dependent evolution of a data stream. That is to say, two streams are considered similar if their evolution over time shows similar characteristics. As an example consider two stock rates both of which continuously increased between 9:00 a.m. and 10:30 a.m. but then started to decrease until 11:30 a.m.

To capture this type of similarity, we shall simply derive the Euclidean distance between the normalization of two streams (a more precise definition follows below). This measure satisfies our demands since it is closely related to the (statistical) *correlation* between these streams. In fact, there is a simple linear relationship between the correlation of normalized time series (with mean 0 and variance 1) and their (squared) Euclidean distance. There are of course other reasonable measures of similarity for data streams or, more specifically, time series [18], but Euclidean distance has desirable properties and is commonly used in applications.

3.1 Data Streams and Sliding Windows

The above example (clustering of stock rates) already suggests that one will usually not be interested in the entire data streams, which are potentially of unbounded length. Instead, it is reasonable to assume that recent observations are more important than past data. Therefore, one often concentrates on a *time window*, that is a subsequence of a complete data stream. The most common type of window is a so-called *sliding window* that is of fixed length and comprises the w most recent observations (cf. Fig.2). A more general approach to taking the relevancy of observations into account is that of *weighing*. Here, the idea is to associate a weight in the form of a real number to each observation such that more recent observations receive higher weights.

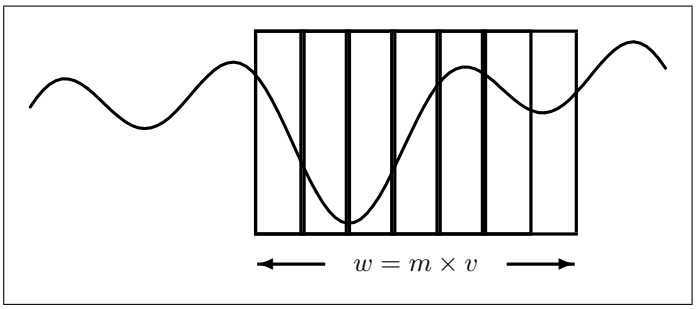

Figure 3: A window of length w is divided into m blocks of size v.

When considering data streams in a sliding window of length w, a stream (resp. the relevant part thereof) can formally be written as a w-dimensional vector $X = (x_0, x_1, \ldots, x_{w-1})$, where a single observation x_i is simply a real number. As shown in Fig. 3, we further partition a window into m blocks (basic windows) of size v, which means that $w = m \cdot v$ (Table 1 provides a summary of notation):[3]

$$X = (\underbrace{x_0, x_1, \ldots, x_{v-1}}_{B_1} \mid \underbrace{x_v, x_{v+1}, \ldots, x_{2v-1}}_{B_2} \mid \ldots \mid \underbrace{x_{(m-1)v}, x_{(m-1)v+1}, \ldots, x_{w-1}}_{B_m})$$

Data streams will then be updated in a "block-wise" manner each time v new items have been observed. This approach gains efficiency since the number of necessary updates is reduced by a factor of v. On the other hand, we tolerate the fact that the clustering structure is not always up-to-date. However, since the delay is at most one block size, this disadvantage is limited at least for small enough blocks. Apart from that, one should note that a small number of observations can change a stream but slightly, hence the clustering structure in the "data stream space" will usually not change abruptly.

We assume data items to arrive synchronously, which means that all streams will be updated simultaneously. An update of the stream X, in this connection also referred to as X^{old}, is then accomplished by the following shift operation, in which B_{m+1} denotes the *entering* block:

$$\begin{aligned} X^{old}: &\quad B_1 \mid B_2 \mid B_3 \mid \ldots \mid B_{m-1} \mid B_m \\ X^{new}: &\quad\quad B_2 \mid B_3 \mid \ldots \mid B_{m-1} \mid B_m \mid B_{m+1} \end{aligned} \tag{4}$$

Finally, we allow for an exponential weighing of observations (within a window). The weight attached to observation x_i is defined by c^{w-i-1}, where $0 < c \leq 1$ is a constant. We denote by V the weight vector $(c^{w-1}, c^{w-2} \ldots c^0)$ and by $V \odot X$ the coordinate-wise product of V and a stream X:

$$V \odot X \stackrel{\text{df}}{=} (c^{w-1}x_0, c^{w-2}x_1 \ldots c^0 x_{w-1}).$$

[3]Typical values as used in our experiments later on are $w = 2048$ and $v = 128$.

symbol	meaning
X	data stream
X^n	normalized data stream
$x_\imath$	single observation
$x_{\imath\jmath}$	$(\jmath+1)$-th element of block $B_\imath$
w	window length
v	length of a block
m	number of blocks in a window
c	weighing constant
V	weight vector
$\bar{x}$	mean value of a stream
s	standard deviation of a stream

Table 1: Notation

3.2 Normalization

Since we are interested in the *relative* behavior of a data stream, the original streams have to be normalized in a first step. By normalization one usually means a linear transformation of the original data such that the transformed data has mean 0 and standard deviation 1. The corresponding transformation simply consists of subtracting the original mean and dividing the result by the standard deviation. Thus, we replace each value $x_\imath$ of a stream X by its normalization

$$x_\imath^n \stackrel{\text{df}}{=} \frac{x_\imath - \bar{x}}{s}. \tag{5}$$

Considering in addition the weighing of data streams, $\bar{x}$ and s become the *weighted* average and standard deviation, respectively:

$$\bar{x} = \frac{1-c}{1-c^w} \cdot \sum_{\imath=0}^{w-1} x_\imath \cdot c^{w-\imath-1},$$

$$\begin{aligned} s^2 &= \frac{1-c}{1-c^w} \cdot \sum_{\imath=0}^{w-1} (x_\imath - \bar{x})^2 \cdot c^{w-\imath-1} \\ &= \frac{1-c}{1-c^w} \cdot \sum_{\imath=0}^{w-1} (x_\imath)^2 \cdot c^{w-\imath-1} - (\bar{x})^2. \end{aligned}$$

As suggested above, $\bar{x}$ and s^2 are updated in a block-wise manner. Let X be a stream and denote by $x_{\imath\jmath}$ the $(\jmath+1)$-th element of the $\imath$-th block $B_\imath$. Particularly, the exiting block leaving the current window ("to the left") is given by the first block $B_1 = (x_{10}, x_{11}, \ldots, x_{1,v-1})$. Moreover, the new block entering the window ("from the right") is $B_{m+1} = (x_{m+1,0}, x_{m+1,1}, \ldots, x_{m+1,v-1})$.

We maintain the following quantities for the stream X:

$$Q_1 \stackrel{\text{df}}{=} \sum_{\imath=0}^{w-1} x_\imath \cdot c^{w-\imath-1}, \quad Q_2 \stackrel{\text{df}}{=} \sum_{\imath=0}^{w-1} (x_\imath)^2 \cdot c^{w-\imath-1}.$$

Likewise, we maintain for each block B_k the variables

$$Q_1^k \stackrel{\text{df}}{=} \sum_{\imath=0}^{v-1} x_{k\imath} \cdot c^{v-\imath-1}, \quad Q_2^k \stackrel{\text{df}}{=} \sum_{\imath=0}^{v-1} (x_{k\imath})^2 \cdot c^{v-\imath-1}.$$

An update via shifting the current window by one block is then accomplished by setting

$$\begin{aligned} Q_1 &\leftarrow Q_1 \cdot c^v - Q_1^1 \cdot c^w + Q_1^{m+1}, \\ Q_2 &\leftarrow Q_2 \cdot c^v - Q_1^2 \cdot c^w + Q_2^{m+1}. \end{aligned}$$

3.3 Discrete Fourier Transform

Another preprocessing step replaces the original data by its Discrete Fourier Transform (DFT). As will be explained in more detail in section 3.5, this provides a suitable basis for an efficient approximation of the distance between data streams and, moreover, allows for the elimination of noise. The DFT of a sequence $X = (x_0, \ldots, x_{w-1})$ is defined by the DFT coefficients

$$\text{DFT}_f(X) \stackrel{\text{df}}{=} \frac{1}{\sqrt{w}} \sum_{\jmath=0}^{w-1} x_\jmath \cdot \exp\left(\frac{-i2\pi f \jmath}{w}\right), \qquad f = 0, 1, \ldots, w-1,$$

where $i = \sqrt{-1}$ is the imaginary unit.

Denote by X^n the normalized stream X defined through values (5). Moreover, denote by V the weight vector $(c^{w-1}, \ldots, c^0)$ and recall that $V \odot X^n$ is the coordinate-wise product of V and X^n, i.e., the sequence of values

$$c^{w-\imath-1} \times \frac{x_\imath - \bar{x}}{s}.$$

Since the DFT is a linear transformation, which means that

$$\text{DFT}(\alpha X + \beta Y) = \alpha\,\text{DFT}(X) + \beta\,\text{DFT}(Y)$$

for all $\alpha, \beta \geq 0$ and sequences X, Y, the DFT of $V \odot X^n$ is given by

$$\begin{aligned} \text{DFT}(V \odot X^n) &= \text{DFT}\left(V \odot \frac{X - \bar{x}}{s}\right) \\ &= \text{DFT}\left(\frac{V \odot X - V \odot \bar{x}}{s}\right) \\ &= \frac{\text{DFT}(V \odot X) - \text{DFT}(V) \odot \bar{x}}{s}. \end{aligned}$$

Since $\text{DFT}(V)$ can be computed in a preprocessing step, an incremental derivation is only necessary for $\text{DFT}(V \odot X)$.

3.4 Computation of DFT Coefficients

Recall that the exiting and entering blocks are $B_1 = (x_{10}, x_{11}, \ldots, x_{1,v-1})$ and $B_{m+1} = (x_{m+1,0}, x_{m+1,1}, \ldots, x_{m+1,v-1})$, respectively. Denote by $X^{old} = B_1|B_2|\ldots|B_m$ the current and by $X^{new} = B_2|B_3|\ldots|B_{m+1}$ the new data stream. Without taking weights into account, the DFT coefficients are updated as follows [28]:

$$\mathrm{DFT}_f \leftarrow e^{\frac{i2\pi fv}{w}} \cdot \mathrm{DFT}_f + \frac{1}{\sqrt{w}} \left(\sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} x_{m+1,\jmath} - \sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} x_{1,\jmath} \right).$$

In connection with our weighing scheme, the weight of each element of the stream must be adapted as well. More specifically, the weight of each element $x_{\imath\jmath}$ of X^{old} is multiplied by c^v, and the weights of the new elements $x_{m+1,\jmath}$, coming from block B_{m+1}, are given by $c^{v-\jmath-1}$, $0 \leq \jmath \leq v-1$. Noting that $\mathrm{DFT}_f(c^v \cdot X^{old}) = c^v\,\mathrm{DFT}_f(X^{old})$ due to the linearity of the DFT, the DFT coefficients are now modified as follows:

$$\begin{aligned} \mathrm{DFT}_f \leftarrow\ & e^{\frac{i2\pi fv}{w}} \cdot c^v\,\mathrm{DFT}_f \\ & + \frac{1}{\sqrt{w}} \left(\sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} c^{v-\jmath-1} x_{m+1,\jmath} - \sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} c^{w+v-\jmath-1} x_{1,\jmath} \right). \end{aligned}$$

Using the values

$$\beta_k^f \stackrel{\mathrm{df}}{=} \sum_{\jmath=0}^{v-1} e^{\frac{i2\pi f(v-\jmath)}{w}} c^{v-\jmath-1} x_{k,\jmath}, \tag{6}$$

the above update rule simply becomes

$$\mathrm{DFT}_f \leftarrow e^{\frac{i2\pi fv}{w}} \cdot c^v\,\mathrm{DFT}_f + \frac{1}{\sqrt{w}} \left(\beta_{m+1}^f - c^w \beta_1^f \right).$$

As can be seen, the processing of one block basically comes down to maintaining the Q- and β-coefficients. The time complexity of the above update procedure is therefore $O(nvu)$. Moreover, the procedure needs space $O(nmu + nv)$: For each stream, the β-coefficients have to be stored for each block plus the complete last block (u is the number of DFT coefficients used for representing a stream, see section 3.5 below).

3.5 Distance Approximation and Smoothing

We now turn to the problem of computing the Euclidean distance

$$\|X - Y\| = \left(\sum_{\imath=0}^{w-1} (x_\imath - y_\imath)^2 \right)^{1/2}$$

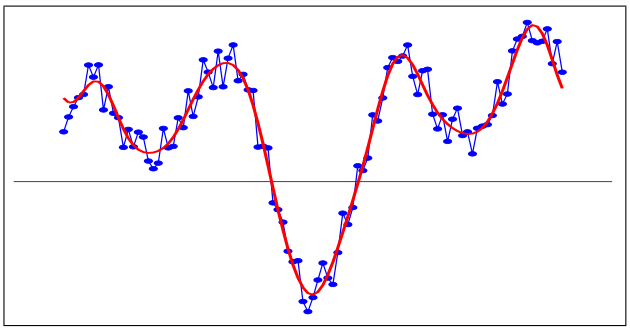

Figure 4: Original (noisy) signal and DFT-filtered curve.

between two streams X and Y (resp. the distance $\|V \odot X^n - V \odot Y^n\|$ between their normalized and weighted versions) in an efficient way. A useful property of the DFT is the fact that it preserves Euclidean distance, i.e.

$$\|X - Y\| = \|\,\mathrm{DFT}(X) - \mathrm{DFT}(Y)\,\|\,. \tag{7}$$

Furthermore, the most important information is contained in the first DFT coefficients. In fact, using only these coefficients within the inverse transformation (which recovers the original signal X from its transform $\mathrm{DFT}(X)$) comes down to implementing a low-pass filter and, hence, to using DFT as a smoothing technique (see Fig. 4 for an illustration).

Therefore, a reasonable idea is to approximate the distance (7) by using only the first $u \ll w$ rather than all of the DFT coefficients and, hence, to store the values (6) only for $f = 0, \ldots, u-1$. More specifically, since the middle DFT coefficients are usually close to 0, the value

$$\left(2 \sum_{f=1}^{u-1} (\mathrm{DFT}_f(X) - \mathrm{DFT}_f(Y))(\overline{\mathrm{DFT}_f(X) - \mathrm{DFT}_f(Y)}) \right)^{1/2}$$

is a good approximation to (7). Here, we have used that $\mathrm{DFT}_{w-f+1} = \overline{\mathrm{DFT}}_f$, where $\overline{\mathrm{DFT}}_f$ is the complex conjugate of DFT_f. Moreover, the first coefficient DFT_0 can be dropped, as for real-valued sequences the first DFT coefficient is given by the mean of that sequence, which vanishes in our case (recall that we normalize streams in a first step).

The above approximation has two advantages. First, by filtering noise we capture only those properties of a stream that are important for its characteristic time-dependent behavior. Second, the computation of the distance between two streams becomes much more efficient due to the related dimensionality reduction.

Figure 5: Snapshots of a dynamic clustering structure at different time points (upper left to lower right).

4 Fuzzy Clustering of Data Streams

The previous section has presented an efficient method for computing the (approximate) pairwise Euclidean distances between data streams in an incremental way. On the basis of these distances, it is principally possible to apply any clustering method. In this section, we propose an incremental version of the fuzzy C-means algorithm, subsequently referred to as FCM-DS (for Fuzzy C-Means on Data Streams).

For the following reasons, FCM appears to be especially suitable in our context: Firstly, K-means is an iterative procedure that can be extended to the online setting in a natural way. Secondly, the *fuzzy* variant of this algorithm is strongly advised, since data streams are evolving objects. As a consequence, the clustering structure will change over time, and typically doing so in a "smooth" rather than an abrupt manner.

To illustrate the second point, suppose that the objects to be clustered are continuously moving points in a two-dimensional space.[4] Fig. 5 shows snapshots of a dynamic clustering structure at different time points. At the beginning, there is only one big cluster. However, this cluster begins to divide itself into three small clusters, two of which are then again combined into a single cluster. Thus, there are time points where the structure definitely consists of one (first picture), two (sixth picture), and three (fourth picture) clusters. In-between, however, there are intermediate states, for which it is neither possible to determine the number of clusters nor to assign an object to one cluster in an unequivocal way.

Our incremental FCM method works as shown in Fig. 6: The standard FCM algorithm is run on the current data streams. As soon as a new block is available for all streams, the

[4]This example just serves an illustration purpose. Even though the objects might in principle be thought of as data streams over an extremely short window ($w = 2$), their movement is not typical of data streams.

1. Initialize K cluster centers at random
2. Repeat
3. Assign membership degrees of each stream to the cluster centers
4. Replace each center by the center of its associated fuzzy cluster
5. If a new block is complete:
6. Update the streams and pairwise distances
7. Update the optimal cluster number K

Figure 6: Incremental version of the K-means algorithm for clustering data streams.

current streams are updated by the shift operation (4). FCM is then simply continued. In other words, the cluster model of the current streams is taken as an initialization for the model of the new streams. This initialization will usually be good or even optimal since the new streams will differ from the current ones but slightly.

An important additional feature that distinguishes FCM-DS from standard FCM is an incremental adaptation of the cluster number K. As already mentioned above, such an adaptation is very important in the context of our application where the clustering structure can change over time. Choosing the right number K is a question of practical importance in standard K-means also, and a number of (heuristic) strategies has been proposed. A common approach is to look at the cluster dissimilarity (the sum of distances between objects and their associated cluster centers) for a set of candidate values $K \in \{1, 2, \ldots, K_{max}\}$. The cluster dissimilarity is obviously a decreasing function of K, and one expects that this function will show a kind of irregularity at K^*, the optimal number of clusters: The benefit of increasing the cluster number K will usually be large if $K < K^*$ but will be comparatively small later on. This intuition has been formalized, e.g., by the recently proposed *gap statistic* [26].

Unfortunately, the above strategy is not practicable in our case as it requires the consideration of too large a number of candidate values. Instead, we pursue a local adaptation process that works as follows: In each iteration phase, one test is made in order to check whether the cluster model can be improved by increasing or decreasing K^*, the current (hopefully optimal) cluster number. By iteration phase we mean the phase between the entry of new blocks, i.e., while the streams to be clustered remain unchanged. Further, we restrict ourselves to adaptations of K^* by ± 1, which is again justified by the fact that the clustering structure will usually not change abruptly. In order to evaluate a cluster model, we make use of a quality measure (validity function) $Q(\cdot)$ that will be introduced in Section 5 below. Let $Q(K)$ denote this quality measure for the cluster number K, that is, for the cluster model obtained for this number. The optimal cluster number is then updated as follows:

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

standard variant	fuzzy variant	complexity
$\max_k \max_{x,y \in C_k} \|x - y\|$	$\max_k \max_{\imath,\jmath} u_{\imath k}^m u_{\jmath k}^m \|x_\imath - x_\jmath\|^2$	$O(Kn^2)$
$\max_k \max_{x \in C_k} \|x - c_k\|$	$\max_k \max_\imath u_{\imath k}^m \|x_\imath - c_k\|^2$	$O(Kn)$
$\sum_k \max_{x,y \in C_k} \|x - y\|$	$\sum_k \max_{\imath,\jmath} u_{\imath k}^m u_{\jmath k}^m \|x_\imath - x_\jmath\|^2$	$O(Kn^2)$

Table 2: Measures of intra-cluster variability. C_k denotes the k-th Cluster and c_k its center.

Intuitively, going from K^* to $K^* - 1$ means that one of the current clusters has disappeared, e.g., since the streams in this cluster have become very similar to the streams in a neighbored cluster. Thus, $Q(K^* - 1)$ is derived as follows: One of the current candidate clusters is tentatively removed, which means that each of its elements is re-assigned to the closest cluster (center) among the remaining ones (note that different elements might be assigned to different clusters). The quality of the cluster model thus obtained in 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.

Going from K^* to $K^* + 1$ assumes that an additional cluster has emerged, e.g., since a homogeneous cluster of streams has separated into two groups. To create this cluster we complement the existing K^* centers by one center that is defined by a randomly chosen object (stream). The probability of a stream to be selected is reasonably defined as an increasing function of the stream's distance from its cluster center. In order to compute $Q(K^* + 1)$, we try out a fixed number of randomly chosen objects and select the one that gives the best cluster model.

5 Quality Measures

5.1 Fuzzy Validity Function

Regarding the evaluation of a cluster model in terms of a measure $Q(\cdot)$, several proposals can be found in 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 tradeoff between intra-cluster and inter-cluster variability (see tables 2 and 3, respectively, for some examples), which is of course a reasonable principle. Besides, our application gives rise to a number of additional requirements:

standard (= fuzzy) variant	complexity
$\min_{k,\ell} \min_{x \in C_k, y \in C_\ell} \|x - y\|^2$	$O(K^2 n^2)$
$\sum_k \min_\ell \min_{x \in C_k, y \in C_\ell} \|x - y\|^2$	$O(K^2 n^2)$
$\sum_k \min_\ell \|c_k - c_\ell\|^2$	$O(K^2)$
$\min_{k,\ell} \|c_k - c_\ell\|^2$	$O(K^2)$

Table 3: Measures of intra-cluster variability. C_k denotes the k-th Cluster and c_k its center.

(a) 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 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.

(b) 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. Both candidate structures, however, are not fully optimized with regard to the objective function (2). In fact, this optimization only starts *after* the apparently optimal structure 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 (2).

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

A widely used validity function is the so-called Xie-Beni index or separation [27], which is defined as

$$\frac{\frac{1}{n} \sum_{\imath=1}^{n} \sum_{k=1}^{K} u_{\imath k}^m \|x_\imath - c_k\|^2}{\min_{k,\ell} \|c_k - c_\ell\|^2}. \tag{8}$$

As most validity measures do, (8) 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 (8) just corresponds to the objective function (2), 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 replace 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,

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

and the dissimilarity between two clusters as

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

These dissimilarities are aggregated by means of

$$\frac{1}{K(K-1)} \sum_{1 \le k < \ell \le K} \frac{1}{D(C_k, C_\ell)}, \tag{9}$$

thereby putting higher weight on smaller dissimilarities. Replacing the denominator in (8) by (9), we thus obtain

$$\sum_{1 \le k < \ell \le K} \frac{1}{D(C_k, C_\ell)} \cdot \sum_{\imath=1}^{n} \sum_{k=1}^{K} \|x_\imath - c_k\|^2 \, u_{\imath k}^m. \tag{10}$$

It is of course not possible to prove the concavity of (10) 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. Corresponding experimental results are omitted here due to reasons of space.

The only remaining problem concerns clusters that are unreasonably small. To avoid such clusters, we add a penalty of M/k for every cluster having less than 3 elements with membership of at least $1/2$ (M is a very high, implementation-dependent constant).

5.2 Similarity between Cluster Models

A validity function $Q(\cdot)$ as introduced above measures the quality of a single cluster model. What we still need (in the experimental section below) is a measure of similarity (distance) for comparing two alternative structures (fuzzy partitions), say, $\mathcal{X} = \{C_1 \dots C_k\}$ and $\mathcal{Y} = \{C'_1 \dots C'_\ell\}$. In literature, such measures are known as *relative* evaluation measures.

Intuitively, a partition $\mathcal{X} = \{C_1 \dots C_k\}$ is similar to a partition $\mathcal{Y} = \{C'_1 \dots C'_\ell\}$ if, for each cluster in $\mathcal{X}$, there is a similar cluster in $\mathcal{Y}$, and vice versa, for each cluster in $\mathcal{Y}$, there is a similar cluster in $\mathcal{X}$. Formalizing this idea, we can write

$$S(\mathcal{X}, \mathcal{Y}) = s(\mathcal{X}, \mathcal{Y}) \otimes s(\mathcal{Y}, \mathcal{X}), \tag{11}$$

where $s(\mathcal{X}, \mathcal{Y})$ denotes the similarity of $\mathcal{X}$ to $\mathcal{Y}$ (in the above sense) and vice versa $s(\mathcal{Y}, \mathcal{X})$ the similarity of $\mathcal{Y}$ to $\mathcal{X}$:

$$s(\mathcal{X}, \mathcal{Y}) = \bigotimes_{\imath=1\ldots k} \bigoplus_{\jmath=1\ldots \ell} s(C_\imath, C'_\jmath), \tag{12}$$

where $\otimes$ is a t-norm (modeling a logical conjunction), $\oplus$ a t-conorm (modeling a logical disjunction), and $s(C_\imath, C'_\jmath)$ denotes the similarity between clusters $C_\imath$ and $C'_\jmath$. Regarding the latter, note that $C_\imath$ and $C'_\jmath$ are both fuzzy subsets of the same domain (namely all data streams), so that we can refer to standard measures for the similarity of fuzzy sets. One such standard measure is

$$s(C_\imath, C'_\jmath) = \frac{|C_\imath \cap C'_\jmath|}{|C_\imath \cup C'_\jmath|} = \frac{\sum_q \min(u_{q,\imath}, u'_{q,\jmath})}{\sum_q \max(u_{q,\imath}, u'_{q,\jmath})},$$

where u_{qi} and u'_{qj} denote, respectively, the membership of the q-th data stream in the clusters $C_\imath$ and $C'_\jmath$.

As one potential drawback of (12) let us mention that it gives the same influence to every cluster, regardless of its size. That is, the degree

$$s_\imath = \bigoplus_{\jmath=1\ldots \ell} s(C_\imath, C'_\jmath) \tag{13}$$

to which there is a cluster in $\mathcal{Y}$ similar to $C_\imath$ has the same influence for every $\imath$, regardless of the size of cluster $C_\imath$. Thus, one might think of weighting (13) by the relative size $w_\imath = |C_\imath|/n$, where n is the number of objects (data streams), that is, to replace (13) in (12) by $m(w_\imath, s_\imath)$. This comes down to using a *weighted* t-norm aggregation instead of a simple one [22]:

$$s(\mathcal{X}, \mathcal{Y}) = \bigotimes_{\imath=1\ldots k} m\left(w_\imath, \bigoplus_{\jmath=1\ldots \ell} s(C_\imath, C'_\jmath)\right). \tag{14}$$

In the experimental section below, we shall employ two different versions of (14), with $\otimes$, $\oplus$, $m(\cdot)$ defined as follows:

$$a \otimes b = \min(a, b), \quad a \oplus b = \max(a, b), \quad m(w, s) = \max(1 - w, s) \tag{15}$$

$$a \otimes b = ab, \quad a \oplus b = a + b - ab, \quad m(w, s) = s^w \tag{16}$$

Again, we refrain from a more detailed discussion of the pros and cons of the above similarity measure. It should be noted, however, that there are of course other options, and that alternative measures can indeed be found in literature. Anyway, as a reasonable feature of (11), note that it is a normalized measure between 0 and 1, where the latter value is assumed for perfectly identical structures. This property is often violated for fuzzifications of standard (relative) evaluation measures such as, e.g., those based on the comparison of coincidence matrices.

6 Experimental Validation

A convincing experimental validation of FCM-DS as introduced above is difficult for several reasons. Firstly, the evaluation of clustering methods is an intricate problem anyway, since an objectively "correct solution" in the sense of a real clustering structure does usually not exist, at least not in the case of real-world data. Moreover, the performance of a clustering method strongly depends on the choice of the data set (selective superiority problem). In fact, most methods give good results for a particular type of data but otherwise perform poorly. Secondly, since FCM-DS is the first method for clustering complete data streams, there are no alternative methods to compare with. Thirdly, real-world streaming data is currently not available in a form that is suitable for conducting systematic experiments. Therefore, we decided to carry out experiments with synthetic data. As an important advantage of synthetic data let us note that it allows for conducting experiments in a *controlled* way and, hence, to answer specific questions concerning the performance of a method and its behavior under particular conditions.

Synthetic data was generated in the following way: First, a *prototype* $p(\cdot)$ is generated for each cluster. This prototype is either predefined in terms of a specific (deterministic) function of time, or is generated as a stochastic process defined by means of a second-order difference equation:

$$\begin{aligned} p(t+\Delta t) &= p(t) + p'(t+\Delta t) \\ p'(t+\Delta t) &= p'(t) + u(t), \end{aligned} \tag{17}$$

$t = 0, \Delta t, 2\Delta t \ldots$. 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 $p(\cdot)$ will be. 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

$$x(t) = p\big(t + h(t)\big) + g(t),$$

where $h(\cdot)$ and $g(\cdot)$ are stochastic processes that are generated in the same way as the prototype $p(\cdot)$.[5] Fig. 7 shows a typical prototype together with a distortion $x(\cdot)$.

Of course, the above data generating process seems to be quite convenient for the K-means method. It should be stressed, therefore, that our experiments are not intended to investigate the performance of K-means itself (by now a thoroughly investigated algorithm with known advantages and disadvantages). Instead, our focus is more on the extensions that have been proposed in previous sections. More specifically, we were interested in the performance of

[5] However, the constant a that determines the smoothness of a process can be different for $p(\cdot), h(\cdot)$, and $g(\cdot)$.

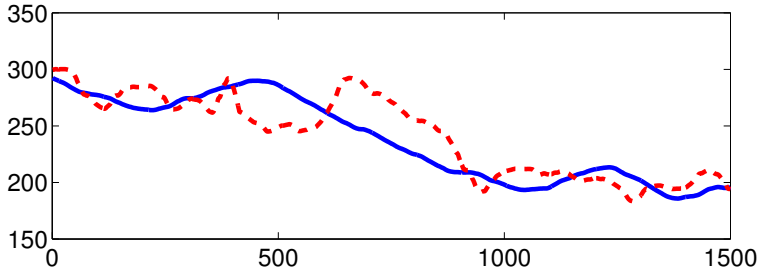

Figure 7: Example of a prototypical data stream (solid line) and a distorted version (dashed line).

our adaptation scheme for the cluster number K, the advantages of using a fuzzy instead of a crisp cluster method in dynamic environments, and the tradeoff between efficiency and quality in connection with the data preprocessing in FCM-DS.

6.1 First Experiment

In a first experiment, we investigated the ability of FCM-DS to adapt to a changing number of clusters. To this end, 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, using the values 0.04 and 0.5, respectively, as a smoothness parameter for the processes $h(\cdot)$ and $g(\cdot)$. The original streams were compressed using 100 DFT coefficients.

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 blocks (the block size is 512 data points).

Fig. 8 shows the value of λ and the number of clusters generated by FCM-DS 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.

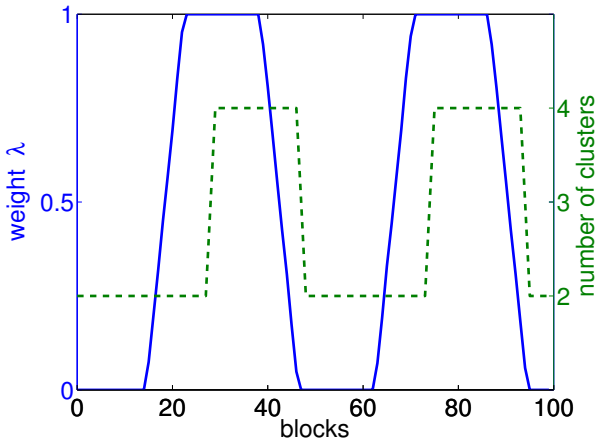

Figure 8: Weight of the parameter λ (solid line) and number of clusters (dashed line) in the first experiment.

6.2 Second Experiment

The second experiment is quite similar to the first 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. 9 shows the value of λ and the (average) membership degree of the two streams in the second cluster. As can be seen, the membership degrees are again correctly adapted with a small delay of time.

We repeated the same experiment, this time using 5 instead of only two streams that move between the two clusters. The results, shown in Fig. 10, are relatively similar, with one notable exception: Since the number of moving streams is now higher (than 3), FCM-DS 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.

6.3 Third Experiment

The purpose of the third experiment was to study the scalability of FCM-DS, that is, the tradeoff between efficiency and quality of data stream clustering. To this end, we have conducted experiments with 100 streams, generated as distortions of 6 prototypes (17). The

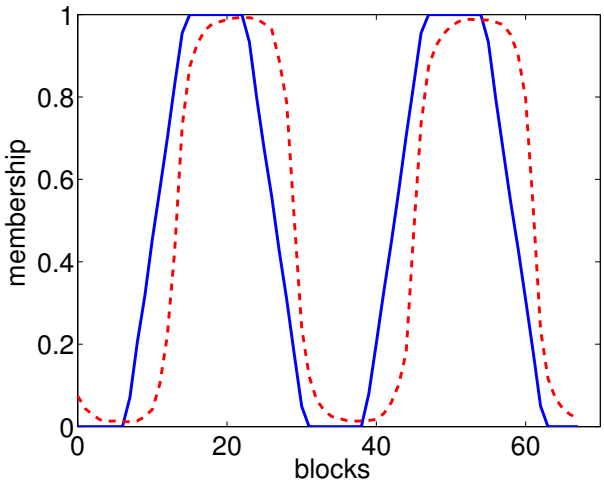

Figure 9: Weight of the parameter λ (solid line) and degree of membership of the moving streams in the second cluster (dashed line).

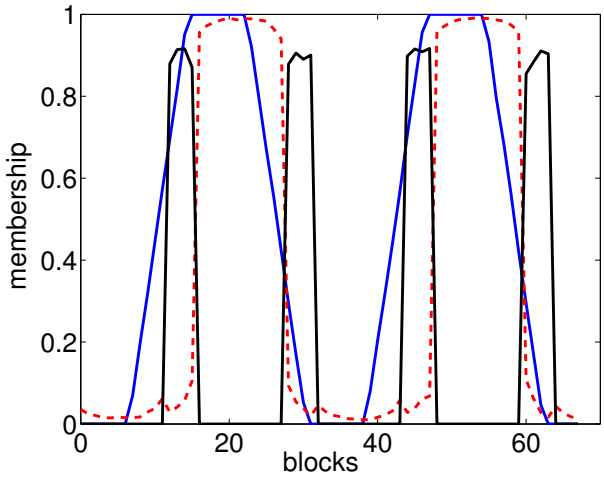

Figure 10: Weight of the parameter λ and degree of membership of the moving streams in the second cluster (dashed line) and the intermediate cluster (solid line).

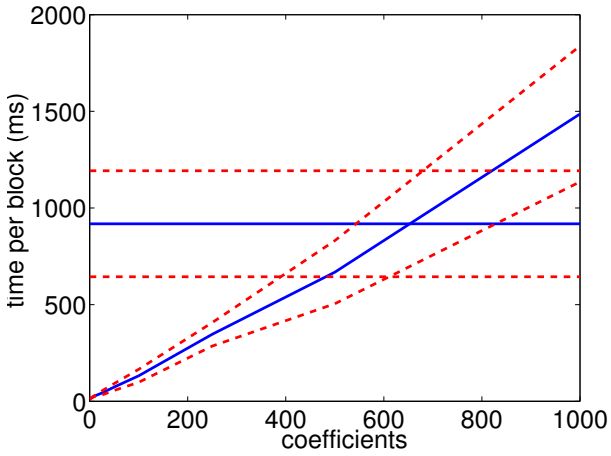

Figure 11: Mean processing time per block (solid) ± standard deviation (dashed) for different numbers of DFT coefficients (increasing curve), average processing time ± standard deviation for the original streams (horizontal lines).

parameter a was again set, respectively, to 0.04, 0.04, and 0.5 for the processes $p(\cdot)$, $h(\cdot)$, and $g(\cdot)$. The window size and block size were set to 2048 and 128, respectively.

As an efficiency parameter we have measured the time needed in order to process one block, that is to preprocess the data and to update the cluster model. Fig. 11 shows the mean processing time together with the standard deviation for different numbers of DFT coefficients. Moreover, we have plotted the average processing time and standard deviation for the original streams. As it was to be expected, the time complexity increases as an approximately linear function of the number of DFT coefficients. The critical number of coefficients is around 600. That is, when using 600 or more DFT coefficients, the preprocessing of the data will no longer pay off.

As can also be seen in Fig. 11, the processing time for one block is < 1 second. Thus, the system can process streams with an arrival rate of ≈ 150 elements per second. When approximating the original streams with 100 DFT coefficients, an arrival rate of ≈ 1000 per second can be handled. As an aside, we note that the memory requirements are not critical in this application, as the number of data streams is still manageable.

Apart from efficiency aspects, we were interested in the quality of the results produced by FCM-DS. More specifically, we aimed at comparing the cluster models obtained by our online method, which preprocesses data streams first and clusters them in a low-dimensional space afterwards, with the models that would have been obtained by clustering the *original* data streams. In fact, recall that in FCM-DS, distances between data streams are computed after having transformed the original streams from a w-dimensional space to a u-dimensional space, where $u \ll w$. In other words, a fuzzy partition is derived, not for the actual streams, but

# coeff.	similarity (15)	similarity (16)
2	0.72867 (0.00341)	0.95853 (0.00680)
5	0.73267 (0.00432)	0.96988 (0.00293)
10	0.74191 (0.00726)	0.97900 (0.00387)
25	0.86423 (0.00988)	0.98814 (0.00388)
50	0.93077 (0.01116)	0.99384 (0.00366)
100	0.93721 (0.01454)	0.99273 (0.00482)
250	0.95006 (0.00811)	0.99509 (0.00341)
500	0.95266 (0.01421)	0.99532 (0.00330)
750	0.93768 (0.01651)	0.99367 (0.00378)
1000	0.94329 (0.01002)	0.99525 (0.00303)

Table 4: Average similarity (and standard deviation) for cluster models derived with and without transformation of data streams, for different numbers of DFT coefficients.

only for *approximations* thereof. On the one hand, this makes clustering in an online scenario more efficient. On the other hand, clustering in a space of lower dimension might of course come along with a quality loss, in the sense that the fuzzy partition in this space is different from the partition in the original space.

To compare the cluster models obtained, respectively, for the uncompressed and compressed data streams, we used the two versions (15) and (16) of the similarity measure (14). More specifically, data streams were clustered over a time horizon of 200 blocks, with and without DFT, and the mean similarity (14) was derived. This was repeated 10 times. Table 4 shows the averages over these 10 runs and the corresponding standard deviations. As can be seen, the results are quite satisfying in the sense that the similarity degrees are reasonably high. Moreover, the more DFT coefficients are used, the better the results become, even though good performance is already achieved for comparatively few coefficients.

We repeated the same experiment, this time using the correct number of $K = 6$ clusters for the reference model. That is, to cluster the original, uncompressed streams, we didn't try to estimate (adapt) the best number of clusters and instead assumed this number to be known. The results, shown in Table 5, are rather similar to those from the first experiment, with an important exception: The quality is no longer a monotone increasing function of the number of DFT coefficients. Instead, the optimal number of coefficients is around 25, and using more exact approximations of the original streams deteriorates rather than improves the cluster models. This interesting phenomenon can be explained as follows: Using a small number of DFT coefficients comes along with a strong smoothing effect of the original streams, and this apparently helps to reveal the real structure in the cluster space. Stated differently, by filtering noise, the data streams are moved closer to their prototypes and, hence, the complete

# coeff.	similarity (15)	similarity (16)
2	0.78258 (0.00639)	0.99192 (0.00206)
5	0.79732 (0.00031)	0.99743 (0.00002)
10	0.80665 (0.00113)	0.99893 (0.00013)
25	0.83063 (0.00420)	0.99242 (0.00098)
50	0.81600 (0.00543)	0.98403 (0.00251)
100	0.81269 (0.00557)	0.98110 (0.00208)
250	0.81171 (0.00730)	0.98267 (0.00336)
500	0.80855 (0.00369)	0.98207 (0.00204)
750	0.81145 (0.00669)	0.98236 (0.00184)
1000	0.80992 (0.01129)	0.98323 (0.00353)

Table 5: Average similarity (and standard deviation) for cluster models derived with and without transformation of data streams. In the latter case, the cluster number was assumed to be known.

structure closer to the "ground truth". Seen from this point of view, data preprocessing does not only increase efficiency but also improves the quality of results.

7 Conclusions

In this paper, we have addressed the problem of clustering data streams in an online manner. To this end, we have developed FCM-DS, an adaptable, scalable online version of the fuzzy C-means algorithm. Data streams can be perceived as moving objects in a very high-dimensional data space, the clustering structure of which is subject to continuous evolution. Therefore, a fuzzy approach appears particularly reasonable.

Another key aspect FCM-DS is an efficient preprocessing step which includes an incremental computation of the distance between data streams, using a DFT approximation of the original data. This way, it becomes possible to cluster thousands of data streams in real-time.

In order to investigate the performance and applicability of FCM-DS in a systematic way, we have performed experiments with synthetic data. The results of these experiments have shown that FCM-DS achieves an extreme gain in efficiency at the cost of an acceptable loss in quality. Depending on the point of view and on the assumptions on the data generating process, it can even be argued that preprocessing can improve the quality by removing noise in the original streams.

Going beyond the relatively simple K-means approach by trying out other clustering methods is a topic of ongoing and future work. In this respect, one might think of extensions of

K-means, such as Gath-Geva clustering [14], as well as alternative methods such as, e.g., self-organizing maps. Likewise, other techniques might be tested in the preprocessing step of our framework, especially for the online approximation of data streams.

We conclude the paper by noting that FCM-DS does not necessarily produce the end product of a data mining process. Actually, it can be considered as transforming a set of data streams into a new set of streams the elements of which are cluster memberships. These "cluster streams" can be analyzed by means of other data mining tools.

The JAVA implementation of FCM-DS is available for experimental purposes and can be downloaded, along with a documentation, under the following address: wwwiti.cs.uni-magdeburg.de/iti_dke.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 1–16. ACM, 2002.

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

[3] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik. Scalable distributed stream processing. In *Proceedings CIDR–03: First Biennial Conference on Innovative Database Systems*, pages 257–268, Asilomar, CA, 2003.

[4] J. Considine, F. Li, G. Kollios, and JW. Byers. Approximate aggregation techniques for sensor databases. In *ICDE–04: 20th IEEE Int'l Conference on Data Engineering*, 00:449, 2004.

[5] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 296–306. ACM Press, 2003.

[6] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 40–51. ACM Press, 2003.

[7] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proc. of the Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 2002)*, pages 635–644, 2002.

[8] Mayur Datar and S.Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Algorithms - ESA 2002*, pages 323–334. Springer, 2002.

[9] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 106–113, Williamstown, MA, 2001.

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

[11] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56:9–33, 2004.

[12] MM. Gaber, S. Krishnaswamy, and A. Zaslavsky. Cost-efficient mining techniques for data streams. In *Proceedings of the 2nd Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation*, pages 109–114. Australian Computer Society, Inc., 2004.

[13] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: you only get one look. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 635–635. ACM Press, 2002.

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

[15] C. Giannella, J. Han, J. Peia, X. Yan, and PS. Yu. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*, pages 105–124. AAAI/MIT, 2003.

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

[17] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[18] D. Gunopulos and G. Das. Time series similarity measures and time series indexing. In *Proc. 2001 ACM SIGMOD International Conference on Management of Data*, page 624, Santa Barbara, California, 2001.

[19] JA. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.

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

[21] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM Press, 2001.

[22] U. Kaymak and HR. van Nauta Lemke. A sensitivity analysis approach to introducing weight factors into decision functions in fuzzy multicriteria decision making. *Fuzzy Sets and Systems*, 97(2):169–182, 1998.

[23] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 102–111, Edmonton, Alberta, Canada, July 2002.

[24] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering, 00:0685, 2002.

[25] S. Papadimitriou, C. Faloutsos, and A. Brockwell. Adaptive, hands-off stream mining. In *29th International Conference on Very Large Data Bases*, pages 560–571, 2003.

[26] R. Tibshirami, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society – Series B*, 63(2):411–423, 2001.

[27] XL. Xie and GA. Beni. Validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(8):841–846, 1991.

[28] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. *Analyzing Data Streams - VLDB 2002*, pages 358–369, Springer , 2002.