ONLINE EMBEDDING AND CLUSTERING OF EVOLVING DATA STREAMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ALAETTİN ZUBAROĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
COMPUTER ENGINEERING

JANUARY 2023

Approval of the thesis:

ONLINE EMBEDDING AND CLUSTERING OF EVOLVING DATA STREAMS

submitted by **ALAETTİN ZUBAROĞLU** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Date:18.01.2023

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Alaettin Zubaroğlu

Signature :

ABSTRACT

ONLINE EMBEDDING AND CLUSTERING OF EVOLVING DATA STREAMS

Zubaroğlu, Alaettin
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. M. Volkan Atalay

January 2023, 123 pages

Number of connected devices is steadily increasing and this trend is expected to continue in the near future. Connected devices continuously generate data streams and the data streams may often be high dimensional and contain concept drift. Real-time processing of data streams is arousing interest despite many challenges. When limited information is available about the data and its labels, unsupervised learning and particularly clustering becomes an important method of analysis. However, most clustering algorithms require the number of clusters to be known a priori and to be given as an input to the algorithm. Moreover, data stream clustering differs from traditional clustering in many aspects and it has several challenging issues. The number of clusters even changes due to the fact that data streams evolve over time. Therefore, not only the initial number of clusters but the change in the number of clusters should also be predicted throughout the stream. Also, data embedding makes the visualization of high dimensional data possible and may simplify clustering process. There exist several data stream clustering algorithms in the literature, however no data stream embedding method exists. Uniform Manifold Approximation and Projection (UMAP) is a data embedding algorithm that is suitable to be applied on stationary (stable) data streams, though it cannot adapt concept drift. In this study, we describe two novel methods, NoCStream that predicts the number of clusters continuously; and EmCStream, to apply UMAP on evolving (non-stationary) data streams, to detect and adapt concept drift and to cluster embedded data instances using a distance or partitioning based clustering algorithm. NoCStream determines the optimal number of clusters and EmCStream embeds and clusters high dimensional evolving data streams continuously in real-time. We have evaluated EmCStream against the state-of-the-art stream clustering algorithms using both synthetic and real data streams containing concept drift. EmCStream outperforms DenStream and CluStream, in terms of clustering quality, on both synthetic and real evolving data streams. We have also evaluated NoCStream and compared its performance with other methods in terms of the prediction of number of clusters, clustering quality and its genericity. NoCStream outperforms other methods on both synthetic and real evolving data streams.

Keywords: data streams, stream clustering, evolving data streams, concept drift, drift detection, drift adaptation, number of clusters, k prediction

ÖZ

DEĞİŞKEN VERİ AKIŞLARININ ÇEVRİMİÇİ BOYUTSAL KÜÇÜLTÜLMESİ VE KÜMELENMESİ

Zubaroğlu, Alaettin
Doktora, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. M. Volkan Atalay

Ocak 2023 , 123 sayfa

Bağlantılı cihazların sayısı giderek artmakta ve bu eğilimin yakın gelecekte de devam etmesi beklenmektedir. Bağlantılı cihazlar sürekli olarak akan veri üretir ve akan veri genellikle yüksek boyutlu olabilir ve kavram kayması içerebilir. Veri akışlarının gerçek zamanlı olarak işlenmesi, birçok zorluğa rağmen ilgi uyandıran bir araştırma konusudur. Veriler ve etiketleri hakkında sınırlı bilgi mevcut olduğunda, denetimsiz öğrenme ve özellikle kümeleme, önemli bir analiz yöntemi haline gelir. Bununla birlikte, çoğu kümeleme algoritması, küme sayısının önceden bilinmesini ve algoritmaya bir girdi olarak verilmesini gerektirir. Ayrıca, akan veri kümelemesi birçok açıdan geleneksel kümelemeden farklıdır ve çeşitli zorlayıcı durumları bulunmaktadır. Akan verinin zaman içinde değişmesi nedeniyle küme sayısı değişebilir. Bu nedenle, yalnızca başlangıçtaki küme sayısının değil, akış boyunca küme sayısındaki değişimin de tahmin edilmesi gerekir. Veri gömme, yüksek boyutlu verinin görselleştirilmesini mümkün kılar ve kümeleme sürecini kolaylaştırabilir. Literatürde çeşitli akan veri kümeleme algoritmaları vardır, ancak akan veri gömme yöntemi yoktur. Uniform Manifold Approximation and Projection (UMAP), durağan, kavram kayması içermeyen akan veriye uygulamaya uygun bir veri gömme algoritmasıdır, ancak kavram kaymasına adapte olamamaktadır. Bu çalışmada iki yeni yöntem sunduk. NoCStream, küme sayısını sürekli tahmin eder. EmCStream, UMAP algoritmasını kavram kayması içeren akan veriye uygulayarak kavram kaymasını algılar, adapte olur ve mesafe ya da bölümleme tabanlı bir kümeleme yöntemi kullanarak akan veriyi kümeler. EmCStream yöntemini, kavram kayması içeren sentetik ve gerçek akan veri kullanarak, en çok bilinen akan veri kümeleme algoritmalarına karşı değerlendirdik. EmCStream, değişken sentetik ve gerçek akan veri üzerinde, kümeleme kalitesi bakımından DenStream ve CluStream'den daha iyi sonuç verdi. Ayrıca NoCStream yömtemini de değerlendirmek için, küme sayısı tahmini, kümeleme kalitesi ve genelliği bakımından diğer yöntemlerle karşılaştırdık. NoCStream, değişken sentetik ve gerçek akan veri üzerinde, diğer yöntemlerden daha iyi sonuç verdi.

Anahtar Kelimeler: akan veri, akan veri kümeleme, değişken akan veri, kavram kayması, kavram kayması algılama, kavram kayması adaptasyonu, küme sayısı, k tahmini

To my dear wife Pelin and lovely daughter Farah Ekin

ACKNOWLEDGMENTS

First and foremost, this dissertation would not have been possible without the patient guidance and never ending support of my advisor, Prof. Dr. M. Volkan Atalay. I am truly grateful for having the opportunity to work under his supervision, not only because his deep knowledge and detail-oriented endeavors allowed me to create my academic persona, but also because his passionate dedication and optimistic motivation always encouraged me to do my best, in every aspect of life. Thank you, Prof. Atalay, for all your psychological and emotional support, as well as academic.

I am absolutely honored to work with members of my thesis progress committee Prof. Dr. Pınar Duygulu Şahin and Assoc. Prof. Dr. Hande Alemdar throughout my thesis studies. I have always been inspired by their unique perspectives and motivated by their kind encouragement.

I am grateful to the members of the examining committee Prof. Dr. Fazlı Can and Prof. Dr. Pınar Karagöz for their valuable contributions.

I am indebted to my parents, Azize Zubaroğlu and Suphi Zubaroğlu, for their continuous support, confidence in me, encouragement and love throughout my life. You have both made substantial sacrifices to help me attain my goals and you will never know the degree of my appreciation or admiration of you.

I am grateful to my siblings, Güler İstanbullu, Tahsin İstanbullu, Arif Zubaroğlu, Nesrin Zubaroğlu, Tahsin Zubaroğlu and Zehra Zubaroğlu for their continuous support, encouragement and love throughout my life.

I wish to thank my nieces İris İstanbullu, Sara Ada İstanbullu, Mira Zubaroğlu and Nefes Zubaroğlu, and my nephews Nathan Ali İstanbullu and expected Baby Zubaroğlu for their cuteness and love.

I express my thanks to my parents in law Fevziye and Hasan Koşman, and to my siblings in law Sinem and Alican Koşman for their continuous support and love.

I am fortunate to have many friends and colleagues, who always showed me their support and encouragement. I thank you all without mentioning your names, for fear of forgetting to mention any of you.

Finally, I wish to express my gratitude and my special thanks to my love, Pelin Zubaroğlu, for her endless love, support, confidence in me, encouragement and understanding all these years. You are the real owner of this thesis. My daughter Farah Ekin, you have no idea how you always make me feel better and support me emotionally with your pure love. Although it is not enough in return, I dedicate this work to my wife Pelin and daughter Farah Ekin. Farah Ekin Zubaroğlu, hope these lines encourage you to pursue your dreams whenever you read, now it is time for us to play.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
EmCStream	Embedding and Clustering of High Dimensional Evolving Data Streams
NoCStream	Number of Clusters on High Dimensional Evolving Data Streams
t-SNE	t-Distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

More devices and sensors in different sectors (healthcare, production, energy, consumption etc.) are becoming interconnected and these devices continuously generate data at high speed, which is called data stream. These data streams are often high dimensional and contain concept drift. Data generated by connected devices need to be processed in temporal order in real time because offline processing of such huge amount of data requires growing storage capacity and causes delayed analysis. Hence, real-time processing of data streams has become an active research area.

A data stream is a potentially unbounded, ordered sequence of instances. A data stream S may be shown as

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N\}$$

where N goes to infinity and $\mathbf{x}_i$ is i^{th} data instance, which is a d-dimensional feature vector. Data stream differs from the traditional, stored data in many aspects. In most cases, true class labels are not available for stream instances and there is no prior knowledge about the number of classes. Therefore, being unsupervised, clustering is one of the most suitable data mining and data analysis methods for data streams. However, most clustering algorithms require the number of clusters to be known a priori and to be given as an input to the algorithm. Sample data stream clustering applications are earthquake forecasting systems, network intrusion detection, stock market analysis, real-time patient tracking, internet of things (IoT) device tracking, auto monitoring of surveillance cameras and border security using sensors.

Data clustering, is the task of grouping instances such that the instances in the same group are similar to each other and the instances in different groups are dissimilar according to the properties of the instances [1]. Hence, the objective of clustering is to minimize intra-cluster distance and maximize inter-cluster distance. However, data stream clustering differs from traditional clustering in many aspects and it has several challenging issues. Data stream clustering is a real-time task, while traditional clustering is an offline task. Data stream instances arrive on the fly, during the processing, and they can be read only once, in a certain order. Moreover, data stream instances must be processed in a short time interval, before the next instance is received. Data streams cannot be stored, only a synopsis of the stream is stored, if required. Because of these limitations, approximate results are often acceptable in stream clustering, while accurate results are expected in traditional clustering.

Data generated by connected devices are often high dimensional. Moreover, data may contain outliers and stream instances may evolve over time, which is called *concept drift*. Concept drift is the unforeseen change in the properties of the input data instances. A concept drift may also be in the form of new cluster creation, cluster disappearance, or split or merge of clusters. Such concept drifts give rise to changes in the number of clusters. For the case of the traditional data, the whole dataset is available, and properties of the instances and number of clusters do not change during the processing of the data. This makes the concept drift a data stream specific challenge. A data stream clustering algorithm should detect and adapt concept drift to obtain more accurate results. According to Bezdek and Keller [2], anomaly and concept drift detection are two most important objectives of stream data analysis.

Clustering is a more difficult task when the data are high dimensional. In order to overcome this difficulty and make visualization possible, we study online embedding of data streams into two dimensions (projection). Two most popular data embedding algorithms in the literature are t-Distributed Stochastic Neighbor Embedding (t-SNE) [3] and Uniform Manifold Approximation and Projection (UMAP) [4]. We adjust these algorithms to apply them on data streams and we perform clustering on the embedded data. In this particular case, when compared against t-SNE, UMAP shows better performance in terms of execution time and silhouette score of the embedded data, and therefore, UMAP is more suitable for data streams [5]. This finding is also supported by Bahri et al. [6] where they survey dimensionality reduction techniques and empirically compare five of them, as applied on data streams. UMAP offers the most interesting visualization while separating classes, among the methods t-SNE, Random Projection (RP) [7], Principal Component Analysis (PCA) [8] and Hashing Trick (HT) [9].

Our method, online embedding and clustering of data streams (EmCStream) [10] continuously embeds high dimensional input data into two dimensions and clusters the embedded data using k-means algorithm. EmCStream continuously checks for concept drift and when a concept drift occurs, it detects and automatically adapts concept drift. It is possible to replace k-means algorithm by any other distance or partitioning based clustering algorithm. Being the most popular and easy-to-implement clustering algorithm, we have employed k-means algorithm in our method. The main contribution of EmCStream is to successfully cluster evolving data streams, detecting and adapting concept drift, with the help of embedding.

Most clustering algorithms require the number of clusters to be known a priori and to be given as an input to the algorithm. The number of clusters even changes due to the fact that data streams evolve over time. Therefore, not only the initial number of clusters but the change in the number of clusters should also be predicted throughout the stream. Here, we also describe a novel method, NoCStream to determine the optimal Number k Of Clusters throughout a data stream. NoCStream determines the optimal number of clusters on high dimensional evolving data streams by continuously embedding high dimensional data streams into two dimensions and predicting the number of clusters in real-time. NoCStream is employed by EmCStream in order to determine the optimal number of clusters.

UMAP [4] is a manifold learning based non-linear dimension reduction technique. Although UMAP is designed to work on the whole data, it is possible to embed new data instances after training the UMAP reducer. Therefore, UMAP is suitable to be applied on stationary (stable) data streams. In this study, we adapt and apply UMAP on evolving (non-stationary) data streams; that is, we employ UMAP for data stream embedding and concept drift detection.

k-means [11] is a well known and popular partitioning based clustering algorithm.

The number of clusters, k, is a hyperparameter of the algorithm. After deciding on k cluster centers in the initialization phase, k-means at each iteration, associates each data instance to the cluster whose center is the closest one and then the algorithm recalculates the cluster centers according to the elements associated with them. k-means algorithm stops when a predefined maximum number of iterations (i) is achieved or cluster centers and labels of the instances do not change any more, which means an optimal solution is achieved.

Concept drift is the unforeseen change in the properties of the input data instances of a data stream and it is a data stream specific problem. In order to cluster successfully data stream instances, concept drift should be detected and adapted. Moreover, number of clusters should be continuously predicted throughout the whole data stream. Furthermore, data have often more than three dimensions, and therefore, they cannot be visualized directly. A common problem among the existing data stream clustering methods is that for each window, cluster labels are generated independent from the labels of the other windows and therefore, it is not possible to merge cluster labels across the windows during the stream and at the end of the stream. Moreover, existing methods do not allow the visualization of the high dimensional data and most of them do not detect and notify concept drift. Furthermore, most of the existing data stream clustering methods require the number of clusters to be known a priori and to be given as an input to the algorithm.

Motivated by these observations, we describe two novel methods, NoCStream to determine the optimal Number k Of Clusters throughout a data stream; and a data stream clustering method, EmCStream. EmCStream embeds and clusters continuously high dimensional evolving data streams, and detects, notifies and adopts concept drift and makes visualization possible. Furthermore, EmCStream outperforms two state-of-the-art stream clustering algorithms, DenStream [12] and CluStream [13], in terms of clustering quality, on both synthetic and real world data streams. NoCStream also outperforms DenStream [12] in terms of the prediction of number of clusters.

The contributions of this study are as follows.

- EmCStream successfully clusters high dimensional, evolving data streams.

- EmCStream detects, notifies and adapts concept drift.
- EmCStream embeds the high dimensional input data continuously and makes visualization possible.
- EmCStream generates consistent and coherent cluster labels that can be concatenated.
- NoCStream predicts k continuously even if it changes, without candidate k values.

1.2 Related Work

There exist several data stream clustering algorithms in the literature [14, 15, 16, 17, 18, 19, 20]. Most of the data stream clustering algorithms use a two phase approach [15]. These phases are *online phase*, also called as data abstraction phase, and *offline phase*, also called as clustering phase. In *online phase*, a synopsis of the data stream is generated and stored in specialized data structures. Synopsis of the data stream is updated when a new instance is received. Therefore the synopsis always remains up-to-date. *Offline phase*, runs periodically or whenever the user requests. In this phase, the final clustering is performed over the generated data synopsis. There also exist several fully online data stream clustering algorithms, which re-cluster the data for every new instance and keep an up-to-date clustering result.

Two most popular, baseline stream clustering algorithms are DenStream [12] and CluStream [13]. In order to adapt concept drift, DenStream and CluStream use mechanisms by which old data instances are outdated and excluded from the process, however they do not detect and notify concept drift.

There exist several concept drift detection algorithms in the literature. Among the most recent drift detection algorithms are label dependency drift detector (LD3) [21], one class drift detector (OCDD) [22], discriminative drift detector (D3) [23], accurate concept drift detection method (ACDDM) [24] and discriminative subgraph-based drift detector (DSDD) [25]. Gama et al. [26] have a comprehensive survey on concept drift detection. A more recent survey is presented by Gemaque et al. [27].

A number of data stream clustering algorithms that are not described in recent surveys are adaptive streaming k-means [28], fast evolutionary algorithm for clustering data streams (FEAC-Stream) [29], multi density data stream clustering algorithm (MuDi-Stream) [30], clustering of evolving data streams into arbitrarily shaped clusters (CEDAS) [31], improved data stream clustering algorithm [32], davies-bouldin index evolving clustering method for streaming data clustering (DBIECM) [33] and improved hierarchical density-based clustering (I-HASTREAM) [34, 35]. These most recent algorithms are reviewed comprehensively in Chapter 2 [36]. Adaptive streaming k-means and FEAC-Stream are fully online, partitioning based algorithms. DBIECM is also fully online and the only distance based algorithm. MuDi-Stream, I-HASTREAM and improved data stream clustering are density based, online-offline algorithms and CEDAS is the only fully online density based algorithm reviewed in Chapter 2.

Hozumi et al. [37] have employed very recently UMAP along with k-means algorithm, as done in EmCStream. However, their data is not streaming.

Anomaly and outlier detection is also a challenging task for data streams [38, 39, 40]. In data streams, anomalies and outliers may be caused by abrupt concept drifts. Here, we focus on incremental concept drift that is defined as step by step change in the data characteristics which is regarded as an evolution, instead of an anomaly.

There also exist a classification algorithm for evolving data streams [41] that uses UMAP for embedding, in a very similar manner with our method.

Despite several data stream clustering methods, none of them embeds the data before clustering. To the extent of our knowledge, our method is the first data stream clustering method based on embedding, in the literature. Also, there are several problems with the existing methods. The cluster labels of the processed windows cannot be unified. Concept drifts are not detected and even if they are detected, no notification is communicated. Visualization is not provided as well by the existing data stream clustering methods.

1.3 The Outline of the Thesis

The rest of the thesis is organized as follows. In Chapter 2 we provide information regarding the concepts and common characteristics of data streams, such as concept drift, data structures for data streams, time window models and outlier detection. We comprehensively review recent data stream clustering algorithms and analyze them in terms of the base clustering technique, computational complexity and clustering accuracy. A comparison of these algorithms is given along with still open problems. We indicate popular data stream repositories and datasets, stream processing tools and platforms. Open problems about data stream clustering are also discussed. Chapter 3 provides the data streams used in this study. In Chapter 4, we explain our proposed methods EmCStream and NoCStream in detail. We provide empirical evaluation details and discuss their results in Chapter 5. Section 4.1 and Section 5.2 are devoted to EmCStream while Section 4.2 and Section 5.3 are devoted to NoCStream. Chapter 6 concludes the thesis, provides a summary of key findings, and gives suggestions for future research.

CHAPTER 2

BACKGROUND INFORMATION

2.1 Introduction

More devices including sensors are becoming interconnected and interconnected devices continuously generate streams of data at high speed. Offline processing of such huge amount of data requires growing storage capacity and may cause delayed analyses. Hence, real-time processing of the data generated by the connected devices has become an active research area.

Similar to traditional data, data streams may include outliers. To achieve better performance, outliers in the data streams should be detected, interpreted and possibly removed. In data streams, it is not easy to mark an instance as outlier, because a dissimilar instance might be the first sample of a new, previously unseen cluster, i.e. it might be a precursor of a concept drift. Moreover, a dissimilar instance might be marker of an anomaly, which is very valuable for anomaly detection systems.

Data stream clustering algorithms use special data structures to keep synopsis of the input data, since it is not possible to store the whole data. Storing agglomerative sum or storing only representative samples of the data are two popular alternative structures. Moreover, users are often interested in the most recent data instances rather than the previous ones. This situation creates a requirement of obsolescence for previous data instances. In data stream clustering, it is solved by time window models.

Most of the data stream clustering algorithms use a two phase approach [15]. In *online phase* which is also called as data abstraction phase, a synopsis of the data stream is generated and stored in specialized data structures. Synopsis of the data stream is updated when a new instance is received. Therefore the synopsis always remains up-to-date. *Offline phase*, called also as clustering phase, runs periodically or whenever the user requests. In this phase, the final clustering is performed over the generated data synopsis. There also exist several fully online data stream clustering algorithms, which re-cluster the data for every new instance and keep an up-to-date clustering result. Among fully online stream clustering algorithms are DPClust [42], CEDAS [31], DBIECM [33], FEAC-Stream [29] and Adaptive Stream k-means [28].

For the evaluation of data stream clustering, traditional techniques are still valid and they are commonly used. A relatively new concept *edge computing* [43, 44, 45] is the technique to process the produced data on several edge nodes that are close to the connected devices, instead of a single central system. It is also an interest arousing novel concept, however it is out of scope of this study. We examine central data stream clustering concept that runs on a single center for the whole system.

In this chapter, Section 2.2 is devoted to issues in data stream clustering. We give information about some mechanisms of stream clustering, which are data structures, time window models, concept drift and outlier detection methods. In Section 2.3, we give brief information about the categories of stream clustering algorithms. Moreover, we examine seven most recent data stream clustering algorithms that are not mentioned in the previous surveys in more detail and explain them one by one. We make a comparative review of the examined algorithms and highlight their advantages and disadvantages against each other in Section 2.4. We summarize the open problems about data stream clustering in Section 2.5. We indicate popular stream data repositories and datasets, stream processing tools and stream processing platforms in Section 2.6, Section 2.7 and Section 2.8 respectively, before concluding the chapter in Section 2.9.

2.2 Concepts in Data Stream Clustering

The information given here that are the basic concepts used in data stream clustering facilitates explaining the recent data clustering algorithms analyzed in Section 2.3.

2.2.1 Concept Drift

Concept drift is the unforeseen change in statistical properties of data stream instances over time. There are four types of concept drift: sudden, gradual, incremental and recurring [46].

- *Sudden concept drift:* Between two consecutive instances, the change occurs at once, and after this time only instances of the new class are received. An instance that has properties of the previous class never arrives again. A data stream containing sudden concept drift might look like as follows, where different colors indicate different classes.

$$S = \{..., \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, \mathbf{x}_{11}, \mathbf{x}_{12}, ...\}$$

- *Gradual concept drift:* The number of instances belonging to the previous class decreases gradually while the number of instances belonging to the new class increases over time. During a gradual concept drift, instances of both previous and new classes are visible. A data stream containing gradual concept drift might look like as follows, where different colors indicate different classes.

$$S = \{..., \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, \mathbf{x}_{11}, \mathbf{x}_{12}, ...\}$$

- *Incremental concept drift:* Data instances belonging to the previous class evolves to a new class step by step. After the concept drift is completed, the previous class disappears. The instances that arrive during the concept drift are of transitional forms and they do not have to belong to either of the classes. A data stream containing incremental concept drift might look like as follows, where different colors indicate different classes.

$$S = \{..., \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, \mathbf{x}_{11}, \mathbf{x}_{12}, ...\}$$

- *Recurring concept drift:* The data instances change between two or more statistical characteristics several times. Neither of the classes disappears permanently but both of them arrive in turns. A data stream containing recurring concept drift might look like as follows, where different colors indicate different classes.

$$S = \{..., \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9, \mathbf{x}_{10}, \mathbf{x}_{11}, \mathbf{x}_{12}, ...\}$$

Creation of new clusters, disappearance or evolution of existing clusters are all examples of concept drift. Concept drift may also affect the cluster boundaries. If the cluster boundaries are modified, it is called *real concept drift* while in the other case, it is called *virtual concept drift*. There exist several studies in the literature for concept drift detection. Gama et al. [26] have a comprehensive survey on concept drift detection.

2.2.2 Data Structures for Data Streams

In data stream processing, it is not possible to store the whole input data because data streams are infinite and all existing processing systems have main memory constraint. Therefore, only a synopsis of the input stream is stored and this situation makes it essential to develop special data structures that enables to incrementally summarize the input stream. Four most commonly used data structures are *feature vectors, prototype arrays, coreset trees* and *grids*. Feature vectors keep the summary of the data instances, prototype arrays keep only a number of representative instances that exemplify the data, coreset trees keep the summary in a tree structure and grids keep the data density in the feature space [15, 17, 47].

2.2.3 Time Window Models

In data stream processing, it is more efficient to process recent data instead of the whole data. Different window models are developed for this purpose. There are three different window models, which are damped window, landmark window and sliding window models. These window models are presented in Figure 2.1.

2.2.3.1 Damped Window

In damped window model, recent data have more weight than the older data. The most recent instance has the most weight and the importance of the instances decreases by time. This method is usually implemented using decay functions which scale down the weight of the instances, depending on the time passed since the instance is received. One of such functions is $f(t) = 2^{-\lambda t}$, where t is the time passed and λ is the decay rate. Higher decay rate in the function means a more rapid decrease in the value. Figure 2.1 (a) demonstrates the damped window model.

2.2.3.2 Landmark Window

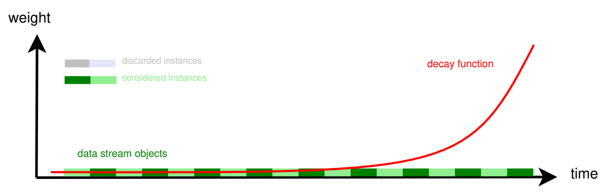
In landmark window model, the whole data between two landmarks are included in the processing and all of the instances have equal weight. Amount of data that belong to a single window is called *window length* and usually indicated by *w*. Window length can be defined as instance count or elapsed time. In landmark window method, consecutive windows do not intersect and the new window just begins from the point the previous window ends. According to this definition, data instances belong to a window are calculated using Equation 2.1 and window number of a data instance is calculated using Equation 2.2 where w is window length, x_i is i^{th} instance and W_m is m^{th} window. Indexes i and m start with zero. Figure 2.1 (b) shows the landmark window model.

$$W_m = [x_{m*w}, ..., x_{(m+1)*w-1}] \tag{2.1}$$

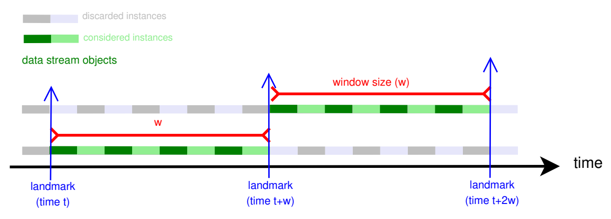
$$m = \left\lfloor \frac{i}{w} \right\rfloor \tag{2.2}$$

2.2.3.3 Sliding Window

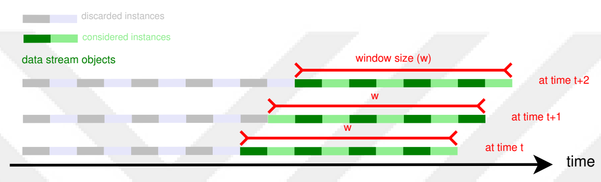
In sliding window model, the window swaps one instance at each step. The older instance moves out of the window, and the most recent instance moves in to the window by FIFO style. All instances in the window have equal weight and consecutive windows mostly overlap. Window length is a user defined parameter and should be decided according to the input data. Incremental clustering algorithms [14] are suitable to employ sliding window model. Figure 2.1 (c) describes this window model and data instances belong to a window are calculated using Equation 2.3 where w is window length, x_i is i^{th} instance and W_m is m^{th} window. Indexes i and m start with



(a) Damped window model.



(b) Landmark window model.



(c) Sliding window model.

Figure 2.1: Time window models.

zero. Figure 2.1 (c) presents the sliding window model.

$$W_m = [x_m, ..., x_{(m+w-1)}] \tag{2.3}$$

2.2.4 Outlier Detection

Outlier is a data instance that seems to be different from the remaining of the data. Either it does not belong to any of the clusters, or it belongs to a cluster whose cardinality is far less than other clusters. Let C_i be i^{th} cluster, $|C_i|$ be cardinality of C_i and k be cluster count, if $|C_j| << \frac{1}{k}\sum_{i=1}^{k}|C_i|$, then C_j is treated as outlier. There exist several definitions for the outliers in the literature [48]. An outlier can occur because of malicious activities, instrumental errors, transmission problems, data collection problems or similar [49]. In data mining, outliers negatively affect the processing accuracy, because of that, outlier detection has a crucial importance in data mining. It is possible to benefit from several existing surveys [48, 50, 51, 52, 53, 54] about outlier detection in data streams. Thakkar et al. [52] have classified outlier detection techniques in four main groups in their survey.

- *Statistical outlier detection* methods make an assumption about the data distribution. Taking the distribution into account, data instances that have a low probability to be generated, are marked as outliers. Statistical outlier detection methods are divided into two categories: parametric methods and non-parametric methods. In parametric methods, a distribution model of the data is assumed before starting, according to the parameters. This method is not suitable for data streams, since the entire dataset is not available in streams and the distribution model may change over time. In non-parametric methods, no distribution model is assumed a priori; instead, the distribution model is learned from the original data instances. This property makes non-parametric statistical outlier detection methods adoptable for data streams.

- *Distance based outlier detection* methods [50] use neighbor count of the instance to decide whether it is an outlier or not. Two parameters R and k play the main role. If the data instance has less than k neighbors in a distance of R, then it is marked as an outlier. A distance measure (or a similarity measure) must be defined. No domain knowledge is required and no distribution model assumption is done. Therefore, distance based outlier detection methods are suitable for data streams. However, they are not effective for high dimensional data streams.

- *Density based outlier detection* methods compare the density around the data instance to the density around its neighbors. If the instance has a density around it similar to its neighbors, then it is not an outlier. Otherwise it is considered as an outlier. This group of methods, are more effective than distance based methods, however they have a higher computational complexity.

- In *Clustering based outlier detection* methods [53] data instances that do not belong to any clusters, or far away from their cluster centroids, are potential outliers. Moreover, outliers may belong to a sparse or small cluster that is not close to other clusters. Ordinary data instances are expected to belong to large, dense clusters and they are relatively close to cluster centroids.

Although real-time analysis of data streams is a more recent research subject, it has many similarities with the analysis of time series data which has been studied for longer time and more abundant in the literature. Especially outlier detection in data streams is very similar to anomaly detection in time series data analysis [55, 56, 57].

2.3 Stream Clustering Algorithms

There exist several data stream clustering algorithms in the literature [15, 16, 17, 18, 19, 46, 58, 59, 60, 61, 62, 63, 64]. Data stream clustering algorithms can be categorized following the classification that is used for traditional (batch) clustering algorithms. This categorization is given in Figure 2.2 and it consists of five main classes: hierarchical based, partitioning based, density based, grid based and model based clustering. We first give brief information about these categories and related algorithms and we then examine seven most recent data stream clustering algorithms in more detail.

- *Hierarchical algorithms* use the dendrogram data structure. Dendrogram is binary tree based, and it is useful to summarize and visualize the data. Hierarchical algorithms are divided in two: agglomerative and divisive. Agglomerative algorithms start with the assumption every instance is a cluster itself, and merge the instances to create clusters step by step. On the other hand, divisive algorithms start assuming a single cluster contains whole data, and divide the clusters into smaller clusters in each step. Hierarchical algorithms have an informative output, which is the dendrogram. However, they have high complexity and they are sensitive to the outliers. Among hierarchical algorithms are BIRCH [65], CHAMELEON [66], ODAC [67], E-Stream [68] and HUE-Stream [69] [58, 59].

- *Partitioning based algorithms* split the data instances into a predefined number of clusters, based on similarity (or distance) to the cluster centroids. Number

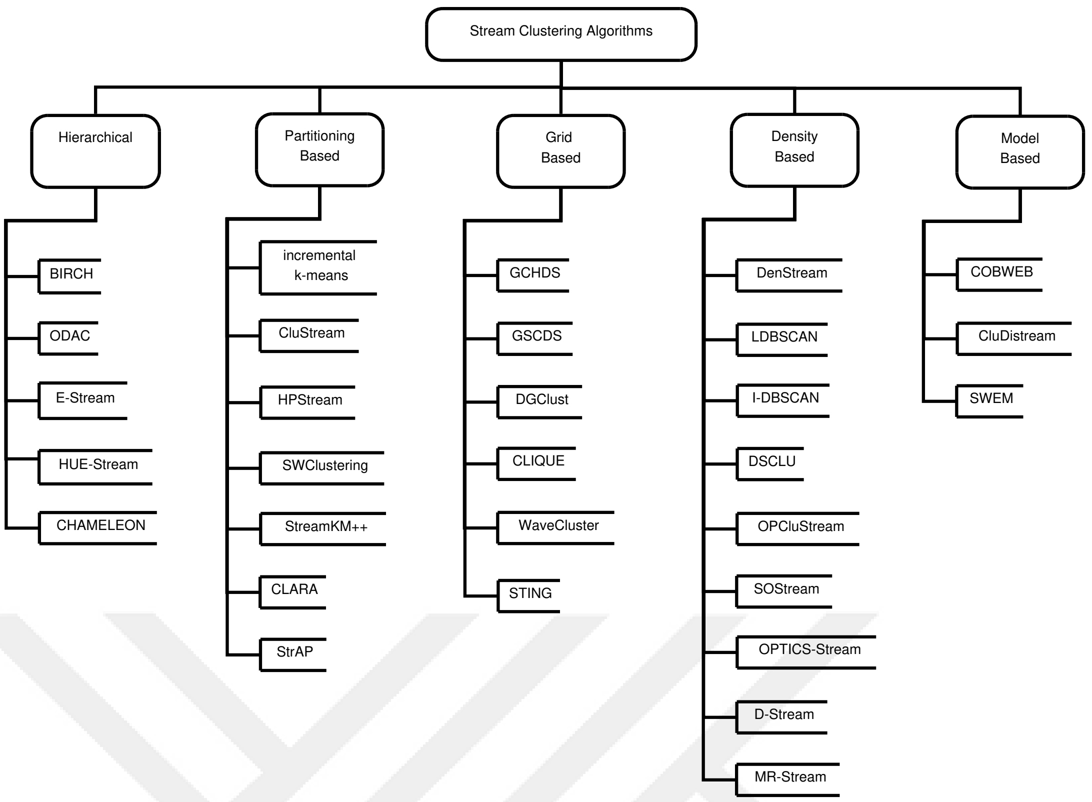


Figure 2.2: Classification of data stream clustering algorithms.

of clusters should be predefined in these algorithms, and only hyper-spherical clusters can be determined. Partitioning based algorithms have an easy implementation in general. StreamLSearch [70], incremental k-means [71], CluStream [13], HPStream [72], SWClustering [73], StreamKM++ [74], strAP [75] and CLARA [76] are partitioning based algorithms [17, 58, 59].

- *Grid based algorithms* use grid data structure. The workspace is divided into a number of cells, in a grid structure, and each instance is assigned to a cell. Then, the grid cells are clustered, according to their density. In grid based algorithms, the run time does not depend on input data count. Therefore, grid based algorithms are fast algorithms. Moreover, they are robust to noise and are able to find arbitrary shaped clusters. However, since their complexity depends on the number of the dimensions of the data, grid based algorithms are more suitable for low dimensional data. Furthermore, they need a predefined grid size. GCHDS [77], GSCDS [78], DGClust [79], CLIQUE [80], WaveCluster [81] and STING [82] are all grid based algorithms [58]. D-Stream [83] and

MR-Stream [84] are classified as grid based by Ghesmoune et al. [17], despite being classified as density based by Mousavi et al. [58].

- *Density based algorithms* keep summary of input data in large number of micro-clusters. Micro-cluster is a set of data instances that are very close to each other. Synopsis of a micro-cluster is kept with a feature vector. Then these micro-clusters are merged and formed final clusters according to density reachability and density connectivity concepts. These terms are defined as follows. If the distance between two micro-clusters is less than or equal to the sum of their radii, then they are directly density reachable. If any adjacent two clusters in a set of micro-clusters are directly density reachable, then the set of micro-clusters is density reachable. All micro-clusters that are density reachable to each other, are density connected [32]. Density based algorithms are able to find arbitrary shaped clusters and detect number of clusters. They are robust to noise as well. However, several parameters have to be selected and there are problems in finding multi-density clusters. Incremental-DBSCAN [85], LDBSCAN [86], DenStream [12], rDenStream [87], DSCLU [88], OPCluStream [89], SOStream [90], OPTICS-Stream [91], D-Stream [83] and MR-Stream [84] are classified as density based [17, 58].

- *Model based algorithms* find the data distribution model that fit best to the input data. One of the important advantages of model based algorithms is their property of noise robustness. However, their performance strongly depends on the selected model. COBWEB [92], CluDistream [93] and SWEM [94] are examples of model based algorithms [58].

Advantages and disadvantages of clustering algorithms are summarized in Table 2.1 [17, 47, 58].

The aforementioned data stream clustering algorithms have already been reviewed in the previous surveys. On the other hand, the algorithms given below have not been analyzed elsewhere, to the best of our knowledge. We give the main flow of the algorithms, show their evaluation results and present the complexity analysis. During the complexity analysis, we ignore the Euclidean distance calculation complexity, which is $O(d)$, because this is the common practice in the literature. Moreover, this cal-

Table 2.1: Advantages and disadvantages of clustering algorithms based on traditional categorization.

Algorithm	**Advantages**	**Disadvantages**
Hierarchical	Informative output	High Complexity
	(dendrogram)	Outlier sensitivity
Partitioning	Easy implementation	Predefined number of clusters
		Only hyper-spherical clusters
Grid-based	Arbitrary shaped clusters	Predefined grid size
	Fast execution time	Only low dimensional data
	Noise robustness	
Density-based	Arbitrary shaped clusters	Multidensity cluster difficulties
	Noise robustness	Many predefined parameters
Model-based	Noise robustness	Strong dependency on the model

culation is done in every data stream clustering algorithm, thus ignoring it does not change the comparison. However, any other data dimension related complexity is included in the analysis. Not surprisingly, complexity of partitioning based algorithms is a function of k and complexity of density based algorithms is a function of *micro cluster count*.

We start with Adaptive Streaming k-Means and FEAC-Stream both of which are partitioning based online algorithms. We then examine MuDi-Stream, which is a density based, online-offline algorithm. CEDAS is a density based online algorithm. Improved Data Stream Clustering Algorithm is a density based, online-offline algorithm. DBIECM, the only distance based algorithm is fully online. Note that the previous, classical classification does not include the distance based algorithms, probably, because there are not many examples of distance based algorithms. Finally, we examine I-HASTREAM, a density based hierarchical, online-offline algorithm. Figure 2.3 shows the main characteristics of the examined algorithms. Although, *ant colony optimization* methods are also being used by a number of data stream clustering algorithms [62], they are not evaluated at this time. Moreover, being an active

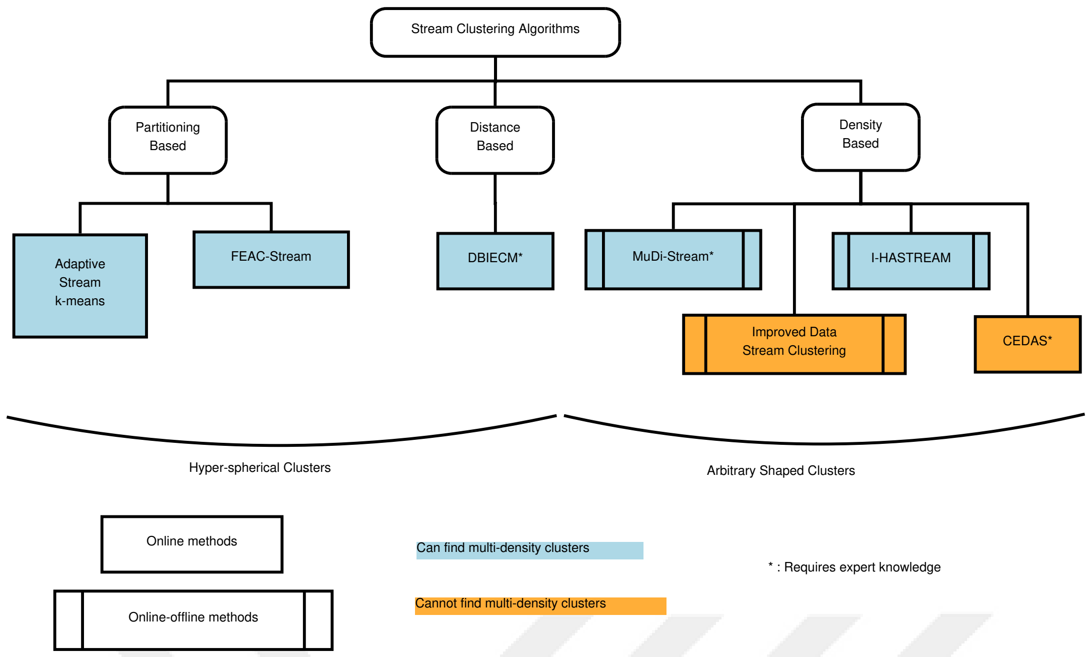

Stream Clustering Algorithms
Partitioning Based
Distance Based
Density Based
Adaptive Stream k-means
FEAC-Stream
DBIECM*
MuDi-Stream*
I-HASTREAM
Improved Data Stream Clustering
CEDAS*
Hyper-spherical Clusters
Arbitrary Shaped Clusters
Online methods
Online-offline methods
Can find multi-density clusters
Cannot find multi-density clusters
* : Requires expert knowledge

Figure 2.3: Recent data stream clustering algorithms.

research area, there also exist several recent data stream clustering algorithms that are not evaluated in this manuscript [95, 96, 97].

2.3.1 Adaptive Streaming *k*-Means (2017)

Adaptive streaming k-means [28] is an online, partitioning based data stream clustering algorithm.In general, partitioning based clustering algorithms need k as an input parameter, and these algorithms have difficulties to adapt concept drift in the input data. In this algorithm, Puschmann et al. [28] claim to overcome these two main problems.

Algorithm 1 shows the main flow of adaptive streaming k-means algorithm. k-means algorithm and silhouette coefficient calculation function are assumed to be already implemented. The algorithm is composed of two main phases, which are *initialization phase* and *continuous clustering phase*. In the initialization phase, l number of data instances are accumulated. Then groups of candidate centroids are determined at line 2. In function *determineCentroids*, in order to find k and determine candidate centroids, probability density function (PDF) of the data is calculated using kernel

Algorithm 1 streamingKMeans (S, l)

```
Input: $S$ : the input data stream
Input: $l$ : length of data sequence used for initialization
 1: % Initialization phase
 2: for $candidateCentroids$ in $determineCentroids(l$ number of data instances)
    do
 3:     run $kmeans$ with $candidateCentroids$
 4:     calculate silhouette coefficient of the $kmeans$ result
 5: end for
 6: keep $centroids$ of the best clustering
 7: % Continuous clustering phase
 8: loop
 9:     if $changeDetected$ on the input stream then
10:         re-initialize the algorithm by running again the initialization phase
11:     end if
12:     run $kmeans$ with last found, best centroids
13: end loop
```

density estimation (KDE) [98, 99]. All directional changes in the shape of PDF curve, are accepted as signs of beginning of a new region. Here the region can be defined as the area between two consecutive directional changes of the PDF curve. Number of regions is considered as a candidate k and centers of these regions are considered as candidate initial centroids. This process is pursued for each feature of the data separately. Because different features generally show different distributions, more than one k values, and different candidate centroids are found.

After finding candidate k values, clustering is performed for a set of k values where $k \in [k_{min}, k_{min} + k_{max}]$. The *for loop* at lines 2-5 is executed for these values of k and candidate centroids. Clustering results of different k values are compared according to silhouette coefficient, and best k is selected with its corresponding centroids.

The *loop* at lines 8-13 runs for continuous clustering phase. Checking for a concept drift (see Section 2.2.1) is performed at line 9. If no concept drift occurs, clustering of the input data proceeds, at line 12. However if a concept drift exists, k and centroids are recalculated (the algorithm is re-initialized) at line 10, and then clustering continues at line 12 with new k and centroids.

For concept drift detection, standard deviation and mean of the input data are stored during the execution. The algorithm tracks how these two values change over time and predicts a concept drift according to the change. When a concept drift is predicted, current cluster centroids are no longer valid. In such a case the concept drift is realized at line 9 and a reinitialization is triggered at line 10. Using this mechanism, the algorithm captures the concept drift and adapts itself to the input stream.

A limitation of this algorithm, being k-means based, only hyper-spherical clusters can be detected. Indeed, the authors indicate that k-means is used as the underlying clustering technique to clarify the approach, and the concepts of the approach can be applied to different clustering techniques.

<u>Evaluation:</u> Adaptive streaming k-means algorithm is evaluated against CluStream and DenStream algorithms, according to silhouette coefficient. Synthetic datasets with three to five dimensions, that include concept drift, are used as input data streams. Clustering quality improvement of the adaptive streaming k-means algorithm is 13%

to 40% with respect to CluStream. DenStream gives a better clustering quality for one of the datasets, for short time intervals during the execution. However, for the other datasets, clustering quality improvement of the adaptive streaming k-means algorithm is up to 280% with respect to DenStream. Furthermore, the algorithm is evaluated with real traffic data, against the non-adaptive technique, in which, the centroids are never recalculated. Adaptive streaming k-means algorithm achieves an improvement up to 31% in clustering quality when they are compared over the course of one day. When they are compared over the course of one week, clustering quality improvement of the adaptive streaming k-means is 12% on average.

Complexity Analysis: Let l be the length of the initial data sequence, and d be the data dimension. Complexity of estimating k for a single dimension is $O(l)$, because this part goes along the PDF and it has a length equal to the data length. Since this estimation is performed for all dimensions, total k estimation complexity becomes $O(d \cdot l)$. After determining initial centroids running k-means takes $O(d \cdot k \cdot cs)$ since no iterations of the algorithm are needed, where cs is the number of different centroid sets. Assigning a newly received data instance to the nearest cluster during the online phase is $O(k)$. As a result, total worst case complexity of the algorithm is $O(k)$ + $O(d \cdot l) + O(d \cdot k \cdot cs)$, which equals to $O(d \cdot l) + O(d \cdot k \cdot cs)$

2.3.2 FEAC-Stream (2017)

Fast evolutionary algorithm for clustering data streams (FEAC-Stream) [29] is an evolutionary algorithm for clustering data streams with a variable number of clusters. FEAC-Stream is a k-means based algorithm, which estimates k automatically using an evolutionary algorithm. Being fully online, FEAC-Stream does not store synopsis of the data, instead maintains the final clustering result. During the execution, clustering quality is tracked using the Page-Hinkley (PH) [100] test and if the quality falls down, the algorithm adjusts itself.

Algorithm 2 shows the main flow of FEAC-Stream algorithm. PH test function and the evolutionary algorithm are assumed to be already implemented. The algorithm is composed of two main phases, which are *initialization phase* and *continuous clustering phase*. In the initialization phase, l number of data instances are accumulated.

Algorithm 2 FEAC-Stream (S, l, λ, α, $iter$)

```
Input: $S$ : the input data stream
Input: $l$ : length of data sequence used for initialization
Input: $\lambda$ : decay rate
Input: $\alpha$ : weight threshold
Input: $iter$ : $k$-means iteration count
 1: % Initialization phase
 2: Estimate $k$ with $l$ number of data instances, using evolutionary algorithm
 3: state = normal
 4: % Continuous clustering phase
 5: loop
 6:     Read next data instance $x$ from data stream $S$
 7:     Add $x$ to the nearest cluster
 8:     Calculate weight of all clusters
 9:     Delete low weighted clusters
10:     $PH_{val}$ = Calculate $PH$ test.
11:     if $PH_{val} >$ warning threshold then
12:         state = warning
13:     end if
14:     if state is warning then
15:         Add $x$ to buffer $B$
16:     end if
17:     if $PH_{val} >$ alarm threshold then
18:         Estimate $k$ with data instances in buffer $B$, using evolutionary algorithm
19:         state = normal
20:     end if
21: end loop
```

Then k and initial clustering is calculated using an evolutionary algorithm, at line 2 and state is set to normal, at line 3. In this evolutionary algorithm, clustering is performed using k-means with a maximum of $iter$ iterations. Simplified silhouette coefficient is used as the fitness function, k is selected randomly such that $k \in [2, \sqrt{l}]$ and initial centroids are also selected randomly from the input data instances.

After clustering the initial l data instances in the initialization phase, the loop at lines 5-21 is executed for continuous clustering phase. When a new data instance is received, it is added to the nearest cluster at line 7. Weight of all clusters are calculated and low weighted clusters are deleted at line 8 and line 9, respectively. After that, PH test is calculated at line 10 and it is compared to warning and alarm threshold values. When PH test value exceeds the warning threshold, the algorithm enters to warning state. In warning state, clustering process continues and received data instances are stored in a buffer, at line 15. If PH test value exceeds alarm threshold, this means a concept drift (see Section 2.2.1) occurs and current clusters are not valid anymore. When PH test signals an alarm state, it also automatically selects samples from the input data instances that reflects a new partitioning. In such a case, FEAC-Stream clusters the data instances stored in the buffer with the evolutionary algorithm, at line 18 and sets the state back to normal, at line 19. In the evolutionary algorithm, clustering is performed using k-means with a maximum of $iter$ iterations. Simplified silhouette coefficient is used as the fitness function, k and initial centroids are specified by the PH test. FEAC-Stream uses damped window model, which is described in Section 2.2.3.1.

Being k-means based, only hyper-spherical clusters can be detected by FEAC-Stream. Moreover, clustering quality of FEAC-Stream strongly depends on the user defined parameters. FEAC-Stream requires three parameters which are length of data sequence used for initialization (l), decay rate (λ) for damped window model and minimum weight threshold (α). These parameters strongly affect the clustering quality and they are directly dependent to the input data. Because of that, FEAC-Stream requires an expert knowledge about the input data. Iteration count of k-means ($iter$) and generation count of evolutionary algorithm are used as hard coded. Moreover, warning and alarm threshold values of PH test are calculated automatically by the PH test.

Evaluation: FEAC-Stream is evaluated against CluStream-OMRk, CluStream-BkM, StreamKM++-OMRk and StreamKM++-BkM, where CluStream and StreamKM++ are the stream clustering algorithms with fixed k, while BkM and OMRk are k estimating algorithms. Both real world and synthetic datasets are used for the evaluation. Real datasets are network intrusion detection dataset, forest cover type dataset and localization data for person activity dataset. Adjusted Rand Index (ARI) is used as clustering quality metric in synthetic datasets. While all mean ARI results are very close to each other (0.97 - 0.99), FEAC-Stream has the lowest execution time. Its execution time is less by; 25$\%$ than StreamKM++-BkM, 58$\%$ than StreamKM++-OMRk, 91$\%$ than CluStream-BkM and nearly 93$\%$ than CluStream++-OMRk. Furthermore, FEAC-Stream successfully reacts to concept drifts and accordingly estimates k. For network intrusion detection dataset, simplified silhouette (SS) coefficient is used to compare the clustering quality. Again all algorithms give very good and very close (0.90 - 0.92) SS values and still FEAC-Stream gives the best execution time. Its execution time is less by; 65$\%$ than StreamKM++-BkM, 87$\%$ than StreamKM++-OMRk, 97$\%$ than CluStream-BkM and 98$\%$ than CluStream++-OMRk. For the other real datasets as well, algorithms have the same running time ordering. These results also show that, StreamKM++ is faster than CluStream and BkM is faster than OMRk.

Complexity Analysis: Let l be the length of the initial data sequence, gen is generation count of evolutionary algorithm and $iter$ is the iteration count of k-means. In the initialization phase, k is randomly selected as $k \in [2, \sqrt{l}]$. Thus, complexity of initialization phase is $O(gen \cdot iter \cdot \sqrt{l})$. Online maintenance of the algorithm requires a complexity of $O(k)$. When a concept drift occurs, the algorithm is reinitialized by running evolutionary algorithm again. However k and centroids are decided by PH test. Therefore, reinitialization requires a complexity of $O(gen \cdot iter \cdot k)$. As a result, total worst case time complexity of FEAC-Stream is $O(k)$ + $O(gen \cdot iter \cdot k)$, which equals to $O(gen \cdot iter \cdot k)$.

2.3.3 MuDi-Stream (2016)

Multi density data stream clustering algorithm (MuDi-Stream) [30] is a two phase data stream clustering algorithm. Main objective of MuDi-Stream is to improve the clustering quality on data streams with multi density clusters. Note that density based algorithms usually have problems with clusters of different densities because of the static density threshold they use. MuDi-Stream customizes the density threshold for each cluster and overcomes the problem of multi density clusters. MuDi-Stream is a hybrid algorithm based on both density based and grid based approaches. Input data instances are clustered in a density based approach and outliers are detected using grids. For data synopsis core mini-clusters are used. Core mini-clusters are specialized feature vectors (see Section 2.2.2), they keep weight, center, radius and the maximum distance from an instance to the mean. In the online phase core mini-clusters are created and kept up to date for each new data instance. In the offline phase final clustering is executed over the core mini-clusters.

Algorithm 3 shows the main flow of online phase of MuDi-Stream. When a new data instance is received, it is tried to be added to an existing core mini-cluster. For this purpose, the nearest core mini-cluster is found at line 4 and it is checked whether nearest core mini-cluster can involve this data instance or not, at line 5. If the nearest core mini-cluster is *large enough*, the data instance is added to the nearest core mini-cluster at line 6. Otherwise, the data instance is mapped into the gird in the outlier buffer, at line 8. When a data instance is mapped to a grid, density of this grid is checked and if it is *dense enough* (more than the density threshold), a new core mini-cluster is created from this grid i.e. the grid is converted to a core mini-cluster, at line 10. MuDi-Stream prunes both the grids in the outlier buffer and the core mini-clusters periodically. It is checked at line 13 whether it is pruning time or not. If it is pruning time, weight of grids and core mini-clusters are calculated according to current time, and then low weighted grids and core mini-clusters are pruned at line 14 and line 15 respectively. This pruning mechanism is an implementation of damped window model, which is described in Section 2.2.3.1.

Algorithm 4 shows the main flow of offline phase of MuDi-Stream. Initially all core mini-clusters are marked as unvisited, at line 1. After that, inside a loop, an unvisited core mini-cluster is randomly chosen at line 3 and marked as visited at line 4. If this core mini-cluster has no neighbors, it is marked as noise at line 16. If it has neighbors, a new final cluster is created with this core mini-cluster and its neighbors, at lines 6-8. After that, each unvisited core mini-cluster in the new created final cluster is marked

Algorithm 3 MuDi-Stream online phase (S, α, λ, $gridGranularity$, G)

```
Input: $S$ : the input data stream
Input: $\alpha$ : density threshold
Input: $\lambda$ : decay rate
Input: $gridGranularity$
Input: $G$ : total density grids for all dimensions
 1: Initialize the grid structure
 2: loop
 3:     Read next data instance $x$ from data stream $S$
 4:     $cmc_s$ = Find the nearest $cmc$ to $x$
 5:     if $cmc_s$ involve $x$ then
 6:         Add $x$ to $cmc_s$
 7:     else
 8:         Map $x$ to the gird
 9:         if Updated grid is dense enough then
10:             Create a $cmc$ from updated grid
11:         end if
12:     end if
13:     if It is pruning period then
14:         Remove low weighted grids
15:         Remove low weighted $cmc$s
16:     end if
17: end loop
```

Algorithm 4 MuDi-Stream offline phase (core mini-clusters)

```
Input:  core mini-clusters
 1: Mark all $cmc$s as unvisited
 2: repeat
 3:     Randomly choose an unvisited $cmc$, called $cmc_p$
 4:     Mark $cmc_p$ as visited
 5:     if $cmc_p$ has neighbors then
 6:         Create new final cluster $C$
 7:         Add $cmc_p$ to $C$
 8:         Add neighbors of $cmc_p$ to $C$
 9:         for each $cmc$ in $C$ do
10:             if $cmc$ is unvisited then
11:                 Mark $cmc$ as visited
12:                 Add neighbors of $cmc$ to $C$
13:             end if
14:         end for
15:     else
16:         Mark $cmc_p$ as noise
17:     end if
18: until All $cmc$s are visited
```

as visited and its neighbors are added to the same final cluster, at lines 9-14. This loop continues until all core mini-clusters are marked as visited.

Damped window model is used, and arbitrary shaped, multi density clusters can be detected by MuDi-Stream. Moreover, MuDi-Stream is able to handle concept drift (see Section 2.2.1), noise and outliers. However it is not suitable for high dimensional data, which makes the processing time longer, because of the grid structure. Furthermore, clustering quality of MuDi-Stream strongly depends on input parameters density threshold (α), decay rate (λ) for damped window model and *grid granularity*. These parameters require an expert knowledge about the data.

Evaluation: MuDi-Stream is tested with two real world (network intrusion detection and Landsat satellite) and six synthetic datasets. It is compared to DenStream on a data stream with concept drifts, a multi density dataset and a multi density data stream with concept drifts. MuDi-Stream outperforms DenStream on all three types of input data, according to clustering quality (Purity, Normalized Mutual Information (NMI), Rand Index (RI), Adjusted Rand Index (ARI), Folkes and Mallow index (FM), Jaccard Index and F-Measure). Clustering quality improvement of MuDi-Stream is 10% to 100% with respect to DenStream, on different datasets.

Complexity Analysis: MuDi-Stream performs a linear search on core mini-clusters for each new data instance. Complexity of this linear search is $O(c)$ where c is the number of core mini-clusters. If the new data instance cannot be merged into existing core mini-clusters, it is mapped to the grid. Let G be total density grids for all dimensions, which is exponential to the number of dimensions. Space complexity of the grid is $O(log\ G)$ because the scattered grid are pruned during the execution. Moreover, time complexity of mapping a data instance to the grid is $O(log\ log\ G)$ because the list of the grids is maintained as a tree. During the pruning, all core mini-clusters and grids are examined. This makes time complexity of pruning $O(c)$ for core mini-clusters and $O(log\ G)$ for grids. As a result, the overall time complexity of MuDi-Stream is $O(c)$ + $O(log\ log\ G)$ + $O(c)$ + $O(log\ G)$, which equals to $O(c)$ + $O(log\ G)$.

2.3.4 CEDAS (2016)

Clustering of evolving data streams into arbitrarily shaped clusters (CEDAS) [31] is a fully online data stream clustering algorithm. CEDAS is a density based algorithm designed for clustering data streams with concept drifts (see Section 2.2.1), into arbitrary shaped clusters. Damped window model (see Section 2.2.3.1) is employed with a linear decay function instead of an exponential one. CEDAS keeps synopsis of the data in micro-clusters and creates a graph structure with the micro-clusters that surpass a user defined threshold. Graph structure, where nodes are the micro-clusters and edges are the connectivity between micro-clusters, keeps the up to date final clustering results.

Algorithm 5 shows the main flow of CEDAS. When a new data instance is received, it is tried to be added to an existing micro-cluster. For that purpose, the distance from new data instance to the nearest micro-cluster is found at line 4 and it is checked whether this distance is less than the micro-cluster radius (r_0) or not, at line 5. Micro-cluster radius is a user defined, static parameter. If the distance is less than the radius, the data instance is added to the nearest micro-cluster, at line 6, and energy of this micro-cluster is set to 1 at line 7. Otherwise, a new micro cluster is created with this data instance, at line 9, and energy of the new micro-cluster is set to 1, at line 10. Energy of micro-clusters linearly fades on every cycle, with an amount of decay rate (λ), at line 12. The micro-clusters whose energy drop below zero are removed at line 13. Lastly, the graph structure is updated with the micro-clusters that surpass the density threshold (α), at line 15. Removed micro-clusters are removed from the graph structure also, and micro-clusters reached the density threshold (α) added to the graph structure. Therefore, CEDAS creates final clustering results as fully online.

CEDAS is suitable for high dimensional data under favor of maintaining a graph structure where nodes are the micro-clusters and edges are the connectivity between micro-clusters. However, clustering quality of CEDAS strongly depends on the user defined parameters. CEDAS requires three parameters which are decay rate (λ), micro-cluster radius (r_0) and minimum density threshold (α). These parameters strongly affect the clustering quality and they are directly dependent to the input data. Because of that, CEDAS requires an expert knowledge about the input data.

Algorithm 5 CEDAS (S, α, λ, r_0)

```
Input: $S$ : the input data stream
Input: $\alpha$ : density threshold
Input: $\lambda$ : decay rate
Input: $r_0$ : micro-cluster radius
 1: Initialize the microcluster structure
 2: loop
 3:     Read next data instance $x$ from data stream $S$
 4:     $dis_{min}$ = Find the distance from $x$ to the nearest micro-cluster center
 5:     if $dis_{min} < r_0$ then
 6:         Add $x$ to the nearest micro-cluster
 7:         Energy of the updated micro-cluster = 1
 8:     else
 9:         Create new micro-cluster with $x$
10:         Energy of the new micro-cluster = 1
11:     end if
12:     Reduce energy of all micro-clusters by $\lambda$
13:     Remove negative energy micro-clusters
14:     if micro-clusters are changed then
15:         Update graph structure with micro-clusters that surpass $\alpha$
16:     end if
17: end loop
```

Evaluation: CEDAS is tested with a data stream consisting of two Mackey-Glass time series, to see how it deals with concept drift, cluster separation, cluster merging and noise over time. Moreover, it is compared to CluStream and DenStream according to complexity, processing speed, cluster quality and memory efficiency. CEDAS, CluStream and DenStream are also compared with high dimensional data according to speed and accuracy. CEDAS successfully deals with concept drift. Noise negatively affects the clustering quality, however results are claimed to be still acceptable. Time measurements show that CEDAS is quite suitable for high dimensional data. Firstly CEDAS is compared against only online phases of DenStream and CluStream. For data with less than 10 dimensions, CEDAS is the slowest one. However, processing time of CEDAS stays nearly constant up to 10,000 dimensions. CluStream becomes slower than CEDAS after 10 dimensions and it consumes nearly 300 times more than CEDAS for 6,000 dimensions. DenStream is faster than CEDAS up to 200 dimensions. For more than 200 dimension, DenStream becomes slower than CEDAS and consumes nearly 2 times more than CEDAS for 6,000 dimensions. After that, CluStream and DenStream are run with a frequent offline phase, to generate near real time final clustering. In this situation CEDAS is the fastest algorithm for both low and high dimensional data. For 5 dimensional data, DenStream consumes 40 times and CluStream consumes 75 times more than CEDAS. For very high dimensional data, time consumption of DenStream grows faster than the others. When the data dimension is 3,000 CluStream consumes nearly 100 times and DenStream consumes nearly 650 times more than CEDAS. The other main advantage of CEDAS is memory efficiency. During the execution, DenStream reaches up to 800 micro-clusters at certain times, while CEDAS reaches up to 100 micro-clusters.

Complexity Analysis: For each new data instance, CEDAS performs a linear search on the micro-clusters. Complexity of this linear search is $O(c)$ where c is the number of micro-clusters. After that, energy of each micro-cluster is reduced, which also requires an $O(c)$ complexity. The last step, which updates the graph structure, is executed only when a new micro-cluster is created or removed. In worst case, all micro-clusters are visited, so worst case time complexity of this step is again $O(c)$. Therefore, the overall time complexity of CEDAS is $O(c)$.

2.3.5 Improved Data Stream Clustering Algorithm (2017)

Improved data stream clustering algorithm [32] is a two phase, density based algorithm that is suitable for arbitrary shaped clusters. Main characteristic of this algorithm is adjusting threshold values automatically, according to the input data. This feature gets rid of the requirement of expert knowledge about the input data.

Algorithm 6 Improved data stream clustering online phase (S, l, λ)

```
Input: $S$ : the input data stream
Input: $l$ : length of data sequence used for initialization
Input: $\lambda$ : decay rate
 1: % Initialization phase
 2: Run DBSCAN on $l$ number of data instances
 3: % Continuous clustering phase
 4: loop
 5:     Read next data instance $x$ from data stream $S$
 6:     Add $x$ to the nearest major micro-cluster OR
 7:     Add $x$ to the nearest critical micro-cluster OR
 8:     Create a new micro-cluster with $x$
 9:     if It is pruning period then
10:         Remove low weighted major micro-clusters
11:         Remove low weighted critical micro-clusters
12:     end if
13: end loop
```

Algorithm 6 shows the main flow of online phase of improved data stream clustering algorithm. DBSCAN algorithm is assumed to be already implemented. The algorithm is composed of two main phases, which are *initialization phase* and *continuous clustering phase*. In the initialization phase, l number of data instances are accumulated and clustered using DBSCAN, at line 2. Major micro-clusters and critical micro-clusters are created as output of DBSCAN algorithm. Major micro-clusters have high densities and will be included in the final clustering process. Critical micro-clusters have low densities and treated as potential outliers. In the continuous clustering phase, when a new data instance is received, it is tried to be added to the nearest major micro-cluster, at line 6. If nearest major micro-cluster is not suitable, this time the new data instance is tried to be added to the nearest critical micro-cluster, at line 7. If neither of them is suitable, a new micro-cluster is created with the new data instance, at line 8. Damped window model (see Section 2.2.3.1) is used and low weighted major and critical micro-clusters are removed periodically, at line 10 and line 11 respectively. Threshold values of major and critical micro-clusters are global parameters in the algorithm, instead of being specific to each micro-cluster. However they are dynamic parameters and continuously updated during the execution.

Algorithm 7 Improved data stream clustering offline phase (micro-clusters)

```
Input: micro-clusters
1: Mark all $mc$s as unvisited
2: repeat
3:     Randomly choose an unvisited $mc$, called $mc_p$
4:     if $mc_p$ is major micro-cluster then
5:         Find all micro-clusters density reachable to $mc_p$
6:         Create a final cluster by them.
7:     else if $mc_p$ is critical micro-cluster then
8:         Continue the next cycle
9:     end if
10: until All $mc$s are visited
```

Algorithm 7 shows the main flow of offline phase of improved data stream clustering algorithm. Initially all micro-clusters are marked as unvisited, at line 1. After that, inside a loop, an unvisited micro-cluster is chosen randomly at line 3. If the selected micro-cluster is a major micro-cluster, all micro-clusters that are density reachable to this micro-cluster are found and a new final cluster is created by them, at line 5 and line 6. If the selected micro-cluster is a critical micro-cluster, then the execution continues with the next cycle, at line 8. When all micro-clusters are visited, the offline phase completes. The term density reachable is defined as follows. If the distance between a micro-cluster and another major micro-cluster is less than or equal to the sum of their radii, then they are directly density reachable. If any adjacent two clusters in a set of micro-clusters are directly density reachable, then the set of micro-clusters is density reachable [32].

Evaluation: Improved data stream clustering algorithm is evaluated against DenStream algorithm, using the network intrusion detection dataset. Clustering quality improvement of the improved data stream clustering algorithm is 2% to 7% with respect to DenStream. Moreover, Yin et al. [32] claims that this algorithm has a better time and spatial complexity, compared with traditional clustering algorithms, however no measurement results are shared.

Complexity Analysis: Let l be the length of the initial data sequence. Complexity of the initialization equals to complexity of DBSCAN, which is $O(l \cdot log\ l)$ in average and $O(l^2)$ in worst case. In the continuous clustering phase, a linear search is performed on micro-clusters for each new data instance. Complexity of this linear search is $O(c)$ where c is the number of micro-clusters. When it is pruning period, pruning task is executed for each micro-cluster one by one and this also requires a complexity of $O(c)$. Therefore, the total worst case complexity is $O(c)$ + $O(c)$, which equals to $O(c)$.

2.3.6 DBIECM (2017)

DBIECM [33] is an online, distance based, evolving data stream clustering algorithm. DBIECM is the only example of distance based clustering algorithms in this survey. DBIECM is an improved version of Evolving Clustering Method (ECM) [101]. Davies Bouldin Index (DBI) is used as the evaluation criteria, instead of shortest distance.

Algorithm 8 shows the main flow of DBIECM. When a new data instance x is received, an attempt is made to add the new data instance to an existing cluster. For this purpose, the distances between x and all clusters are calculated. If radius of any cluster is greater than or equal to its distance to x, then x is added to this cluster, as indicated at line 6. If the distance from x to any cluster is greater than maximum cluster radius r_0, which is a user defined, static parameter, then a new cluster is created with x, at line 8. Otherwise, if there exist any clusters such that their radii are less than their distance to x, then x is added to all of these clusters one by one and DBI of the results are calculated separately. x is added to the cluster that gives the least DBI, which means the best clustering.

Algorithm 8 DBIECM (S, r_0)

```
Input: $S$ : the input data stream
Input: $r_0$ : max cluster radius
1:  Initialize the cluster structure
2:  loop
3:      Read next data instance $x$ from data stream $S$
4:      $dis_i$ = Find the distance from $x$ to all cluster centers $C_i$, $i \in [1, k]$
5:      if $dis_i <$ radius of $C_i$ then
6:          Add $x$ to $C_i$
7:      else if $dis_i > r_0$ for all $i \in [1, k]$ then
8:          Create new micro-cluster with $x$
9:      else % There exist clusters such that radius of $C_i < dis_i < r_0$
10:         Find all clusters such that radius of $C_i < dis_i$
11:         Add $x$ to the best cluster, according to DBI
12:     end if
13: end loop
```

DBIECM requires the maximum cluster radius as a parameter. This parameter directly affects the final cluster count and consequently the clustering quality. Maximum cluster radius strongly depends on the input data and requires an expert knowledge about the data. Being distance based, DBIECM can detect only hyper-spherical clusters. DBIECM does not employ any time window model, thus no input data instance out dates, all input data exist in the final clustering. Moreover, no outlier detection mechanism is implemented. However, it is possible to specify an outlier threshold value and mark the clusters with low cardinality as outliers.

Evaluation: DBIECM is evaluated against ECM, with Iris, Wine, Seeds, Glass and Breast Cancer datasets, from UCI machine learning database. Both of the algorithms are run with the same maximum cluster radius parameter. Firstly, three different radius values are tried, and their direct impact on the resultant cluster number is observed. This shows the importance of the expert knowledge for radius selection. Moreover, clustering quality is compared according to objective function value, DBI, accuracy and purity. For these tests, radius value is selected according to the correct cluster number. DBIECM achieve up to 43% better DBI, up to 33% better accuracy and up to 11% better purity values than ECM.

Complexity Analysis: When a new data instance is received, a linear search is performed on clusters. Complexity of this linear search is $O(k)$. Pairwise distances between all clusters are used for DBI calculation, thus DBI calculation requires a complexity proportional to $O(k^2)$. When there exist more than one candidate clusters for the new data instance, the instance is added to all of them one by one and DBI is calculated accordingly. This requires a complexity proportional to $O(k^3)$. Therefore, although the average complexity of DBIECM depends on the input data, the total worst case complexity is $O(k)$ + $O(k^3)$ which equals to $O(k^3)$.

2.3.7 I-HASTREAM (2015)

I-HASTREAM [34, 35] is a two phase, adaptive, density based hierarchical, data stream clustering algorithm. I-HASTREAM is an improved version of HASTREAM [102]. In the online phase, synopsis of the data is created as micro-clusters. In the offline phase, micro-clusters are maintained in a graph structure as a minimum spanning tree and hierarchical clustering is employed for the final clustering. Main contributions of I-HASTREAM are to perform the final clustering on a minimum spanning tree and to incrementally update the minimum spanning tree according to the changes in the micro-clusters, instead of generating it from scratch. Both of these contributions are related to the offline phase. For I-HASTREAM and its ancestor HASTREAM no algorithmic details are specified about the online phase, instead, it is stated that any micro-cluster model can be employed. For evaluation purpose, HASTREAM employs online phases of *DenStream* and *ClusTree* algorithms and these results are presented by Hassani et al. [102].

Algorithm 9 I-HASTREAM offline phase (micro-clusters, α)

```
Input: micro-clusters
Input: $\alpha$ : weight threshold
1: $MST$ = Update minimum spanning tree($MST$, micro-clusters)
2: $HC$ = Employ hierarchical clustering($MST$, $\alpha$)
3: Extract final clustering($HC$)
```

Algorithm 9 shows main flow of offline phase of I-HASTREAM. The minimum spanning tree is updated according to the changes in the micro-clusters at line 1, and a hierarchical clustering on the minimum spanning tree is employed at line 2. As result of hierarchical clustering, a dendrogram is created. Final clustering is performed according to this dendrogram, at line 3.

Evaluation: Four variants of I-HASTREAM (with different parameters) are evaluated against HASTREAM, MR-Stream and DenStream, using network intrusion detection dataset and the physiological dataset. Purity and Cluster Mapping Measure (CMM) [103] are used as evaluation criteria. One of the I-HASTREAM variants gives up to 25% better purity values than DenStream in network intrusion detection dataset. Its result is also up to 10% better than other versions of I-HASTREAM and HASTREAM. In the physiological dataset, the same variant of I-HASTREAM gives the best CMM and purity values in general. HASTREAM and I-HASTREAM have very close CMM values and both of them outperforms DenStream with up to 30% better CMM values. For purity, again I-HASTREAM has the best values in general and it outperforms both DenStream and MR-Stream with up to 15% better purity values. When we look at the execution time comparison of the algorithms, I-HASTREAM is more than five times faster than DenStream.

Complexity Analysis: Because no algorithmic details are specified about the online phase, we could not analyze complexity of I-HASTREAM.

2.4 Comparison of the Algorithms

As common characteristics of seven data stream clustering algorithms given in Section 2.3, all of them predict number of clusters themselves and they are all able to adopt concept drift in the data streams. All but MuDi-Stream are suitable for high dimensional data. The reason MuDi-Stream is not suitable for high dimensional data is that, it uses a grid based approach for outlier detection. When the data are high dimensional, the number of empty grids increases and the execution time gets higher.

Adaptive Streaming k-means and FEAC-Stream are both k-means based (partitioning based) algorithms. DBIECM is distance based and the others are density based

Table 2.2: Comparison of recent data stream clustering algorithms.

Algorithm	Year	Base Algorithm	Phases	Window Model	Cluster Count	Cluster Shape
Adaptive Streaming k-Means	2017	Partitioning based	Online	Sliding	Auto	Hyper-spherical
FEAC-Stream	2017	Partitioning based	Online	Damped	Auto	Hyper-spherical
MuDi-Stream	2016	Density based	Online-offline	Damped	Auto	Arbitrary
CEDAS	2016	Density based	Online	Damped	Auto	Arbitrary
Improved Data Stream Clustering	2017	Density based	Online-offline	Damped	Auto	Arbitrary
DBIECM	2017	Distance based	Online	None	Auto	Hyper-spherical
I-HASTREAM	2015	Density based	Online-offline	Damped	Auto	Arbitrary

Table 2.3: Comparison of recent data stream clustering algorithms (continued from Table 2.2).

Algorithm	**Multi Density Clusters**	**High Dimensional Data**	**Outlier Detection**	**Drift Adaption**	**Expert Knowledge**
Adaptive Streaming k-Means	Yes	Suitable	No	Yes	No
FEAC-Stream	Yes	Suitable	Yes	Yes	No
MuDi-Stream	Yes	Not suitable	Yes	Yes	Required
CEDAS	No	Suitable	Yes	Yes	Required
Improved Data Stream Clustering	No	Suitable	Yes	Yes	No
DBIECM	Yes (not multi size)	Suitable	No	Yes	Required
I-HASTREAM	Yes	Suitable	Yes	Yes	No

algorithms. Distance based approaches are similar to density based approaches, however they do not have a density threshold, instead they have maximum cluster radius threshold.

In general, density based algorithms have problem about finding clusters with different densities, because of the static density threshold. However, MuDi-Stream and I-HASTREAM have improvements for this problem and they successfully adopt the density threshold to each cluster separately. This makes them able to find multi-density clusters. Adaptive Streaming k-means and FEAC-Stream, being partition based algorithms, are also able to find clusters with different densities. DBIECM is successful for multi density clusters, but not for multi size clusters. It has a static maximum cluster radius threshold and this is a problem for clusters with different sizes. As a result, CEDAS and Improved Data Stream Clustering algorithm are not able to find multi density clusters, but the others are. Furthermore, all density based algorithms are able to find arbitrary shaped clusters, while partitioning and distance based algorithms are limited with hyper-spherical clusters.

For Adaptive Streaming k-means and DBIECM, no outlier detection mechanism is mentioned. However, it is possible to define an outlier threshold and to mark the clusters have less cardinality than the threshold as outliers, for both algorithms. The other algorithms already have outlier detection mechanisms.

Up to the recent years, most of data stream clustering algorithms were online-offline algorithms. A synopsis of the data is employed in the online phase and the final clusters are generated in the offline phase. In this type of algorithms, offline phase is executed periodically or upon user request. Therefore, final clustering results are obtained with a latency and they are not up to date most of the times. However, there exist several recent fully online algorithms in the literature. Fully online algorithms maintain the final clustering results up to date. Therefore, users get the results with no latency. CEDAS, Adaptive Streaming k-means, FEAC-Stream and DBIECM are online algorithms, while MuDi-Stream, Improved Data Stream Clustering and I-HASTREAM are online-offline algorithms.

Damped window model is the most popular time window model among data stream clustering algorithms. On the other hand, DBIECM does not use any time window model. Moreover, Adaptive Streaming k-means uses sliding window model. All other mentioned algorithms use damped window model.

Finally, clustering quality of MuDi-Stream, CEDAS and DBIECM is strongly sensitive to the input parameter *threshold* value. It directly affects the number of clusters and accordingly the clustering quality. Selecting a proper threshold value requires an expert knowledge about the input data. Therefore, for successful results of MuDi-Stream, CEDAS and DBIECM, it is necessary to have prior information about characteristics of the input data. Table 2.2 and Table 2.3 show the comparison summary of examined data stream clustering algorithms and Figure 2.3 shows their main characteristics.

In conclusion, Adaptive Streaming k-means, FEAC-Stream and DBIECM have limitations about the cluster shape; they are able to find only hyper-spherical clusters. MuDi-Stream is not suitable for high dimensional data because of its grid based outlier detection mechanism. CEDAS and Improved Data Stream Clustering algorithm cannot be used for clusters with different densities and DBIECM cannot be used for clusters with different radii. Finally, an expert knowledge about the input data and the clusters is required for MuDi-Stream, CEDAS and DBIECM. I-HASTREAM claims to have no limitations, however no algorithmic details are specified for online phase of it. It is stated that online phases of *DenStream* and *ClusTree* are employed instead.

2.5 Open Problems

There exist several open problems about data stream clustering. Here, we indicate the most notable open problems and describe them briefly.

- **Finding k:** Finding k is still an open problem, especially for partitioning based algorithms. There exist some recent methods for this purpose, however none of them is widely accepted and well matured yet. For density based algorithms, determining k is easier, however parameters that depend on domain knowledge are necessary. If cluster characteristics such as density and minimum allowable gap between clusters are known *a priori*, current algorithms are then able to detect k; however, in most cases, knowledge about input data is not available before the execution and it may not be possible to specify parameters that are valid for all clusters. For example, multi-density clusters require different density thresholds and multi-size clusters require different distance thresholds. Determining such parameters is another open problem by itself. Moreover, concept drift, which may invalidate data specific parameters, is very common in data streams. Therefore, finding a k estimation method that adopts to changes in both k and cluster characteristics is a challenge. Such a method should react to concept drift fast, adopt the new data distribution with minimum quality loss and estimate k.

- **Parameter Requirements:** Current data stream clustering algorithms require parameters such as k, density threshold, distance threshold, decay rate and window length. Such parameters are very sensitive to the input data and they directly affect the clustering quality. It is a challenge to automatically specify these parameters without domain knowledge, manage them for each cluster separately, and update them according to the data characteristics.

- **Evaluation Criteria:** There is no *de facto* evaluation criteria for data stream clustering. Traditional evaluation methods are used for stream clustering results. Defining a new evaluation metric that is suitable for data streams might contribute to this field and inspire interest.

- **Benchmark Data:** There is a lack of high quality benchmark data to use in data stream clustering algorithms. One of the most popular datasets for stream clustering is the forest cover type dataset and it is not even a stream data. Synthetic and real world datasets that include concept drift, outliers and class labels, are necessary for benchmarking purposes in data stream clustering field. Generating and collecting such synthetic and real world stream datasets and popularizing them is a challenge.

- **Experimental Comparison Environment:** There is not a system that runs more than one data stream clustering algorithms at the same time, feeds them in the same way, and compares their execution performance and clustering quality.

- **Different Data Types:** Handling different data types is another challenging task in data stream clustering. Most of the stream clustering algorithms work with quantitative features and define the similarity based on euclidean distance. Current data structures that keep the data synopsis are also specialized for quantitative features. There exists a lack of clustering algorithms that work with categorical data. It is common to convert categorical data to quantitative data and use existing algorithms.

- **Performance Improvements:** Any performance improvements is always welcome, since the number of connected devices is increasing and the data generated by them are scaling up and accelerating every day. This situation requires a continuous performance improvement in data stream clustering algorithms. It is possible to improve the performance by using *parallel programming* and *edge computing*. However in this study, we focus on processing where the whole data is gathered and processed directly on a single processor.

Concept drift is a data stream specific and it generates several challenges. The number of clusters, cluster densities, sizes and shapes may change over time due to concept drift. The problems of traditional clustering become continuous problems for stream clustering.

2.6 Popular Data Repositories and Datasets

2.6.1 Data Repositories

There exist several stream data resources on the internet. Moreover, it is common to use traditional datasets as streams or to generate synthetic data streams. Traditional datasets are generally read by order and treated as streams for testing and benchmarking purposes. We mention the stream data sources in this section. Data streams in Stream Data Mining Repository (see Section 2.6.1.4) and MOA (see Section 2.6.1.5) already have true class labels. However, Citi Bike System Data (see Section 2.6.1.1) does not possess explicitly a class label. One should decide how to employ the data and then assign accordingly the class labels. Moreover, National Weather Service Public Alerts (see Section 2.6.1.3) and Meetup RSVP Stream (see Section 2.6.1.2) have several features that can be used as class labels.

2.6.1.1 Citi Bike System Data

Citi Bike NYC [104] is a public bicycle sharing system. It is composed of 750 stations and 12,000 bikes. Citi Bike publicly publishes real time system data in [105] which includes system information, station information, free bike status etc. in a json structure. Moreover, Citi Bike also publishes trip histories, daily ridership and membership data, and monthly operating reports stored as data streams.

2.6.1.2 Meetup RSVP Stream

Meetup [106] is a website providing membership software, allowing its users to schedule events using a common platform. Meetup has an invitation response mechanism in which the invitees click to RSVP button and enter their responses. Meetup publicly publishes these RSVP responses as a stream [107], which is suitable for data stream clustering.

2.6.1.3 National Weather Service Public Alerts

National Weather Service (NWS) [108] creates public alerts, watches, warnings, advisories, and other similar products in the Common Alerting Protocol (CAP) and Atom Syndication Format (ATOM) [109]. These are data streams and they can be used for data stream clustering studies.

2.6.1.4 Stream Data Mining Repository

Stream Data Mining Repository [110] is a public repository holding four different stream datasets, which are Sensor Stream (2,219,803 instances, 5 features, and 54 clusters), Power Supply Stream (29,928 instances, 2 features, and 24 clusters), Network Intrusion Detection 10% Subset (494,021 instances, 41 features, and 23 clusters) and Hyper Plane Stream (100,000 instances, 10 features, and 5 clusters).

2.6.1.5 MOA

Massive Online Analysis (MOA) [111] [112] is a popular open source framework for data stream mining. MOA includes 4 different datasets which are suitable for data stream processing. Moreover, it also includes a number of classes to generate synthetic data streams. There exist several studies in the literature that use MOA as a data source. More information about MOA is available in Section 2.7.1 and synthetic data stream generation classes of MOA are listed in Section 2.6.2.1.

2.6.1.6 Other Repositories

Some other data repositories are listed here.

- Real World Data in Real Time API : `https://www.hooksdata.io/`
- New York City Open Data : `https://opendata.cityofnewyork.us/`

- Registry of Open Data on AWS : `https://registry.opendata.aws/`
- Twitter Data : `https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data`
- AirNow Air Quality Observations : `https://docs.airnowapi.org/`
- National Wind Technology Center (NWTC) : `https://data.nrel.gov/submissions/33`
- Solar Radiation Research Laboratory (SRRL) : `https://data.nrel.gov/submissions/7`
- Awesome Public Datasets : `https://github.com/awesomedata/awesome-public-datasets`

2.6.2 Popular Datasets

It is very common to use synthetic datasets in data stream clustering for both testing and benchmark purposes. Synthetic datasets give the user opportunity to specify the stream properties such as noise ratio, concept drift, cluster shapes and densities. Synthetic data stream generation by MOA and details of popular datasets are given. All datasets mentioned in this section, except Charitable Donation Dataset, have true class labels. Table 2.4 summarizes properties of popular datasets.

2.6.2.1 Synthetic Data Streams

Massive Online Analysis (MOA) [111] (described in Section 2.7.1) has a number of classes [113] to generate synthetic data streams in different shapes and with or without concept drift.

2.6.2.2 Forest Cover Type Dataset

Forest Cover Type Dataset is publicly available on Machine Learning Repository of UCI. It has totally 581,012 instances and each of them belongs to one of 7 cover

Table 2.4: Properties of popular datasets.

Dataset Name	Number of Instances	Number of Features	Number of Clusters
Forest Cover Type	581,012	54	7
Network Intrusion Detection	4,898,431	41	23
Network Intrusion Detection Subset	494,021	41	23
Charitable Donation	191,779	481	*not specified*
Sensor Stream	2,219,803	5	54
Power Supply Stream	29,928	2	24
Hyper Plane Stream	100,000	10	5

types. The instances are described by 54 features, 10 of which are quantitative and 44 of which are binary. Each instance is giving information of an area of 30x30 meters. This dataset is not actually a data stream, but a stationary dataset. It does not have a time stamp or an exclusive order information. However, it is converted into a data stream by taking the data input order as the streaming order.

2.6.2.3 Network Intrusion Detection Dataset

Network Intrusion Detection Dataset is used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was a session of KDD-99, The Fifth International Conference on Knowledge Discovery and Data Mining. It is publicly available on KDD archive of UCI. This set has 4,898,431 records of network traffic data and each of them belongs to one of 23 types of connection (22 attack types and normal connection). The instances are described by 41 features, some of which are discrete and the others are continuous. There exists also a 10% subset of this dataset which is more concentrated than the original dataset. The subset itself is yet another most used dataset.

2.6.2.4 Charitable Donation Dataset

Charitable Donation Dataset is used for The Second International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-98, The Fourth International Conference on Knowledge Discovery and Data Mining. This dataset has 191,779 instances and each instance has 481 features. These instances, are information about people who have made charitable donations in response to direct mailing requests. This dataset is publicly available on KDD archive of UCI.

2.6.2.5 Various Spam Mail Datasets

There exist several spam mail datasets publicly available in different online data repositories. Spam mail datasets are suitable for stream clustering because mails inherently are data streams. They have a date-time information which makes them easily interpreted as data streams.

2.6.2.6 Various Sensor Network Datasets

There exist several sensor network datasets publicly available on the Internet. One of sensor network data repositories is [114]. It is very common to use sensor network datasets in data stream clustering, since they inherently are data streams.

2.7 Data Stream Processing Tools

We provide brief information about popular tools that are used for data stream mining.

2.7.1 MOA

Massive Online Analysis (MOA) [111] [112] is a popular open source framework for data stream mining. It is implemented in Java and released under the GNU General Public License. MOA is specialized for data streams. It includes algorithms for regression, clustering, classification, outlier detection, concept drift detection and recommender systems, and it also includes tools for evaluation. Data stream generators are provided. It can be used as both a stream processing tool and an environment to develop stream processing algorithms. Furthermore, MOA has the ability to interact with Waikato Environment for Knowledge Analysis [115], which is a data mining software.

2.7.2 RapidMiner

Rapid Miner [116], formerly known as Yet Another Learning Environment (YALE), is another data mining tool but it is developed by a private company. It has an integrated development environment, which is called RapidMiner Studio. It supports all data preparation, result visualization, model validation and optimization steps of the machine learning process. It has a *Streams* plugin [117] which integrates the stream oriented processing into the RapidMiner suite. This plugin allows developing data stream processing tools using utilities of RapidMiner.

2.7.3 R

R [118] is a free software environment and programming language for statistical computing. R is an open source project and it is released under the GNU General Public License. R, a rich in packages software environment, has special packages for clustering, data streams, stream mining etc. These packages are as follows.

- stream: A framework for data stream modeling and associated data mining tasks such as clustering and classification.
- rstream: Unified object oriented interface for multiple independent streams of random numbers from different sources.
- streamMOA: Interface for data stream clustering algorithms implemented in the MOA framework.
- RMOA: Connects R with MOA framework to build classification and regression models on streaming data.

2.8 Data Stream Processing Platforms

Currently there exist several data stream mining platforms [119, 120] developed by different organizations.

- **Apache Storm** [121] is a distributed, real time stream processing computation framework. It is free and open source. Moreover, Apache Storm is scalable and fault tolerant. It is designed to be used with any programming language.

- **Apache Spark** [122] is a well known, open source, fast and general engine for large-scale data processing. Apache Spark has an extension, called **Spark Streaming** [123], that enables scalable, high-throughput, fault tolerant stream processing of live data streams. Spark Streaming can be seen as a layer between data streams and Apache Spark. Spark Streaming gets a data stream, creates data batches from the stream and feeds Apache Spark with these batches. In this way, results of the data stream processing are produced by Apache Spark batch by batch. Spark Stream accepts input from many different sources such as Kafka, Flume, Twitter, ZeroMQ, Kinesis, or TCP sockets.

- **Apache Samza** [124] [125] is another open source, distributed stream processing framework. It is near real time and asynchronous. It provides fault tolerance, processor isolation, security, and resource management using Apache Hadoop Yarn. It uses Apache Kafka for messaging. Apache Samza, together with Apache Kafka, is developed by LinkedIn engineers, and commonly known as LinkedIn's framework for stream processing.

- **Apache Kafka** [126] is an open source stream processing software platform. The objective of the project is to provide a unified, high throughput, low latency platform for real time data streams. It is scalable and fault tolerant. It has a publish-subscribe messaging system. Apache Kafka is the other platform developed by LinkedIn, similar to Apache Samza.

- **Amazon Kinesis** [127] is one of the Amazon web services. It is a cloud based, real time data processing service that is developed for large and distributed data streams. In functionality, Amazon Kinesis has similarities to Apache Kafka. It is scalable and able to pull any amount of data, from any number of sources. It is designed to make it easier to develop real time applications and it has a fully managed infrastructure.

- **IBM Infosphere** [128] [129] is a commercial, enterprise-grade stream processing platform, that is designed to retrieve meaningful information from data in motion, working on time window models with windows of minutes to hours. It provides low latency for time critical applications such as fraud detection and network management. It also has the ability to fuse streams. IBM Inforsphere adapts rapidly to changing data forms and types and it manages high availability itself.
- **Google Cloud Stream** [130] is Google's solution for data stream processing. It has a fully managed infrastructure and it provides ingesting, processing and analyzing event streams in real time. It is an integrated, scalable and open stream analytics solution. Google Cloud Stream works with a full harmony with other solutions of Google Cloud, like Cloud Pub/Sub, Cloud Dataflow, BigQuery, Cloud Machine Learning etc.
- **Microsoft Azure Stream Analytics** [131] is Microsoft's solution for data stream processing. It is a serverless, scalable, on demand real time, complex event processing engine. It is able to run on multiple streams from different sources. Azure Stream Analytics has a declarative SQL like language. It can be used as integrated with other Azure solutions such as Azure Machine Learning, Azure IoT Hub, Power BI etc.

2.9 Conclusions

With the technological improvements, number of interconnected devices is increasing. Connected devices continuously generate large scale data with high speed, which are called data streams. Therefore, processing data streams in real time is arousing more interest and clustering seems to be the most suitable data processing method for data streams.

We present a survey of recent progress in data stream clustering algorithms. There are essential differences between traditional data clustering algorithms and data stream clustering algorithms. We emphasize the most important data stream clustering concepts such as concept drift, window models, outlier detection methods and data structures. Seven most recent data stream clustering algorithms are analyzed in detail. For each algorithm, a comprehensive analysis is presented including algorithmic detail, evaluation of the results and complexity. Global comparison of these algorithms highlighting their advantages and disadvantages is also presented. An overview of the most popular stream processing tools and platforms is given along with stream datasets.

Several open challenges exist regarding data stream clustering. Finding number of clusters and adopting to changes in the number of clusters in data streams are the most crucial challenges. Furthermore, existing algorithms need critical parameters that directly affect clustering quality and require prior knowledge about input data. Moreover, concept drift may change data characteristics and invalidate these parameters. Developing generic and self-adapting algorithms is another popular data stream clustering challenge. Additionally, there is a lack of algorithms that handle different data types. Most of existing algorithms are able to deal with only quantitative data. Last but not least, data stream clustering algorithms should execute with high performance in despite of memory restrictions.

It may be ideal to compare the efficiency and the effectiveness of the data stream clustering algorithms on a benchmarking framework under controlled conditions of synthetic datasets that contain concept drift, outliers and class labels and of real world datasets. Data stream clustering using deep neural network models and within edge computing are the two emerging topics to be explored further.

CHAPTER 3

DATASETS

The proposed methods EmCStream and NoCStream are evaluated using both synthetic and real world stream datasets having various characteristics. In total, sixteen synthetic and nine real world datasets are used. This chapter is devoted to the datasets considered in this study.

3.1 Synthetic Datasets

We have created high dimensional, evolving, synthetic data streams by *DSD_RandomRBFGeneratorEvents* function of streamMOA [132] package of R [133]. This function is able to create evolving data streams with the defined characteristics. It is possible to decide on several parameters, some of which are k, dimensions, drift speed, cluster radius, density, cluster split or merge, noise, etc. We have used the first 50,000 data instances generated by these data streams. We assume each data source in a stream composes a cluster and generates one data instance per second. In this way, in a stream that has ten clusters, ten data instances are generated per second.

Sixteen streams are created in total and each data stream has different characteristics. One of the data streams (Stream-8) is stationary, which means it does not contain concept drift. Other data streams evolve at different speeds. Eleven streams have a constant number of clusters while in five streams the number of clusters changes. In the latter five streams, clusters emerge and disappear as a result of concept drift. Hence, the number of clusters is changing in these streams. Three streams (9 to 11) are created with different levels of noise and the remaining thirteen streams are noiseless. We have specified the properties of the streams in such a way that it is possible to observe the effects of change in k, dimensions, speed of the concept drift, and noise level. Detailed information on the synthetic data streams is given in Table 3.1, where k indicates *number of clusters* and d indicates *dimensions*.

Table 3.1: Characteristics of synthetic data streams. Number of instances is 50,000.

Stream	k	d	**Noise (%)**	**Drift speed**
Stream-1	10	50	0	normal
Stream-2	10	100	0	normal
Stream-3	10	10	0	normal
Stream-4	20	50	0	normal
Stream-5	4	50	0	normal
Stream-6	10	50	0	high speed
Stream-7	10	50	0	low speed
Stream-8	10	50	0	no drift
Stream-9	10	50	5	normal
Stream-10	10	50	10	normal
Stream-11	10	50	20	normal
Stream-12	10 ± 2	10	0	normal
Stream-13	10 ± 2	20	0	normal
Stream-14	10 ± 2	5	0	normal
Stream-15	20 ± 4	10	0	normal
Stream-16	4 ± 1	10	0	normal

3.2 Real World Datasets

3.2.1 Meteorological Datasets

We have composed three real world meteorological stream datasets using weather data from `https://www.renewables.ninja/`. There are three sets of data: meteorological data of two cities from Turkey (Meteo-TR), two cities from Europe (Meteo-EU) and two cities from the US (Meteo-US). For each dataset, we chose two cities that have different climate characteristics to create separable data. All datasets consist of hourly measurements for five years, from Jan 1st, 2015, till Dec 31st, 2019. There exist 43,824 measurements for each city. Every instance includes six features, which are temperature, precipitation, snowfall, snow mass, air density, and cloud cover. Temperature and air density are the two features that are always discriminative while other features are not. For example, precipitation is not a discriminative property when it is not raining, and snowfall or snow mass is not effective when it is not snowing. For this reason, we have defined an extra weight to temperature and air density; that is, the values of these features are multiplied by two.

To create a dataset, we have merged the instances of two cities according to the date and time of the measurement. In this way, the datasets become real-stream datasets. Moreover, these are evolving data streams by nature, since weather data changes both in a day of 24 hours, from daytime to night, and in a year, from season to season. Using such data streams, it is possible to focus either on concept drift that occurs every day or on concept drift that occurs from season to season.

3.2.2 Keystroke Dynamics

We have used four subsets of the larger CMU Keystroke Dynamics - Benchmark Data Set [134] which is created by typing characteristics of 51 users. The participants type the password ".tie5Roanl" and the *Enter* key 400 times, captured in eight different sessions on different days, 50 times on each session. Each instance consists of 31 features. Keystroke datasets incrementally evolve due to the participants' practice. Moreover, an abrupt concept drift is also expected from one session to the next one, because of the loss of practice.

We have created four subsets of the Keystroke dataset, getting all features of some participants. We did not change, eliminate or convert any of the features. Three of the subsets have constant k while k changes in the last one. We have specified k as two, three and four for the first three of the datasets. The last dataset is the concatenation of the first three datasets. Hence, in this last dataset k is two in the first part, three around the middle, and four in the last part. Since each participant has 400 records, the subsets that we have used have 800, 1,200, 1,600, and 3,600 instances respectively.

3.2.3 Sensor Data

We have also used two subsets of the Sensor stream dataset from `http://db.csail.mit.edu/labdata/labdata.html`. Sensor data consists of sensor readings from 54 sensors deployed in the Intel Berkeley Research laboratory. The stream contains about 2.2 million instances, each of which consists of four features, *temperature, humidity, light, voltage* in addition to the sensor ID and reading time. There also exist invalid readings in the stream. Moreover, being in the same laboratory, all sensors produce similar readings. We have created two subsets of this larger stream dataset, which we call Sensor-2 and Sensor-3. In order to create these subsets, we have extracted the readings of two and three sensors accordingly. In Sensor-2, k is two and constant throughout the whole stream. However, in Sensor-3, k is three in the beginning but it changes to two and back to three a few times. We have eliminated the invalid readings and have normalized the data while generating the subsets.

Table 3.2 presents a summary of the characteristics of the real world datasets, all of which are evolving.

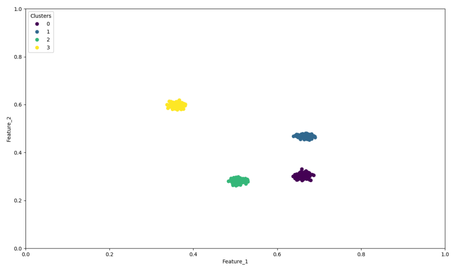
Table 3.2: Characteristics of real world data streams.

Stream	k	d	**Instances**
Meteo-TR	2	6	87,648
Meteo-EU	2	6	87,648
Meteo-US	2	6	87,648
Keystroke-2	2	31	800
Keystroke-3	3	31	1,200
Keystroke-4	4	31	1,600
Keystroke-C	2-4	31	3,600
Sensor-2	2	4	64,200
Sensor-3	2-3	4	108,000

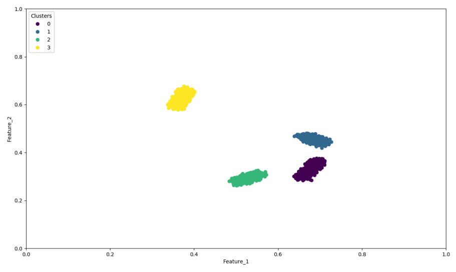
3.3 Drifts in the Datasets

Concept drift is defined as the unforeseen change in the statistical characteristics of the data instances and it is explained in Chapter 2. In order to visualize the concept drift, it is possible to animate the input data in its streaming order. This makes apparent the change on the data over time. To understand better the concept drift in the data we have used, we made visualizations of various numbers of data instances from one of the synthetic data streams and also one of the real data streams. Because the data are high dimensional, we preferred to visualize two features of the synthetic data, and one feature of the real data in time domain. Figure 3.1 shows visualization of parts of Stream-5 in different lengths, and colored according to true labels. Figure 3.1 (a), (b), (c) and (d) show visualizations of first 1,000, 5,000, 10,000 and 20,000 instances of synthetic Stream-5, respectively. According to Figure 3.1, it is easy to observe that properties of the data instances change and the cluster centroids move over time. Because the data stream is synthetic and drift speed is constant, movement of the data seems straight. Moreover, Figure 3.1 (c) and (d) show that, some clusters violate regions of other clusters over time. In other words, data instances belong to different clusters may be present in the same region in different time intervals. However, a data stream clustering algorithm must successfully cluster crossing data instances.

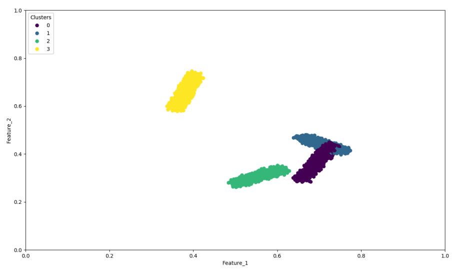
Figure 3.2 shows the visualization of parts of meteorology data of Turkey, in different lengths, and colored according to true labels. Figure 3.2 (a), (b), (c) and (d) show the visualizations of first 1,000, 5,000, 10,000 and 20,000 instances of the stream, starting from Jan 1st, 2015, respectively. Since the data are high dimensional, only temperature feature is visualized over time. As mentioned in Section 3.2, meteorology data of Turkey consist of hourly measurements of two cities from Turkey. This means data of one day includes 48 data instances. Hence, 1,000 data instances means nearly 21 days, 5,000 data instances means 105 days and so on. Figure 3.2 (a) shows the daily night-day temperature difference very clearly, which is also a local concept drift. In (b), a slight rise appears in the data. In (c) a significant temperature rise appears and in (d) the rise is followed by a decrease because the visualization includes more than a full year data. According to Figure 3.2 (c) and (d), instances of opposite clusters are present in the same region at different time intervals, which is caused by the seasonal



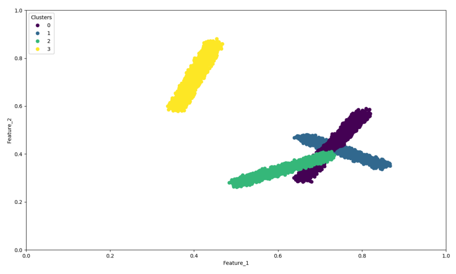
(a) Visualization of 1,000 data instances.



(b) Visualization of 5,000 data instances.

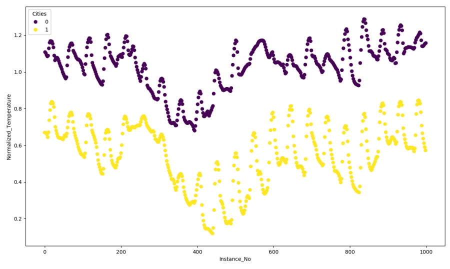


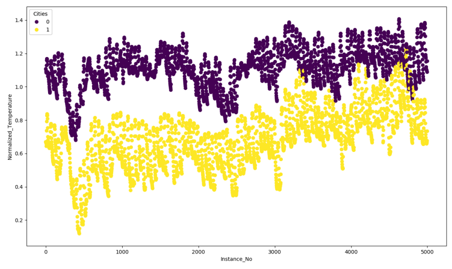
(c) Visualization of 10,000 data instances.

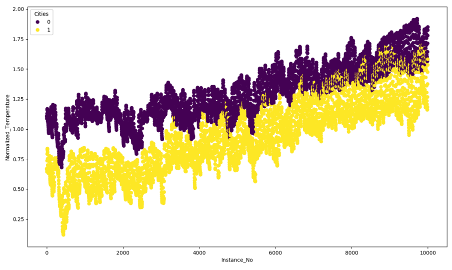


(d) Visualization of 20,000 data instances.

Figure 3.1: Visualization of different number of data instances from the first part of synthetic Stream-5. Only two features of the data are visualized. Properties of the data instances change and the cluster centroids move over time. This change is called concept drift. Samples of three clusters present in the same region in different time intervals, as a result of the concept drift. Because the data stream is synthetic and drift speed is constant, movement of the data seems straight.


(a) Visualization of data of 21 days.


(b) Visualization of data of 105 days.


(c) Visualization of data of 208 days.

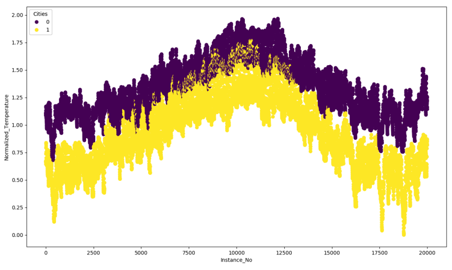

(d) Visualization of data of 417 days.

Figure 3.2: Visualization of different number of data instances from the first part of meteorology data of Turkey. Only temperature feature is visualized over time. (a) shows the daily night-day temperature difference clearly, which is a local concept drift. In (b) a slight rise appears in the data. In (c) a significant seasonal temperature rise appears and in (d) the rise is followed by a decrease because the visualization includes more than a full year data. Seasonal climate change is a concept drift.

climate change. This change is a concept drift.

CHAPTER 4

METHODS

4.1 Online Embedding and Clustering of Evolving Data Streams

Our method, online embedding and clustering of data streams (EmCStream) continuously embeds high dimensional data streams into two dimensions, detects, reports and adapts concept drift and clusters input data instances in real time. EmCStream processes the input data in terms of a window determined by the *horizon*. Horizon is in fact the number of instances inside a window and it is set as an input parameter. EmCStream does not outdate the data instances using a fading function mechanism, instead, it slides the window in order to process each data instance only once. During the embedding of the input data in 2D, EmCStream also checks for a concept drift. When a concept drift is detected, it reports the drift and adapts itself to the current characteristics of the data. After the drift detection and adaptation, EmCStream clusters the data. Implementation of EmCStream is available online at `https://gitlab.com/alaettinzubaroglu/emcstream` with all other supplementary resources.

Algorithm 10 shows the main flow of EmCStream algorithm. EmCStream begins with the initialization phase. After the initialization, it continuously embeds the input data window by window, and periodically checks for a concept drift. When the drift check period is met, EmCStream clusters the data, that are embedded since the previous drift check time, and checks for a concept drift. If no concept drift detected, EmCStream indicates the embedded data and clustering results, then continues to embed the input data until the drift check period is met again. When the algorithm detects a concept drift, it rewinds the data instances to the previous drift check position and rerun the initialization phase. Main flow of EmCStream is also described as a flowchart at Figure 4.1 where S is the input data stream, k is number of clusters and h is horizon.

Algorithm 10 EmCStream (S, k, h)

```
Input: $S$ : the input data stream
Input: $k$ : number of clusters
Input: $h$ : horizon
1: loop
2:     % Initialization phase
3:     Get input data from $S$
4:     Initialize EmCStream
5:     % Embedding and clustering phase
6:     while not drift check period met do
7:         Get next $h$ data instances from $S$
8:         Embed the data
9:     end while
10:    Cluster embedded data as $k$ clusters
11:    Check for a concept drift
12:    if Concept drift detected then
13:        Report the concept drift
14:        Rewind $S$ to the previous drift check position
15:        Continue the loop (Go to Initialization phase)
16:    else
17:        Indicate clustered data
18:        Go to line 6, Embedding and clustering phase
19:    end if
20: end loop
```

EmCStream uses a few parameters during the execution. These parameters are described as follows:

- ***k:*** number of clusters; hyperparameter, determined by the user.
- ***horizon (h):*** number of data instances processed together; hyperparameter,

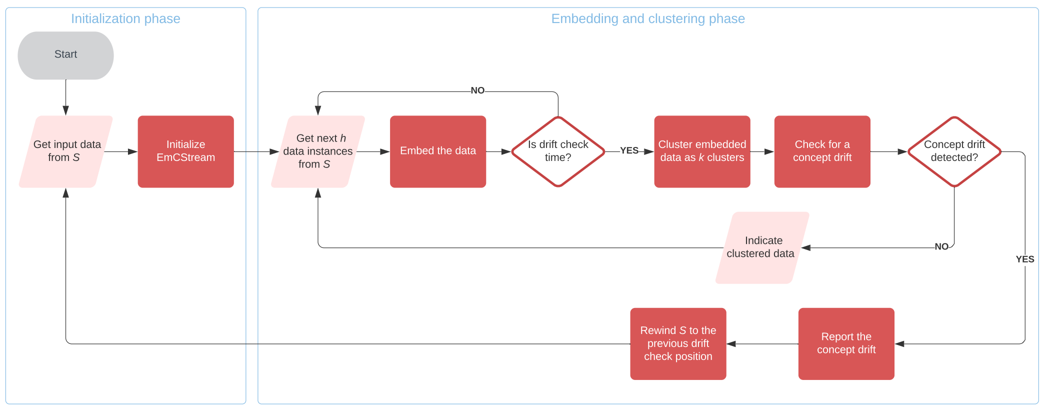


Figure 4.1: Main flow of EmCStream.

determined by the user.

- ***init size:*** number of data instances used for initialization; set to $2 \times horizon$.
- ***drift check period (p):*** number of data instances processed between two consecutive concept drift detections; set to $5 \times horizon$ at the beginning and adjusted in an adaptive manner during the execution.
- ***drift threshold:*** threshold value for adjusted Rand index; used for concept drift decision and adjusted in an adaptive manner during the execution.

EmCStream embeds the input data in 2D by UMAP algorithm. Let

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N\}$$

be the d-dimensional input data stream, where N goes to infinity and $\mathbf{x}_i$ is i^{th} data instance, which is a d-dimensional vector. Let

$$Y = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, ..., \mathbf{y}_N\}$$

be a two-dimensional representation of S, where $\mathbf{y}_i$ is a two-dimensional vector, as a result of UMAP. Let

$$L = \{l_1, l_2, l_3, ..., l_N\}$$

be cluster labels of Y, assigned by k-means algorithm, where l_i is the cluster label of $\mathbf{y}_i$. EmCStream is fed with S as the input data, and outputs Y and L during the execution. However, EmCStream does extra processing for concept drift detection and adaptation.

EmCStream embeds the input data twice, once using the knowledge acquired during the initialization, and another time without any previous knowledge. In the initialization phase of EmCStream, UMAP generates a mapping for *init size* data instances and embeds them. After the initialization phase, new coming data are embedded as groups of *horizon* data instances according to the mapping generated in the initialization phase. This process continues until it is time to check for a concept drift. EmCStream, checks for a concept drift periodically and this period is defined by *drift check period*. Let S_i be the set of data instances that are received between i^{th} and $(i+1)^{th}$ drift check times. To check for a concept drift, EmCStream re-embeds S_i using a new UMAP model by generating a new mapping. Let Y_i^1 be the two-dimensional representation of S_i, generated by UMAP, according to the initialization phase, and Y_i^2 be that is generated by UMAP, according to the input data itself, using no prior knowledge.

In this situation EmCStream has two different embedding outputs (Y_i^1 and Y_i^2) for the same data. When these two embedding results are coherent to each other, this means the data characteristics remain same and there is no concept drift. However, when two embedding results are not consistent, this means the data characteristics have changed and there exists a concept drift. Let L_i^1 be cluster labels of Y_i^1 and L_i^2 be cluster labels of Y_i^2. After generating L_i^1 and L_i^2, EmCStream calculates the consistency (ς) between L_i^1 and L_i^2. $\varsigma = Consistency(L_i^1, L_i^2)$. Let θ be the threshold value for the consistency measure, used for drift decision. When the calculated consistency is less than the threshold value, $\varsigma < \theta$, this situation is accepted as a concept drift and EmCStream rerun the initialization phase. When the calculated consistency is greater than or equal to the threshold value, $\varsigma \geq \theta$, this means the input data characteristics remain same and there is no concept drift. In such a case, EmCStream indicates the clustering results and continues to embed the input data using UMAP by the already generated mapping. When it is time to check for a concept drift again, EmCStream follows the drift check process again. We have used adjusted Rand index as the consistency metric (ς).

EmCStream adaptively adjusts drift check period during the execution. When EmC-Stream detects a drift on the same instances more than once, it shortens the drift check period and checks again. While working with a shortened period, if EmCStream detects no drift twice consecutively, it extends the period until it reaches the initial drift check period. This feature of EmCStream lets it detect concept drift precisely and give more accurate clustering results.

There exist a few characteristics of EmCStream that make it differ from other state of the art stream clustering algorithms. To the best of our knowledge, EmCStream, is the only method that embeds the data before clustering. Moreover, though there exist concept drift detection algorithms in the literature, most of data stream clustering algorithms adapt concept drift without detecting it. Advantages of EmCStream can be listed as follows:

- EmCStream makes visualization of high dimensional input data possible, by embedding them.
- EmCStream reports concept drift.
- EmCStream generates consistent and coherent cluster labels on each window, during the whole execution. Consequently cluster labels can be concatenated.

EmCStream, provides the clustering results periodically for a group of most recent data instances, similar to the other stream clustering algorithms. However, the most important characteristic of EmCStream is that, it gives the cluster labels consistent and coherent during the whole execution and it pursues the clusters during a concept drift, while other clustering algorithms do not give consistent cluster labels on every update. In other algorithms, clustering results on each update are independent from each other, and they cannot be concatenated. EmCStream does an extra operation in order to convert the cluster labels given on each update and make them coherent, while other algorithms do not.

EmCStream is set to use the euclidean distance. EmCStream first processes the data using UMAP and UMAP supports several distance types. Therefore, EmCStream can be easily converted to use a different distance type supported by UMAP or to get the distance type as a hyperparameter.

EmCStream does not detect *k (number of clusters)* itself. It needs k as an input parameter. In order to remedy this shortcoming we present a novel method NoCStrem for determining the optimal number of clusters, in Section 4.2.

4.1.1 Complexity Analysis

EmCStream processes the input data window by window. h (*horizon*) is the window length; that is, the number of data instances processed together. EmCStream checks for a concept drift and indicates the results periodically. This period is the *drift check period (p)* and set to $(5 \times h)$. In a drift check period, EmCStream runs five times UMAP on a data of size h and runs once k-means on a data of size p. For drift check purposes, EmCStream runs once UMAP and k-means again, on the same data of size p. Algorithms run by EmCStream and their repetitions in a drift check period are listed as follows:

- UMAP on a data of size h, 5 times.
- UMAP on a data of size p, once.
- k-means on a data of size p, twice.

The complexities of UMAP and k-means are $O(N^{1.14})$ and $O(Nkdi)$, respectively. Here d is two, because we run k-means algorithm on the embedded, two dimensional data. N is equal to p, N being the number of data instances processed between two consecutive concept drift detections. After removing the scalar multipliers, time complexity of EmCStream is bounded by maximum of $O(N^{1.14})$ and $O(Nki)$. In the worst case, drift check process may be repeated five times and this situation adds a scalar multiplier to the complexity.

4.2 Determining the Optimal Number of Clusters on High Dimensional Evolving Data Streams

4.2.1 Introduction

Thanks to the technological improvements, the number of connected devices is continuously increasing. Connected devices constantly generate data streams and the data streams may often be of high dimension and evolve over time, which is called concept drift. Data generated by connected devices need to be processed in real-time because offline processing of such a huge amount of data requires growing storage capacity and causes delayed analysis. Hence, real-time processing of data streams has become an active research area. Being unsupervised, clustering is one of the most suitable real-time data stream processing methods.

Data clustering is the task of grouping similar data instances together and dissimilar data instances in different groups, according to the properties of the instances. The objective of clustering is to minimize intra-cluster distance and maximize inter-cluster distance. While some of the data stream clustering algorithms predict the number of clusters (k) during the execution, many need the number of clusters to be given by the user as an input parameter. Moreover, the clustering algorithms that predict the number of clusters, need their specific hyperparameter values to be input and the clustering quality is very sensitive to these hyperparameter values.

In most cases, true class labels are not available for data stream instances and there is no prior knowledge about the number of clusters (k). Moreover, the number of clusters may change over time, because of the evolution of data, which is also called concept drift. Hence, real-time and continuous prediction of k is a crucial problem.

Data generated by connected devices are often of high dimension. Moreover, data may contain outliers and concept drift. Concept drift is the unforeseen change in the properties of the input data instances, which is a data stream-specific challenge. A concept drift may also be in the form of new cluster creation, cluster disappearance, or split or merge of clusters. Such concept drifts give rise to changes in the value of k. Hence, for stream data, k should be continuously predicted throughout the whole data stream.

The described method that determines the optimal Number Of Clusters on high dimensional evolving data Streams, NoCStream continuously embeds high dimensional data streams into two dimensions and predicts the number of clusters (k) in real-time.

In this study, we describe a novel method, NoCStream to determine the optimal Number k Of Clusters throughout a data stream. We especially focus on high-dimensional evolving data streams. NoCStream analyzes the data as bunches of data instances, which are also called windows or batches. It is suitable to be used as a pre-processing tool in order to find k, integrated into data stream clustering algorithms that need k as a hyperparameter. NoCStream embeds the high dimensional data into two-dimensional space (projection) using Uniform Manifold Approximation and Projection (UMAP) [4], and then it finds k using mean shift clustering algorithm [135, 136].

4.2.2 Traditional Methods for Predicting the Number of Clusters

Despite several traditional methods to predict the number k of clusters, none is specialized for data streams. In data streams, if data evolve, k may change with time, and changing k must be predicted continuously. Some of the data stream clustering algorithms, such as DenStream [12] and BIRCH [65] predict k implicitly, and others such as CluStream [13] and EmCStream [10] need k to be supplied by the user. In this section, we describe general approaches of popular traditional k prediction methods.

4.2.2.1 Elbow Method

The elbow method [137] is a common heuristic in mathematical optimization to choose the optimal value. In cluster analysis, the elbow method is used to determine the number of clusters, k. The explained variation [138] is plotted as a function of k and the elbow of the curve is picked as the number of clusters to use. If k is selected as the number of data instances, a 100% explained variation is calculated, however, it is obvious that this is not an optimal solution for the prediction of k. The elbow of the curve gives the point that more clusters do not yield a far better value of explained variation. This method is used to choose the optimal value among the considered candidates. Hence, it is important to determine the candidate values. If the correct or best value of k is not among the initial candidates, it is not possible to choose this best value. The curve of the intra-cluster variance, which is a sample of this method, is calculated by Equation 4.1, where $W(C_r)$ is the intra-cluster variance within the r^{th} cluster C_r [139].

$$E(k) = \sum_{r=1}^{k} W(C_r) \tag{4.1}$$

4.2.2.2 Silhouette Method

Silhouette method [140] is another technique that is used to choose the best k among the initial candidates, according to the best silhouette score [141]. The silhouette value of an instance is the measurement of how similar that instance is to the cluster assigned to, compared to other clusters. It is defined in the [-1.0, 1.0] interval and the higher values indicate a better match of the instance to its own cluster. For a dataset of size N, which may be shown as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N\}$, silhouette value of an instance, $S(\mathbf{x}_i)$ is calculated according to the Equation 4.2, where $a(\mathbf{x}_i)$ is the average distance between $\mathbf{x}_i$ and all the instances in the same cluster with $\mathbf{x}_i$; and $b(\mathbf{x}_i)$ is the average distance between $\mathbf{x}_i$ and all the instances in the closest cluster. Silhouette score, $SS(\mathbf{X})$ is the mean of silhouette values of all instances in the dataset, as shown in Equation 4.3. In the silhouette method, the silhouette score is calculated for all candidate k values and the k yields the greatest silhouette score is selected.

$$S(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max(a(\mathbf{x}_i), b(\mathbf{x}_i))} \tag{4.2}$$

$$SS(\mathbf{X}) = \frac{1}{N} \sum_{i=1}^{N} S(\mathbf{x}_i) \tag{4.3}$$

4.2.2.3 Density Based Methods

There exist several density-based data stream clustering algorithms in the literature [36, 58, 17] where DenStream [12] is the most popular algorithm. Density-based data stream clustering algorithms predict k inherently based on the *density reachability* and *density connectivity* concepts [32]. However, these algorithms are very sensitive to hyperparameter values. They need several hyperparameter values to be selected according to the characteristics of the input data and there may occur problems in finding multi-density clusters.

4.2.2.4 Other Methods

The silhouette method is based on the idea of maximizing the silhouette score, which is an internal validation technique for clustering. There exist other internal validation indices that can be used to predict k in a similar manner. Thirty of such indices are examined by Milligan and Cooper [142]. Another popular k prediction technique is the gap statistic method [143]. This method is also based on selecting k that yields the largest gap statistic. The gap statistic is the metric that measures the total within intra-cluster variation with their expected values under the null reference distribution of the data. More recent indices are analyzed by Charrad et al. [144]. Another method of estimating k is introduced by Can and Ozkarahan [145].

4.2.3 Method

The described method that determines the optimal Number Of Clusters on high dimensional evolving data Streams, NoCStream continuously embeds high dimensional data streams into two dimensions and predicts the number of clusters (k) in real-time. NoCStream processes the input data in terms of a window determined by the *horizon (h)*. Horizon is in fact the number of instances processed together inside a window and it is defined as a hyperparameter. NoCStream uses non-overlapping sliding windows to process each data instance only once. The proposed method embeds the high dimensional data into 2D using Uniform Manifold Approximation and Projection (UMAP) [4] continuously, and then it finds k by the aid of mean shift clustering algorithm. Several studies in the literature show that UMAP is the better choice among its alternatives [5, 6]. In this section, we first give a sketch of the mean shift algorithm as background information. We then describe the method in detail.

4.2.3.1 Mean Shift Algorithm

Mean shift [135, 136] is a centroid-based clustering algorithm commonly used in image processing and computer vision. It is also known as the mode-seeking algorithm. Mean shift works iteratively. At each iteration, mean shift calculates *regional* means as cluster centroids, it moves each instance towards the nearest centroid and recalculates the centroid, which is again the regional mean. With this procedure, the centroids shift at each iteration. Each data instance is assigned to the nearest centroid when the mean shift converges. Mean shift has a hyperparameter which is called *bandwidth*. The bandwidth defines the radius of the *region*, whose mean is the cluster centroid. Bandwidth is the only hyperparameter mean shift needs. Bandwidth affects the number of clusters [146, 147]. Selecting small values of bandwidth makes the algorithm focus on narrow regions and this causes more clusters to be predicted. Selecting large values of bandwidth causes the opposite.

Mean shift is a simple algorithm that has several advantages.

- It has a single hyperparameter, bandwidth, and there exist a few automatic bandwidth estimation methods in the literature.
- It does not make any model assumption on the data.
- It works well on clusters that have nonconvex shape.
- It has no issue of local minima.
- It is outlier tolerant.
- Its output does not depend on the initialization.

Affinity propagation [148] is an alternative algorithm that we could use instead of mean shift algorithm. However, affinity propagation has two critical hyperparameters, preference and damping and it is difficult to determine optimum values for these hyperparameters. DBSCAN [149, 150] and OPTICS [151] are other two alternative algorithms that are similar to each other and each needs a single hyperparameter. Nevertheless, to the best of our knowledge there exist no automated hyperparameter value selection tools for DBSCAN or OPTICS.

4.2.3.2 NoCStream

NoCStream is presented as an algorithm in Algorithm 11 and its flow is given in Figure 4.2. NoCStream embeds the input data into 2D by UMAP algorithm. Let

$$S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N\}$$

be the d-dimensional input data stream, where N goes to infinity and $\mathbf{x}_i$ is i^{th} data instance, which is a d-dimensional vector. Let

$$Y = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, ..., \mathbf{y}_N\}$$

be a two-dimensional representation of S, where $\mathbf{y}_i$ is a two-dimensional vector, as a result of UMAP. After generating Y using UMAP, NoCStream employs mean shift algorithm in order to cluster Y, several times with a different bandwidth value. Let

$$L^j = \{l_1^j, l_2^j, l_3^j, ..., l_N^j\}$$

be cluster labels of Y, assigned by mean shift using j^{th} bandwidth value (bw^j), where l_i^j is the cluster label of $\mathbf{y}_i$. k^j is calculated as the cardinality of set of L^j, that is the count of different labels in L^j.

NoCStream calculates the silhouette score of L^j (let it be SS^j) and decides on the most suitable k according to the best silhouette score. That is $k = k^j$ where SS^j (silhouette score of L^j) is the maximum one. Implementation of NoCStream is available online at `https://gitlab.com/alaettinzubaroglu/nocstream` with all other supplementary resources.

Algorithm 11 NoCStream (S)

```
Input: S : input data
Output: $k$ : number of clusters
  $BW$ = Set of Bandwidth Values
  $Y = UMAP(S)$
  $estimated\_bw = ESTIMATE\_BW(Y)$
  $BW = BW \cup estimated\_bw$
  $max\_ss = 0$
  for j = 1 to length of $BW$ do
      $bw^j = BW[j]$
      $L^j = MEAN\_SHIFT(Y, bw^j)$
      $SS^j = SILHOUETTE\_SCORE(L^j)$
      % get the count of different labels in $L^j$
      $k^j = |Set(L^j)|$
      if $SS^j > max\_ss$ then
          $max\_ss = SS^j$
          $k = k^j$
      end if
  end for
  return $k$
```

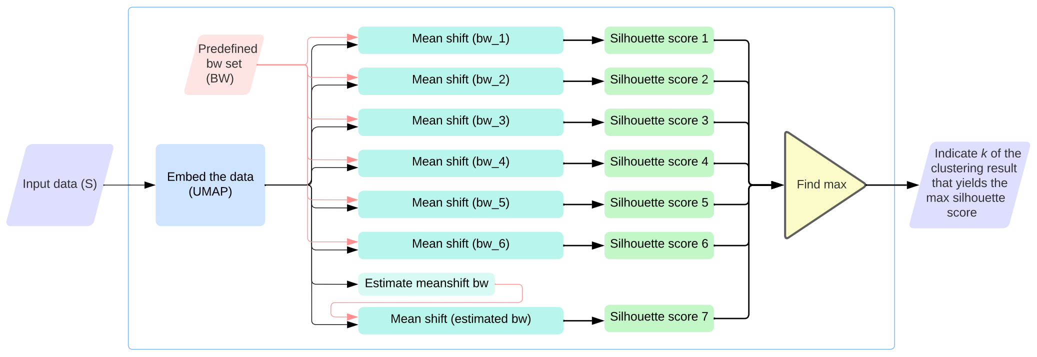


Figure 4.2: Main flow of NoCStream.

CHAPTER 5

EXPERIMENTAL RESULTS AND PERFORMANCE EVALUATION

5.1 Experimental Setup

5.1.1 Environment

All experiments are performed on a Lenovo T440p personal computer with "Intel Core i7-4700MQ CPU 2.40GHz x 4" processor and 16 GB memory. Installed operating system is Linux Mint 19.1 Cinnamon. Python version 3.8.10 is used for implementation of the proposed methods, EmCStream and NoCStream. DenStream and CluStream algorithms are used from streamMOA [132] package of R [133]. R version 3.4.4 is used.

5.1.2 Metrics

There exist several data clustering validation techniques in the literature [1]. Some of these traditional techniques, including Silhouette Score and Davies-Bouldin, have been recently adapted to data streams [152]. Cluster Mapping Measure (CMM) [153] is another data stream clustering validation metric. However, all the aforementioned metrics are for clustering algorithms that process streaming instances individually [20]. On the contrary, EmCStream processes the data stream window by window. Due to the algorithm's design, it is not possible to directly apply aforementioned metrics on the output of EmCStream. Instead, we apply traditional validation techniques, adjusted Rand index and purity, on each window and on the whole output, at the end of the processing.

5.1.2.1 Adjusted Rand Index

We have used adjusted Rand indexes (ARI) [154] as the clustering quality measure. Adjusted Rand index is the corrected-for-chance version of Rand index [155]. Rand index is calculated as a value between 0 and +1.0, while adjusted Rand index is calculated as a value between -1.0 and +1.0. It can yield a negative value when the Rand index is less than the expected index according to the distribution.

5.1.2.2 Purity

We have also compared average purity of clusters according to the clustering results of the algorithms. Purity measures the ratio of the instances that are labeled as in the same cluster and they are already in the same cluster according to the true labels [156]. In order to calculate purity, for each of the clusters, data instances from the most common cluster according to the true labels are counted and the sum, over all clusters is divided by the total number of data instances. Purity is calculated as a value between 0 and +1.0. Purity achieves the maximum value when each data instance is clustered as its own and this is obviously not a good clustering. Shortcoming of purity, it does not penalize having more clusters than actually exists.

5.2 Online Embedding and Clustering of Evolving Data Streams

We have evaluated EmCStream against two popular, baseline stream clustering algorithms, DenStream [12] and CluStream [13] based on the adjusted Rand index and purity, as the clustering quality metrics. EmCStream generates consistent and coherent cluster labels during the whole execution of a data stream. Thanks to this property, labels generated by EmCStream can be concatenated correctly to each other in subsequent windows. However, labels generated by DenStream and CluStream may change from one window to the next one for an evolving data stream, and it is not possible to concatenate these results. This situation makes it possible to calculate adjusted Rand index and purity for the whole execution of EmCStream, but not of DenStream or CluStream. Therefore, for a fair comparison, we calculate adjusted

Rand index and purity for each window of execution of three algorithms, and then provide the average of these values over all of the executed windows. Nevertheless, we also give a total adjusted Rand index and total purity on the concatenated labels.

5.2.1 Selection of Hyperparameter Values

Clustering quality of DenStream is highly dependent on the input parameter ϵ. ϵ defines the maximal radius of micro clusters created by DenStream during the execution and it is defined in the interval [0, 1]. We did grid search in the parameter search space with intervals of 0.01 for each data stream separately. Table 5.1 gives the best ϵ values for DenStream, and the *horizon* used during the tests, for synthetic and real world data streams. For the sake of fairness, we have used the same *horizon* for all three of the algorithms. We also have provided k to the algorithms, for all data streams.

Table 5.1: Algorithm parameters used during evaluation.

Stream	ϵ	***Horizon (h)***
Synthetic Streams	0.05	100
Meteo-TR	0.26	50
Meteo-EU	0.33	50
Meteo-US	0.03	50
Keystroke-2	0.18	20
Keystroke-3	0.57	50
Keystroke-4	0.40	80

As shown in Table 5.1, DenStream gives its best clustering results with the same ϵ for all synthetic data streams. This is expected, because all synthetic data streams are created using the same method and similar parameters (See Section 3.1). Characteristics of the clusters in these data streams are similar and this situation leads to equivalent ϵ values. As shown in Table 5.1, DenStream surprisingly gives its best clustering results on different ϵ values for each real world data stream. This is a little strange, because three meteorological data streams have the same features and equal number of clusters, which is two (See Section 3.2). The same result is also surprising for keystroke data streams. Even they have different number of clusters, Keystroke-2 is a subset of Keystroke-3 and they both are subsets of Keystroke-4. Because of the relationship between the data streams, we were expecting best ϵ values to be close to each other for the meteorological data streams and for the keystroke data streams as well. Current results show the sensitivity of DenStream to the ϵ parameter.

5.2.2 Evaluation on Synthetic Datasets

Synthetic and real world data streams used in this study are presented in Section 3.1 and Section 3.2, respectively. Table 5.2 gives adjusted Rand index values (average and total) of the algorithms on synthetic data streams. Average adjusted Rand index is calculated as average of adjusted Rand index of every single window, during the calculation. Total adjusted Rand index is calculated once, at the end of the process, over concatenated cluster labels. All elements of Table 5.2 are the averages of 10 repetitions. EmCStream and DenStream works in a deterministic manner on the synthetic data and therefore, they have 0 standard deviation. CluStream gives very close results for the repetitions and it has standard deviation values between 0.001 and 0.004.

According to total adjusted Rand indexes, it is clear that EmCStream gives coherent cluster labels on each window during the whole execution, while DenStream and CluStream do not, even on Stream-8, which is a stationary, non-evolving data stream. Moreover, EmCStream gives 1.0 adjusted Rand index and achieves perfect clustering results on noiseless synthetic data streams, while the two other algorithms cannot. Results of DenStream are also nearly perfect, however results of CluStream are obviously far from being a perfect clustering.

EmCStream successfully clusters evolving noisy data streams too. Stream-9, Stream-10 and Stream-11 are evolving noisy data streams and results of EmCStream are still better than DenStream and CluStream on these data streams. Clustering quality of EmCStream seems to decrease as the noise level in the data increases. This situation is already expected, because noise points do not belong to any cluster and even if all clean data instances are clustered successfully, noise points lead to a decrease in the adjusted Rand index value. Outputs of EmCStream are compatible with the noise

Table 5.2: Adjusted Rand index comparison on synthetic data streams.

	EmCStream		**DenStream**		**CluStream**	
Stream	**Average**	**Total**	**Average**	**Total**	**Average**	**Total**
Stream-1	1.000	1.000	0.998	0.032	0.880	0.002
Stream-2	1.000	1.000	0.998	0.050	0.878	0.002
Stream-3	1.000	1.000	0.987	0.033	0.862	0.002
Stream-4	1.000	1.000	0.996	0.018	0.892	0.002
Stream-5	1.000	1.000	0.998	0.091	0.955	0.002
Stream-6	1.000	1.000	0.996	0.021	0.821	0.002
Stream-7	1.000	1.000	0.998	0.075	0.900	0.002
Stream-8	1.000	1.000	0.998	0.246	0.868	0.002
Stream-9	0.934	0.933	0.931	0.026	0.836	0.002
Stream-10	0.851	0.846	0.848	0.042	0.803	0.002
Stream-11	0.649	0.631	0.646	0.031	0.670	0.001

levels of noisy data streams.

These results also show that, EmCStream and DenStream successfully adapt concept drift. According to the adjusted Rand index values, CluStream also adapts concept drift, but it is not as successful as EmCStream and DenStream. EmCStream has an extra feature, which is detecting the concept drift. A notification is generated when EmCStream detects a concept drift. On the other hand, DenStream and CluStream do not detect concept drift and they only adapt concept drift by outdating the stale data instances; that is why they do not create a notification about concept drift.

Table 5.3 gives purity values (average and total) of the algorithms on synthetic data streams. All elements of Table 5.3 are the averages of 10 repetitions. Purity values are very similar to adjusted Rand index values as expected and this supports our inferences.

Table 5.3: Purity comparison on synthetic data streams.

	EmCStream		**DenStream**		**CluStream**	
Stream	**Average**	**Total**	**Average**	**Total**	**Average**	**Total**
Stream-1	1.000	1.000	0.998	0.214	0.896	0.118
Stream-2	1.000	1.000	0.998	0.227	0.898	0.119
Stream-3	1.000	1.000	0.989	0.203	0.881	0.118
Stream-4	1.000	1.000	0.997	0.143	0.905	0.068
Stream-5	1.000	1.000	0.998	0.396	0.962	0.267
Stream-6	1.000	1.000	0.997	0.208	0.850	0.120
Stream-7	1.000	1.000	0.998	0.254	0.912	0.118
Stream-8	1.000	1.000	0.998	0.404	0.876	0.119
Stream-9	0.965	0.958	0.949	0.189	0.864	0.114
Stream-10	0.931	0.918	0.900	0.225	0.845	0.109
Stream-11	0.854	0.828	0.806	0.185	0.770	0.108

5.2.3 Evaluation on Real World Datasets

Table 5.4 gives adjusted Rand index values (average and total) of the algorithms on real world data streams. All elements of Table 5.4 are the averages of 10 repetitions. All algorithms produce very close results on each repetition and the maximum value of the standard deviation is 0.018, for CluStream, on Keystroke-4 data stream.

Again it is obvious that EmCStream generates cluster labels that can be concatenated, during the whole execution while other algorithms do not. EmCStream achieves perfect clustering on meteorology data streams and the other algorithms also produce nearly perfect results, according to average adjusted Rand index. On Keystroke data streams, clustering quality of EmCStream decreases a little bit, while clustering quality of other algorithms drops off. Moreover, DenStream again gives better results than CluStream.

Table 5.5 gives purity values (average and total) of the algorithms on synthetic data streams. All elements of Table 5.5 are the averages of 10 repetitions. Purity values

Table 5.4: Adjusted Rand index comparison on real world data streams.

	EmCStream		**DenStream**		**CluStream**	
Stream	**Average**	**Total**	**Average**	**Total**	**Average**	**Total**
Meteo-TR	1.000	1.000	0.998	0.995	0.995	0.000
Meteo-EU	1.000	1.000	0.997	0.662	0.999	0.001
Meteo-US	1.000	1.000	0.998	0.997	0.999	0.000
Keystroke-2	1.000	1.000	0.775	0.600	0.410	0.005
Keystroke-3	0.946	0.956	0.537	0.438	0.328	0.012
Keystroke-4	0.894	0.894	0.540	0.485	0.355	0.025

Table 5.5: Purity comparison on real world data streams.

	EmCStream		**DenStream**		**CluStream**	
Stream	**Average**	**Total**	**Average**	**Total**	**Average**	**Total**
Meteo-TR	1.000	1.000	0.998	0.998	0.997	0.504
Meteo-EU	1.000	1.000	0.998	0.906	0.999	0.507
Meteo-US	1.000	1.000	0.999	0.998	0.999	0.514
Keystroke-2	1.000	1.000	0.887	0.825	0.745	0.496
Keystroke-3	0.982	0.985	0.730	0.682	0.565	0.355
Keystroke-4	0.959	0.959	0.761	0.741	0.517	0.289

are similar to adjusted Rand index values as expected and they support our inferences.

5.2.4 Embedded Dimensions

In the current version of EmCStream, the data are embedded into 2D and clustering is then applied in 2D embedded space. When the data are embedded into a lower dimensional space, there is a risk that some of the characteristics of data may have been lost and therefore, clustering becomes a more challenging task in the lower dimensional embedded space. Hence, we have evaluated and compared the clustering performance of EmCStream when data stream is embedded in 3D and in 2D. Table 5.6 shows adjusted Rand index results of EmCStream using 2D and 3D embedding spaces, on both synthetic and real world data streams. EmCStream using 2D and 3D embedding spaces both have the same performance on the synthetic streams and on some of the real world data streams. On Keystroke-3 and Keystroke-4 data streams, 3D embedding space gives slightly better results than 2D embedding space. Since the difference is very small, we have decided to use 2D embedding space, in order to make the visualization easier. Purity results are also analyzed and similar discussions are valid for purity results as well. We did not include here the purity results for the sake of simplicity. The embedded space dimension is a parameter in the source code, and its value can be modified by the user.

5.2.5 Detected Drift Count

Since the main objective of EmCStream is to cluster evolving high dimensional data streams successfully, a concept drift is detected only if it destroys the clustering quality. Table 5.7 shows detected drift count by EmCStream, on both synthetic and real world data streams. EmCStream detects two drifts in the baseline Stream-1. In Stream-3, which has lower dimensionality compared to Stream-1, eight drifts are detected. A lower dimensional space is less discriminating and the reason why a same-speed drift destroys the clustering quality in Stream-3 more than in Stream-1 can be explained by this. Stream-4 has two times more clusters than Stream-1 and this causes clusters to violate other clusters' spaces in a less drift in the data. Hence,

Table 5.6: Performance comparison (in terms of total adjusted Rand index) of EmCStream when 2D embedding space and 3D embedding space are used.

Stream	**2D**	**3D**	**Stream**	**2D**	**3D**
Stream-1	1.000	1.000	Meteo-TR	1.000	1.000
Stream-2	1.000	1.000	Meteo-EU	1.000	1.000
Stream-3	1.000	1.000	Meteo-US	1.000	1.000
Stream-4	1.000	1.000	Keystroke-2	1.000	1.000
Stream-5	1.000	1.000	Keystroke-3	0.956	0.961
Stream-6	1.000	1.000	Keystroke-4	0.894	0.903
Stream-7	1.000	1.000			
Stream-8	1.000	1.000			
Stream-9	0.933	0.933			
Stream-10	0.846	0.846			
Stream-11	0.631	0.631			

EmCStream eleven times notifies a drift in Stream-4. Stream-5 has less clusters than Stream-1, but their other characteristics are same and EmCStream notifies equal number of drifts on them. Stream-6 has a four times faster drift than Stream-1 and four times more drifts are detected. Stream-7 has a low speed drift and this does not destroys the clustering quality in 50,000 data instances. EmCStream successfully clusters Stream-7 by notifying no concept drift. Last noiseless stream, Stream-8 is a stationary, non-evolving data stream and EmCStream notifies no concept drift. Stream-9, Stream-10 and Stream-11 are noisy streams and they have a same speed drift with Stream-1. However EmCStream notifies huge number of drifts on these streams. The reason of this situation is most probably the noise points in these streams that might have destroyed the clustering quality and this situation could have been interpreted as concept drift by EmCStream. Since the noise points appear randomly and do not follow a pattern, it is not incorrect to interpret them as concept drift. Moreover, EmCStream notifies more concept drifts when the noise level is higher and this supports our claim. According to these results, EmCStream successfully detects concept drift that destroys the clustering quality and detected drifts are coherent with the stream characteristics.

Meteorology data that is used in this study are measurements of five years and detected drifts are 12, 17 and 16. We believe these are reasonable drift counts for seasonal changes in the climate. Lastly, Keystroke-3 data stream has more drifts than Keystroke-2, which should be caused by the third cluster added to the stream. Moreover, fourth added cluster seems to have no effects on concept drift because Keystroke-3 and Keystroke-4 have the same number of drifts.

Table 5.7: Number of detected drifts by EmCStream algorithm.

Stream	**Drifts**	**Stream**	**Drifts**
Stream-1	2	Meteo-TR	12
Stream-2	2	Meteo-EU	17
Stream-3	8	Meteo-US	16
Stream-4	11	Keystroke-2	2
Stream-5	2	Keystroke-3	6
Stream-6	8	Keystroke-4	6
Stream-7	0		
Stream-8	0		
Stream-9	91		
Stream-10	107		
Stream-11	132		

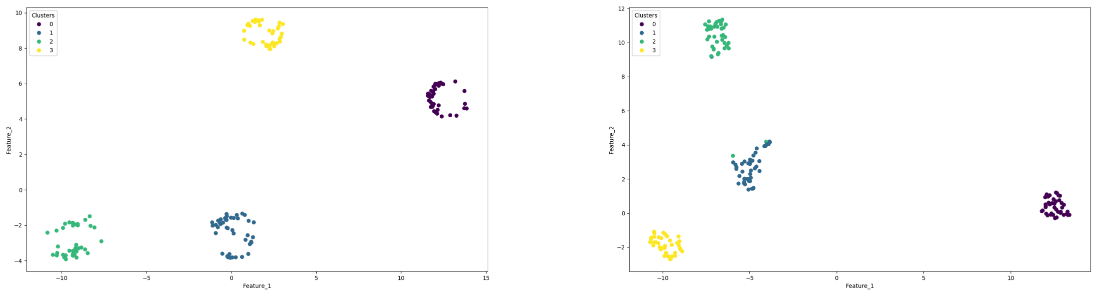
5.2.6 Execution Time

One other characteristic of EmCStream is that, its execution time is not affected by the dimension of the data. We are not comparing execution time of these algorithms, because EmCStream is not implemented in the same environment with DenStream and CluStream. Hence, it is not fair to compare execution time of EmCStream against other algorithms. However, we are analyzing how execution time of each algorithm is affected by the properties of input data. Table 5.8 shows execution times of the algorithms on the synthetic data streams. First 3 streams have different dimensions and execution time of EmCStream on these streams are close to each other. However execution times of DenStream and CluStream on the same streams differ a lot. The reason of this difference is their dependency on the data dimensionality.

Table 5.8: Execution times (in seconds) of the algorithms on synthetic data streams.

Stream	k	d	Noise (%)	Detected Drifts	Drift Speed	EmCStream	DenStream	CluStream
Stream-1	10	50	0	2	normal	210	160	40
Stream-2	10	100	0	2	normal	206	244	57
Stream-3	10	10	0	8	normal	224	89	33
Stream-4	20	50	0	11	normal	292	120	47
Stream-5	4	50	0	2	normal	213	241	44
Stream-6	10	50	0	8	high	222	159	44
Stream-7	10	50	0	0	low	225	140	43
Stream-8	10	50	0	0	no	181	180	43
Stream-9	10	50	5	91	normal	257	149	42
Stream-10	10	50	10	107	normal	347	135	41
Stream-11	10	50	20	132	normal	539	115	41

Even though execution time of EmCStream is not affected by the data dimension, it is directly affected by the detected drift count. Drift detection and adaptation is a time consuming task for EmCStream and this increases the execution time on fast evolving, or noisy data streams, as seen in Table 5.8 on Stream-9, Stream-10 and Stream-11. Execution time of EmCStream gets longer when the data have more noise points, while execution times of DenStream and CluStream are not affected negatively. Moreover, it is obvious that CluStream is a faster algorithm than DenStream, however clustering quality of DenStream is better than CluStream. We are not commenting on execution times of Streams 4-8 at this time. They also can be examined to see how other factors, such as k and drift speed, affect the execution time.

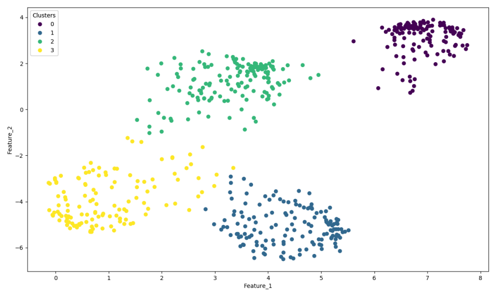


(a) (b)

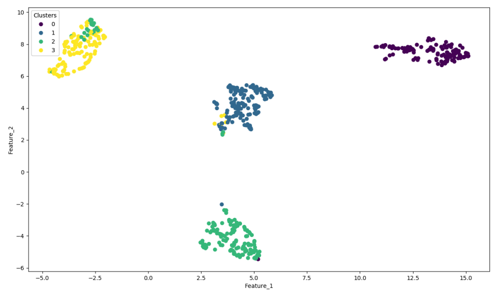
Figure 5.1: Visualization of how EmCStream checks for a concept drift, on Keystroke-4 dataset. The same data instances are embedded twice. In (a) the data are embedded according to previous knowledge, acquired in the initialization, in (b) the data are embedded using no previous knowledge. Both figures are colored according to clustering results of (a), in order to show the coherency between the two embedding. This situation is accepted as no drift because clustering of (a) and (b) are coherent, as shown by coloring.

5.2.7 Visualization and Drift Check

Figure 5.1 and Figure 5.2 show visualization of the embedded data during a concept drift check. In (a), the data are embedded according to the knowledge acquired during the initialization, in (b), the same data are embedded with no previous knowledge. Both (a) and (b) are colored according to clustering results of (a), in order to show the coherency between the two embedding. In Figure 5.1, coloring of (b) according to clustering results of (a) seems to be compatible and this situation is accepted as no drift. However, in Figure 5.2 (b), that is colored according to clustering results of (a), there exist several blue and yellow instances in the green cluster and several yellow instances in the blue cluster. This instances are labeled as in different clusters in (a) and (b) and this situation is accepted as a concept drift.



(a)



(b)

Figure 5.2: Visualization of how EmCStream checks for a concept drift, on Keystroke-4 dataset. The same data instances are embedded twice. In (a) the data are embedded according to previous knowledge, acquired in the initialization, in (b) the data are embedded using no previous knowledge. Both figures are colored according to clustering results of (a), to show the coherency between the two embedding. This situation is accepted as a drift because clustering of (a) and (b) are not coherent for a number of instances, as shown by coloring.

5.2.8 Discussion

We have presented a novel method EmCStream that embeds and clusters in real time evolving data streams. EmCStream continuously embeds high dimensional input data into two dimensions and clusters the embedded data using k-means algorithm. UMAP is employed for the embedding process. EmCStream is capable of detecting and adapting concept drift, while most of other stream clustering algorithms do not detect and notify concept drift. EmCStream also makes possible the visualization of high dimensional data, by means of embedding. Moreover, EmCStream generates consistent and coherent cluster labels that can be concatenated, during the whole execution. We have compared EmCStream against two most popular state of the art data stream clustering algorithms, DenStream and CluStream, on both synthetic and real world data streams. Our method outperforms DenStream and CluStream in terms of clustering quality, on both synthetic and real world data streams. We have used adjusted Rand index and purity as clustering quality metrics.

As a shortcoming, clustering quality of EmCStream depends on embedding quality of UMAP. Moreover, it is possible to further evaluate EmCStream using data streams evolving with different characteristics.

5.3 Determining the Optimal Number of Clusters on High Dimensional Evolving Data Streams

5.3.1 Evaluation Methodology

We have evaluated the proposed method in three different ways as follows.

1. ***k* prediction** DenStream and NoCStream are employed as k prediction algorithms. The prediction is performed on the whole data stream window by window. Two algorithms are compared with respect to the number of windows on which k is predicted correctly.

2. **clustering quality** EmCStream with NoCStream and DenStream are evaluated in terms of clustering quality when the number of clusters k is not specified. We employ NoCStream as a k prediction method for EmCStream [10], which is a data stream clustering algorithm that does not predict k, and we have evaluated them against DenStream, in terms of clustering quality. In Section 4.1, the clustering quality of EmCStream is evaluated against DenStream and CluStream [13] on evolving data streams with a constant number of clusters, and when the true k is specified by the user. Here, k is not specified, and therefore CluStream is not included in the evaluation since it cannot predict k.

3. **genericity** In order to test the genericity of NoCStream and the quality of its accordance with different clustering algorithms as a k prediction method, we have integrated NoCStream into the spectral clustering algorithm [157] and we have evaluated the clustering quality of this incorporation against the case where true k is given to the spectral clustering algorithm.

We have performed the evaluation on both synthetic and real world evolving data streams with different characteristics. In addition to the data streams used in Section 4.1, eight new evolving data streams with changing k are included in this study. Five of these new data streams are synthetic and three of them are real, which are the Keystroke-C, Sensor-2, and Sensor-3 data streams. Details of data streams are presented in Chapter 3.

We have used adjusted Rand index (See Section 5.1.2.1) and purity (See Section 5.1.2.2) as the clustering quality metrics. Both of the metrics have shown similar results and we did not include here the purity results for the sake of simplicity. Python version 3.8.10 is used for the implementation of the proposed method, NoCStream. DenStream algorithm is used from streamMOA [132] package of R [133]. R version 3.6.3 is used. The spectral clustering algorithm is used from Scikit-learn library [158, 159] of Python.

5.3.2 Selection of Hyperparameter Values

1. ϵ **for DenStream** The clustering quality of DenStream is highly dependent on the value of hyperparameter ϵ. ϵ defines the maximal radius of micro clusters created by DenStream during the execution and its value is defined in the interval [0, 1]. Table 5.9 shows the best ϵ values for DenStream, for synthetic and real world data streams. The explanation of finding these best ϵ values is given in Section 5.2.1 [10].

2. **horizon ($h_{clustering}$) for EmcStream** Selecting the value for $h_{clustering}$ requires domain knowledge. It is possible to specify the horizon as either number of instances or a time duration. For instance, for a stream in which 100 data instances are generated per second, it is possible to specify the horizon as, for example "$h =$ *1-minute data*", or "$h = 60 \times 100 = 6000$ *data instances*". Ten is an adequate number of instances in order to form a cluster. In synthetic data streams, it is assumed that each data source generates one data instance per second. This means in a data batch of ten seconds, there will exist about ten samples from each data source on average. For synthetic data streams, $h_{clustering}$ is specified as a data batch of ten seconds. Each data source generates the samples of a single cluster. In the same way, $h_{clustering}$ for Keystroke datasets is also specified as a data batch of ten unit times. Here unit time means the time

duration that is needed for each data source to generate a single instance. In the Sensor datasets, the instances of different clusters are not well distributed, hence we have set $h_{clustering}$ to a data batch of 20 unit times. The meteorological datasets consist of hourly measurements of two cities with different climate characteristics. For these data, we aimed to catch seasonal concept drifts, instead of daily (day-night) concept drifts. Hence we have set $h_{clustering}$ to a data batch of one day, which composes of 48 data instances.

3. **horizon ($h_{k-prediction}$) for NoCStream** We have integrated NoCStream as a k prediction method for the initialization phase of EmCStream. EmCStream uses $2 \times h$ data instances for initialization [10]. In this way, we have defined $h_{k-prediction} = 2 \times h_{clustering}$.

Table 5.9 presents the values for hyperparameter horizon (h) for clustering and k prediction.

NoCStream employs mean shift algorithm with seven different bandwidth values; six of them are fixed and one is estimated for each run according to the input data window being processed. The empirically formed set of bandwidth values is as follows, BW = {1.5, 1.75, 2, 3, 4, 8, estimated_bw}. Here estimated bandwidth is calculated using *estimate_bandwidth* function of Scikit-learn library [158, 159]. This function estimates a bandwidth value according to the mean of pairwise distances between the instances. Aforementioned set of bandwidth values works well for all synthetic and real world data streams used in this study. It is also possible to define a different bandwidth set for different data and needs. Several different bandwidth selection techniques are mentioned in [146]. Moreover, a study on the comparison of automated bandwidth selection methods is presented in [147].

5.3.3 Evaluation Results

5.3.3.1 *k* Prediction Evaluation

For evaluation purposes, NoCStream is applied to the whole data streams window by window. A number of clusters is predicted on each window by NoCStream and by

Table 5.9: Evaluation Parameters.

Stream	ϵ	$h_{clustering}$	$h_{k-prediction}$
Synthetic Streams	0.05	10 seconds batch	20 seconds batch
Meteo-TR	0.26	1 day batch	2 days batch
Meteo-EU	0.33	1 day batch	2 days batch
Meteo-US	0.03	1 day batch	2 days batch
Keystroke-2	0.18	10 unit time batch	20 unit time batch
Keystroke-3	0.57	10 unit time batch	20 unit time batch
Keystroke-4	0.40	10 unit time batch	20 unit time batch
Keystroke-C	0.18	10 unit time batch	20 unit time batch
Sensor-2	0.05	20 unit time batch	40 unit time batch
Sensor-3	0.05	20 unit time batch	40 unit time batch

DenStream separately. DenStream is run with the ϵ value that gives the best result on each input stream, according to Table 5.9. The number of windows on which k is predicted correctly are compared. Stream properties and comparison results are presented in Table 5.10 for synthetic data streams and in Table 5.11 for real world data streams. k indicates *number of clusters*, d indicates *dimensions* and h indicates *horizon*.

NoCStream and DenStream give perfect or nearly perfect results on noiseless data streams with constant k. However on noisy data streams, NoCStream significantly outperforms DenStream in terms of the number of windows, and k is predicted successfully. On data streams with changing k, which are also noiseless, NoCStream slightly outperforms DenStream. Details of the evaluation are given below.

- Synthetic Streams 1 to 8 do not include any noise and NoCStream predicts k correctly on all windows of these streams. DenStream is incorrect only on two windows of Stream-4 and on a single window of Stream-5, which may be neglected.
- Streams 9 to 11 have noise with different ratios. NoCStream again predicts k correctly on nearly all windows of these streams, while DenStream cannot. The

noisier the data, the less successful result DenStream gives. On Stream-11, k is predicted correctly only on 82 of 250 windows, by DenStream.

- Streams 12 to 16 are data streams with changing k, which means some clusters disappear and emerge. NoCStream predicts k perfectly on four of them, and nearly perfectly on Stream-15. DenStream predicts k perfectly on two of them, nearly perfectly on another two streams, and with a 91% success on Stream-14.

Table 5.10: Number of windows in which k is predicted successfully, on synthetic data streams.

Stream	k	d	**Noise (%)**	h	**Drift Speed**	**NoCStream**	**DenStream**	**Total Windows**
Stream-1	10	50	0	200	normal	250	250	250
Stream-2	10	100	0	200	normal	250	250	250
Stream-3	10	10	0	200	normal	250	250	250
Stream-4	20	50	0	400	normal	125	123	125
Stream-5	4	50	0	80	normal	625	624	625
Stream-6	10	50	0	200	high	250	250	250
Stream-7	10	50	0	200	low	250	250	250
Stream-8	10	50	0	200	no	250	250	250
Stream-9	10	50	5	200	normal	247	134	250
Stream-10	10	50	10	200	normal	249	101	250
Stream-11	10	50	20	200	normal	247	82	250
Stream-12	10 ± 2	10	0	200	normal	250	250	250
Stream-13	10 ± 2	20	0	200	normal	250	250	250
Stream-14	10 ± 2	5	0	200	normal	250	228	250
Stream-15	20 ± 4	10	0	400	normal	121	118	125
Stream-16	4 ± 1	10	0	80	normal	625	623	625

NoCStream falls behind DenStream on meteorological real world data streams. However, NoCStream gives more balanced results on three of the meteorological data streams while DenStream does not. NoCStream significantly outperforms DenStream on Keystroke datasets. Neither of the algorithms show remarkable success on Sensor datasets, however, predictions of EmCStream again are better than those of Den-

Stream. NoCStream outperforms DenStream on both Keystroke and Sensor datasets. The details of the evaluation are indicated below.

- On meteorological data of Turkey, Europe and US, success ratio of NoCStream is 66%, 69% and 72% respectively. These are 89%, 100% and 3% for DenStream, on the same datasets. On both Turkey and Europe data, DenStream outperforms NoCStream, while on US data, DenStream totally fails and NoCStream outperforms it, in spite of the success ratio of NoCStream is about 72%.
- On the Keystroke-2 dataset, NoCStream predicts k perfectly while DenStream can predict only nearly half of the whole windows. On Keystroke-3 and Keystroke-4 datasets, NoCStream shows a success of 95% and 80%, while DenStream cannot do even a single successful prediction. On the Keystroke-C dataset, the success ratio of NoCStream is 91% while that of DenStream is 33%.
- Success ratio of NoCStream is 25% and 37% on Sensor-2 and Sensor-3 datasets, respectively. These are 18% and 12% for DenStream. Neither of the algorithms show remarkable success on Sensor datasets.

5.3.3.2 Clustering Evaluation

NoCStream can be integrated into clustering algorithms as a k prediction method. For evaluation, we have integrated NoCStream into the initialization phase of EmCStream and evaluated them against DenStream, in terms of clustering quality. Adjusted Rand index (See Section 5.1.2.1) and purity (See Section 5.1.2.2) are used as the clustering quality metrics. EmCStream generates consistent and coherent cluster labels during the whole execution of a data stream, as long as k does not change. If k changes during the execution, EmCStream resets the labels and creates a notification about this situation. Thanks to this property, labels generated by EmCStream can be concatenated correctly to each other in subsequent windows, until k changes. However, labels generated by DenStream may change from one window to the next one for an evolving data stream, and it is not possible to concatenate these results. This situation makes it possible to calculate the adjusted Rand index and purity for the whole

Table 5.11: Number of windows in which k is predicted successfully, on real world data streams.

Stream	k	d	h	**NoCStream**	**DenStream**	**Total Windows**
Meteo-TR	2	6	96	602	809	913
Meteo-EU	2	6	96	633	912	913
Meteo-US	2	6	96	657	26	913
Keystroke-2	2	31	40	20	9	20
Keystroke-3	3	31	60	19	0	20
Keystroke-4	4	31	80	16	0	20
Keystroke-C	2-4	31	80	41	15	45
Sensor-2	2	4	80	198	152	802
Sensor-3	2-3	4	80	505	167	1350

execution of EmCStream, but not of DenStream. Therefore, for a fair comparison, we calculate the adjusted Rand index and purity for each window of execution of the algorithms and then provide the average of these values over all of the executed windows. Nevertheless, we also give a total adjusted Rand index on the concatenated labels. Both adjusted Rand index and purity have shown similar results and we did not include here the purity results for the sake of simplicity.

Table 5.12 presents adjusted Rand index values of clustering results on the synthetic data streams. Both EmCStream and DenStream show similar and successful clustering performance on synthetic data streams, which shows that NoCStream predicts the number of clusters successfully in different and changing stream characteristics. Average adjusted Rand index values are very similar for both of the algorithms and they are perfect or nearly perfect on all of the streams. However this is not the case for total adjusted Rand index values. On the data streams with constant k, which are Streams 1 to 11, total adjusted Rand index values are very similar to the average values for EmCStream but not for DenStream. Because k changes on Streams 12 to 16, total adjusted Rand index values are low for both of the algorithms. Still, total adjusted Rand index values of EmCStream are greater than DenStream. This situation is already expected because EmCStream resets the cluster labels only when k changes, while DenStream may reset them on every new window. NoCStream achieves this performance without any hyperparameter selection or optimization while DenStream needs the optimum ϵ value as a hyperparameter.

Table 5.12: Adjusted Rand index comparison on synthetic data streams.

		Noise		**Average ARI**		**Total ARI**	
Stream	k	**(%)**	h	**EmCStream**	**DenStream**	**EmCStream**	**DenStream**
Stream-1	10	0	100	1.000	0.998	1.000	0.015
Stream-2	10	0	100	1.000	0.998	1.000	0.018
Stream-3	10	0	100	1.000	0.998	1.000	0.016
Stream-4	20	0	200	1.000	0.998	1.000	0.012
Stream-5	4	0	40	1.000	0.998	1.000	0.020
Stream-6	10	0	100	1.000	0.998	1.000	0.008
Stream-7	10	0	100	1.000	0.998	1.000	0.047
Stream-8	10	0	100	1.000	0.998	1.000	0.246
Stream-9	10	5	100	0.934	0.938	0.932	0.013
Stream-10	10	10	100	0.848	0.857	0.846	0.012
Stream-11	10	20	100	0.647	0.659	0.631	0.009
Stream-12	10 ± 2	0	100	0.997	0.997	0.098	0.019
Stream-13	10 ± 2	0	100	0.998	0.996	0.107	0.016
Stream-14	10 ± 2	0	100	0.997	0.989	0.108	0.023
Stream-15	20 ± 4	0	200	0.999	0.998	0.099	0.013
Stream-16	4 ± 1	0	40	0.996	0.998	0.131	0.045

Clustering results of real world data streams are presented in Table 5.13. EmCStream outperforms DenStream on all real world datasets. EmCStream successfully clusters all datasets, while DenStream cannot cluster Keystroke and sensor datasets. EmCStream outperforms DenStream on real world data streams in terms of clustering quality, with the aid of NoCStream as a k prediction method.

- EmCStream gives nearly perfect clustering results on three of the meteorological data streams. DenStream also gives nearly perfect clustering results on both the meteorological data of Turkey and Europe but not on the meteorological data of US.

- EmCStream clusters successfully the Keystroke datasets as well. It gives 1.0 adjusted Rand index for Keystroke-2 and nearly 0.9 for Keystroke-3, Keystroke-4, and Keystroke-C datasets. However, DenStream cannot give successful results on Keystroke datasets.
- Also on Sensor datasets, results of EmCStream are much more better than those of DenStream.

Table 5.13: Adjusted Rand index comparison on real world data streams.

			Average ARI		**Total ARI**	
Stream	k	h	**EmCStream**	**DenStream**	**EmCStream**	**DenStream**
Meteo-TR	2	48	1.000	0.992	1.000	0.955
Meteo-EU	2	48	0.997	0.998	0.992	0.657
Meteo-US	2	48	1.000	0.673	1.000	0.437
Keystroke-2	2	20	1.000	0.733	1.000	0.431
Keystroke-3	3	30	0.886	0.122	0.355	0.076
Keystroke-4	4	40	0.899	0.000	0.873	0.000
Keystroke-C	2-4	40	0.891	0.657	0.320	0.523
Sensor-2	2	40	0.761	0.143	-0.001	-0.001
Sensor-3	2-3	40	0.821	0.246	0.005	0.003

5.3.3.3 Genericity Evaluation

In order to test the genericity of NoCStream and the quality of its accordance with different clustering algorithms as a k prediction method, we have integrated NoCStream to the spectral clustering algorithm [157]. Spectral clustering is a traditional clustering technique that first reduces the dimensionality of the data using eigenvalues of the similarity matrix, then clusters the data in reduced dimensions. In that technique, k is not predicted by the algorithm and it should be stated by the user. It is not specialized for data streams, however, we have applied NoCStream-aided spectral clustering on the streams window by window. We have evaluated this cooperation against spectral clustering which is given the true k, in terms of clustering quality. We have compared the average adjusted Rand index values of all windows of execution, for each data stream. For the tests with given true k, we have provided the initial true k to the spectral clustering algorithm. For most of the data streams, k is already constant. However, in Streams 12 to 16, k changes over time. Details of data streams are explained in Chapter 3.

Table 5.14 presents evaluation results on the synthetic data streams. On the datasets that have constant k (with or without noise), NoCStream-aided spectral clustering shows similar successful results with the version that true k is provided. On the noisy datasets, which are Streams 9 to 11, both of the implementations give nearly perfect clustering results and these are in coherence with the noise ratio. On the datasets that k changes, which are Streams 12 to 16, NoCStream-aided version gives more successful clustering results as expected. NoCStream successfully predicts k on each window, although data evolve and k changes.

Table 5.15 presents evaluation results on the real world data streams. On meteorological data streams, perfect clustering is performed when true k is provided. For NoCStream-aided implementation, the results are only a little bit worse than perfect. On the other hand, NoCStream-aided implementation gives slightly better results on Keystroke data streams. Both meteorological and Keystroke data streams are evolving, real world data streams in which k remains constant, except Keystroke-C, in which k changes from two to three and then to four. Unsurprisingly, NoCStream-aided implementation gives much more better results than its competitor on the Keystroke-C data stream. However on Sensor datasets, the result of true k given implementation outperforms the NoCStream-aided version, even though both results are not very different.

By the tests explained in this section, we show that NoCStream is a generic and successful k prediction method that can be integrated to various clustering algorithms.

Table 5.14: Spectral clustering adjusted Rand index comparison on synthetic data streams.

					Average ARI	
					true k	**NoCStream**
Stream	k	**given k**	**Noise (%)**	h	**given**	**aided**
Stream-1	10	10	0	100	0.999	0.996
Stream-2	10	10	0	100	1.000	0.996
Stream-3	10	10	0	100	1.000	0.997
Stream-4	20	20	0	200	1.000	0.996
Stream-5	4	4	0	40	1.000	0.997
Stream-6	10	10	0	100	1.000	0.996
Stream-7	10	10	0	100	1.000	0.997
Stream-8	10	10	0	100	1.000	0.996
Stream-9	10	10	5	100	0.934	0.930
Stream-10	10	10	10	100	0.851	0.851
Stream-11	10	10	20	100	0.662	0.685
Stream-12	10 ± 2	10	0	100	0.929	0.993
Stream-13	10 ± 2	10	0	100	0.941	0.994
Stream-14	10 ± 2	10	0	100	0.943	0.986
Stream-15	20 ± 4	20	0	200	0.932	0.992
Stream-16	4 ± 1	4	0	40	0.922	0.994

Table 5.15: Spectral clustering adjusted Rand index comparison on real world data streams.

				Average ARI	
				true k	**NoCStream**
Stream	k	**given** k	h	**given**	**aided**
Meteo-TR	2	2	48	1.000	0.935
Meteo-EU	2	2	48	1.000	0.950
Meteo-US	2	2	48	1.000	0.965
Keystroke-2	2	2	20	0.935	0.967
Keystroke-3	3	3	30	0.847	0.824
Keystroke-4	4	4	40	0.718	0.734
Keystroke-C	2-4	2	40	0.523	0.800
Sensor-2	2	2	40	0.976	0.892
Sensor-3	2-3	3	40	0.939	0.934

5.3.4 Discussion

Within data stream processing, online k prediction is a challenging task, especially for evolving data streams with changing k. To this end, we have presented NoC-Stream that continuously predicts k in real-time, on high dimensional, evolving data streams. NoCStream is also capable of finding the changing k. It is specialized to be used on high dimensional, evolving data streams that have some clusters emerge and disappear.

- For evaluation, which is against DenStream, we have counted how many times NoCStream predicts a number of clusters correctly on all instances of a data stream.
- We have also employed NoCStream as the k prediction method of EmCStream and evaluated the clustering performance of NoCStream-aided EmCStream.
- Furthermore, we have integrated NoCStream to the spectral clustering algo-

rithm and have evaluated it against the same algorithm with given true k.

With no hyperparameter selection and optimization, NoCStream successfully predicts a number of clusters on both real world and synthetic data streams with different characteristics. As an advantage, NoCStream does not need candidate k values as it is required in the silhouette method, elbow method, or their variations. Moreover, NoCStream is successfully integrated into EmCStream and spectral clustering algorithms as a k prediction method and they work together in success. Adjusted Rand index and purity are used as the clustering quality metrics.

CHAPTER 6

CONCLUSIONS

Online data stream processing is a popular research area that is arousing interest. Within this research area, we have presented two novel methods, Online Embedding and Clustering of Evolving Data Streams (EmCStream) and Determining the Optimal Number of Clusters on High Dimensional Evolving Data Streams (NoCStream).

EmCStream continuously embeds high dimensional input data into two dimensions and clusters the embedded data. EmCStream is capable of detecting and adapting concept drift, while most of other stream clustering algorithms do not detect and notify concept drift. In general, other algorithms only adapt concept drift by outdating the stale data instances. EmCStream also makes possible the visualization of high dimensional data, by means of embedding. Moreover, EmCStream generates consistent and coherent cluster labels during the whole execution. In other algorithms, clustering results on each update are independent from each other, and they cannot be concatenated. Labels generated by EmCStream can be concatenated. EmCStream remedies this shortcoming of other state of the art stream clustering algorithms and offers an alternative method for online clustering of evolving data streams. We have compared EmCStream against two most popular state of the art data stream clustering algorithms, DenStream and CluStream, on both synthetic and real world data streams. Our method outperforms DenStream and CluStream in terms of clustering quality, on both synthetic and real world data streams.

NoCStream continuously predicts k in real-time, on high dimensional, evolving data streams. NoCStream is also capable of finding the changing k. It is specialized to be used on high dimensional, evolving data streams that have some clusters emerge and disappear. Moreover, NoCStream does not need candidate k values as it is required in the silhouette method, elbow method, or their variations. For evaluation, which is against DenStream, we have counted how many times NoCStream predicts a number of clusters correctly on all instances of a data stream. With no hyperparameter selection and optimization, NoCStream successfully predicts a number of clusters on both real and synthetic data streams with different characteristics. We have also employed NoCStream as the k prediction method of EmCStream and evaluated the clustering performance of NoCStream-aided EmCStream. NoCStream is successfully integrated into EmCStream and they outperform DenStream in terms of clustering quality when k is not given to the algorithms. Furthermore, we have integrated NoCStream to the spectral clustering algorithm and have evaluated it against the same algorithm with given true k. NoCStream is successfully integrated into spectral clustering algorithm as a k prediction method and they work together in success.

The methods proposed in this thesis are set to use the euclidean distance. Both of the methods first process the data using UMAP and UMAP supports several distance types. Therefore, EmCStream and NoCStream can be easily converted to use a different distance type supported by UMAP or to get the distance type as a hyperparameter. Thanks to this feature, the proposed methods can be applied on different types of data, using different distance types. EmCStream adaptively adjusts drift check period during the execution. This makes EmCStream successfully handle the change in the speed of the concept drift throughout a data stream. Speed of the concept drift is constant in the synthetic datasets used in this study. However, the real world datasets are also evolving and the speed of the concept drift changes by nature. Thus, the change in the speed of the concept drift is already tested by the real world datasets presented. For the proposed methods the horizon is a hyperparameter and it is constant throughout the processing. If there is a need, changing the horizon during the processing can be studied. Moreover, it is possible to run multiple copies of EmCStream simultaneously on the same data, with different horizons. Determining the horizon requires some domain knowledge and depends on what the user prefers to find. For example if the user wants to catch the daily concept drift on a meteorological data, horizon can be set to the data of one hour duration. However, when the user wants to catch the seasonal or day to day concept drift, horizon can be set to the data of a couple of days. Knowing the data generation period of the data sources also helps in determining the horizon. By way of illustration, in our tests on the synthetic datasets we set the horizon to ten unit times, where unit time is the average data generation period of each data source. We consider ten is an adequate number of instances in order to form a cluster and when such a horizon is used, each cluster has ten data instances on average.

There is a lack of high quality benchmark data to use in data stream clustering algorithms. It is very common to use traditional datasets for testing and benchmark purposes. However, traditional datasets do not include concept drift. Evaluation of data stream clustering algorithms on traditional datasets does not produce reliable evaluation results in terms of concept drift detection and adaptation. In order to remedy this shortcoming, in this study we provide both synthetic and real world evolving stream datasets with different characteristics, which are suitable for testing and benchmark purposes. To the best of our knowledge, this is the only study to provide both synthetic and real world high dimensional, evolving stream datasets in such a systematic way. Moreover, most of the data stream clustering algorithms in the literature do not detect and report concept drift. In general, other stream clustering algorithms only adapt concept drift intuitively. We believe that it is possible to retrieve very valuable information from concept drift detection in several fields like patient tracking, border security using sensors and surveillance cameras etc. We also believe that data stream analysis is in its infancy yet. In the near future, we will encounter many different academic and professional studies, in the field of data stream analysis.

NoCStream employs mean shift for k prediction purpose. Mean shift is selected among its alternatives. It is possible to further evaluate alternative methods instead of mean shift. Moreover, both EmCStream and NoCStream do not detect outliers. Outlier detection is a popular research topic and it is left as a future work in this thesis.

REFERENCES

[1] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. USA: Prentice-Hall, Inc., 1988.

[2] J. C. Bezdek and J. M. Keller, “Streaming data analysis: Clustering or classification?,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 91–102, 2021.

[3] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[4] L. McInnes, J. Healy, and J. Melville, “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction,” *ArXiv e-prints*, feb 2018.

[5] A. Zubaroğlu and V. Atalay, “Online embedding and clustering of data streams,” in *Proceedings of the 2019 3rd International Conference on Big Data Research*, ICBDR 2019, (New York, NY, USA), p. 142–146, Association for Computing Machinery, 2019.

[6] M. Bahri, A. Bifet, S. Maniu, and H. M. Gomes, “Survey on feature transformation techniques for data streams,” in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20* (C. Bessiere, ed.), pp. 4796–4802, International Joint Conferences on Artificial Intelligence Organization, 7 2020. Survey track.

[7] S. S. Vempala, *The Random Projection Method*, vol. 65. American Mathematical Society, 2004.

[8] K. P. F.R.S., “Liii. on lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[9] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, “Feature hashing for large scale multitask learning,” in *Proceedings of the 26th Annual*

International Conference on Machine Learning, ICML '09, (New York, NY, USA), p. 1113–1120, Association for Computing Machinery, 2009.

[10] A. Zubaroğlu and V. Atalay, "Online embedding and clustering of evolving data streams," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 16, no. 1, pp. 29–44, 2023.

[11] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[12] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *In 2006 SIAM Conference on Data Mining*, vol. 2006, pp. 328–339, 04 2006.

[13] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pp. 81–92, 2003.

[14] F. Can, "Incremental clustering for dynamic information processing," *ACM Trans. Inf. Syst.*, vol. 11, p. 143–164, apr 1993.

[15] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, and J. a. Gama, "Data stream clustering: A survey," *ACM Comput. Surv.*, vol. 46, pp. 13:1–13:31, July 2013.

[16] F. Alam, R. Mehmood, I. Katib, and A. Albeshri, "Analysis of eight data mining algorithms for smarter internet of things (iot)," *Procedia Computer Science*, vol. 98, pp. 437 – 442, 2016.

[17] M. Ghesmoune, M. Lebbah, and H. Azzag, "State-of-the-art on clustering data streams," *Big Data Analytics*, vol. 1, p. 13, Dec 2016.

[18] M. Carnein, D. Assenmacher, and H. Trautmann, "An empirical comparison of stream clustering algorithms," in *Proceedings of the Computing Frontiers Conference*, CF'17, pp. 361–366, 2017.

[19] K. Yasumoto, H. Yamaguchi, and H. Shigeno, “Survey of real-time processing technologies of iot data streams,” *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[20] C. Nordahl, V. Boeva, H. Grahn, and M. Persson, “Evolvecluster: an evolutionary clustering algorithm for streaming data,” *Evolving Systems*, pp. 1–21, 11 2021.

[21] E. B. Gulcan and F. Can, “Unsupervised concept drift detection for multi-label data streams,” *Artificial Intelligence Review*, Jul 2022.

[22] Ö. Gözüaçık and F. Can, “Concept learning using one-class classifiers for implicit drift detection in evolving data streams,” *Artificial Intelligence Review*, Nov 2020.

[23] Ö. Gözüaçık, A. Büyükçakır, H. Bonab, and F. Can, “Unsupervised concept drift detection with a discriminative classifier,” in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM ’19, (New York, NY, USA), p. 2365–2368, Association for Computing Machinery, 2019.

[24] M. M. W. Yan, “Accurate detecting concept drift in evolving data streams,” *ICT Express*, vol. 6, no. 4, pp. 332–338, 2020.

[25] R. Paudel and W. Eberle, “An approach for concept drift detection in a graph stream using discriminative subgraphs,” *ACM Trans. Knowl. Discov. Data*, vol. 14, Sept. 2020.

[26] J. a. Gama, I. Žliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM Comput. Surv.*, vol. 46, pp. 44:1–44:37, Mar. 2014.

[27] R. N. Gemaque, A. F. J. Costa, R. Giusti, and E. M. dos Santos, “An overview of unsupervised drift detection methods,” *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 6, p. e1381, 2020.

[28] D. Puschmann, P. Barnaghi, and R. Tafazolli, “Adaptive clustering for dynamic iot data streams,” *IEEE Internet of Things Journal*, vol. 4, pp. 64–74, Feb 2017.

[29] J. d. Andrade Silva, E. R. Hruschka, and J. a. Gama, “An evolutionary algorithm for clustering data streams with a variable number of clusters,” *Expert Syst. Appl.*, vol. 67, pp. 228–238, Jan. 2017.

[30] A. Amini, H. Saboohi, T. Herawan, and T. Y. Wah, “Mudi-stream: A multi density clustering algorithm for evolving data stream,” *J. Netw. Comput. Appl.*, vol. 59, pp. 370–385, Jan. 2016.

[31] R. Hyde, P. Angelov, and A. MacKenzie, “Fully online clustering of evolving data streams into arbitrarily shaped clusters,” *Information Sciences*, vol. 382-383, pp. 96 – 114, 2017.

[32] C. Yin, L. Xia, S. Zhang, R. Sun, and J. Wang, “Improved clustering algorithm based on high-speed network data stream,” *Soft Computing*, Jul 2017.

[33] K.-S. Zhang, L. Zhong, L. Tian, X.-Y. Zhang, and L. Li, “DBIECM-an Evolving Clustering Method for Streaming Data Clustering,” *Amse Journals-Amse Iieta*, vol. 60, no. 1, pp. 239–254, 2017.

[34] M. Hassani, P. Spaus, A. Cuzzocrea, and T. Seidl, “Adaptive stream clustering using incremental graph maintenance,” in *Proceedings of the 4th International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications - Volume 41*, BIGMINE’15, pp. 49–64, 2015.

[35] M. Hassani, P. Spaus, A. Cuzzocrea, and T. Seidl, “I-hastream: Density-based hierarchical clustering of big data streams and its application to big graph analytics tools,” in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 656–665, May 2016.

[36] A. Zubaroğlu and V. Atalay, “Data stream clustering: a review,” *Artificial Intelligence Review*, vol. 54, pp. 1201–1236, Jul 2020.

[37] Y. Hozumi, R. Wang, C. Yin, and G.-W. Wei, “Umap-assisted k-means clustering of large-scale sars-cov-2 mutation datasets,” *Computers in Biology and Medicine*, vol. 131, p. 104264, 2021.

[38] A. Duraj and P. S. Szczepaniak, “Outlier detection in data streams — a comparative study of selected methods,” *Procedia Computer Science*, vol. 192, pp. 2769–2778, 2021.

[39] A. Forestiero, “Self-organizing anomaly detection in data streams,” *Information Sciences*, vol. 373, pp. 321–336, 2016.

[40] S. Guggilam, V. Chandola, and A. Patra, “Tracking clusters and anomalies in evolving data streams,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. n/a, no. n/a, 2021.

[41] M. Bahri, B. Pfahringer, A. Bifet, and S. Maniu, “Efficient Batch-Incremental Classification Using UMAP for Evolving Data Streams,” in *IDA 2020 - 18th International Symposium on Intelligent Data Analysis* (M. R. Berthold, A. Feelders, and G. Krempl, eds.), vol. 12080 of *LNCS - Lecture Notes in Computer Science*, (Konstanz / Virtual, Germany), pp. 40–53, Springer, Apr. 2020.

[42] J. Xu, G. Wang, T. Li, W. Deng, and G. Gou, “Fat node leading tree for data stream clustering with density peaks,” *Knowledge-Based Systems*, vol. 120, pp. 99 – 117, 2017.

[43] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, pp. 637–646, Oct 2016.

[44] W. Shi and S. Dustdar, “The promise of edge computing,” *Computer*, vol. 49, pp. 78–81, May 2016.

[45] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, pp. 30–39, Jan 2017.

[46] S. Ramirez-Gallego, B. Krawczyk, S. Garcia, M. Wozniak, and F. Herrera, “A survey on data preprocessing for data stream mining: Current status and future directions,” *Neurocomputing*, vol. 239, 02 2017.

[47] S. Mansalis, E. Ntoutsi, N. Pelekis, and Y. Theodoridis, “An evaluation of data stream clustering algorithms,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 11, no. 4, pp. 167–187, 2018.

[48] K. D. Modi and P. B. Oza, "Outlier analysis approaches in data mining," *International Journal of Innovative Research in Technology (IJIRT)*, vol. 3, pp. 6–12, 2017.

[49] J. A. Merino, *Streaming Data Clustering in MOA using the Leader Algorithm*. PhD thesis, Universitat Politècnica de Catalunya, 2015.

[50] P. Chauhan and M. Shukla, "A review on outlier detection techniques on data stream by using different approaches of K-Means algorithm," in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015.

[51] I. Souiden, Z. Brahmi, and H. Toumi, "A Survey On Outlier Detection In The Context Of Stream Mining : review Of Existing Approaches And Recommadations," in *Advances in Intelligent Systems and Computing*, 2016.

[52] P. Thakkar, J. Vala, and V. Prajapati, "Survey on Outlier Detection in Data Stream," *International Journal of Computer Applications*, vol. 136, no. 2, 2016.

[53] S. V. Bhosale, "A Survey: Outlier Detection in Streaming Data Using Clustering Approached," *(IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 5, pp. 6050–6053, 2014.

[54] S. Sadik and L. Gruenwald, "Research issues in outlier detection for data streams," *SIGKDD Explor. Newsl.*, vol. 15, pp. 33–40, Mar. 2014.

[55] X. Kong, Y. Bi, and D. H. Glass, "Detecting anomalies in sequential data augmented with new features," *Artificial Intelligence Review*, vol. 53, pp. 625–652, 2019.

[56] V. Christodoulou, Y. Bi, and G. Wilkie, "A fuzzy shape-based anomaly detection and its application to electromagnetic data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, pp. 3366–3379, Sep. 2018.

[57] E. Keogh, J. Lin, and A. Fu, "Hot sax: Efficiently finding the most unusual time series subsequence," in *Proceedings of the Fifth IEEE International Con-*

ference on Data Mining, ICDM '05, (USA), p. 226–233, IEEE Computer Society, 2005.

[58] M. Mousavi, A. Bakar, and M. Vakilian, "Data stream clustering algorithms: A review," *International Journal of Advances in Soft Computing and its Applications*, vol. 7, pp. 1–15, 2015.

[59] P. Kumar, "Data Stream Clustering in Internet of Things," *SSRG International Journal of Computer Science and Engineering*, vol. 3, no. 8, 2016.

[60] S. Ding, F. Wu, J. Qian, H. Jia, and F. Jin, "Research on data stream clustering algorithms," *Artif. Intell. Rev.*, vol. 43, pp. 593–600, Apr. 2015.

[61] H.-L. Nguyen, Y.-K. Woon, and W.-K. Ng, "A survey on data stream clustering and classification," *Knowledge and Information Systems*, vol. 45, pp. 535–569, Dec 2015.

[62] C. Fahy, S. Yang, and M. Gongora, "Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 2215–2228, 2019.

[63] A. R. Mahdiraji, "Clustering data stream: A survey of algorithms," *Int. J. Know.-Based Intell. Eng. Syst.*, vol. 13, p. 39–44, Apr. 2009.

[64] C. C. Aggarwal, "A survey of stream clustering algorithms," in *Data Clustering: Algorithms and Applications* (C. C. Aggarwal and C. K. Reddy, eds.), pp. 231–258, CRC Press, 2013.

[65] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," *SIGMOD Rec.*, vol. 25, pp. 103–114, June 1996.

[66] G. Karypis, E.-H. Han, and V. Kumar, "Chameleon a hierarchical clustering algorithm using dynamic modeling," *Computer*, vol. 32, pp. 68 – 75, 09 1999.

[67] P. Rodrigues, J. Gama, and J. Pedroso, "ODAC: Hierarchical Clustering of Time Series Data Streams," in *Proceedings of the Sixth SIAM International*

Conference on Data Mining (J. Ghosh, D. Lambert, D. Skillicorn, and J. Srivastava, eds.), vol. 2006 of *SIAM Proceedings Series*, pp. 499–503, 2006. Citations: crossref, dblp, scopus, wos.

[68] K. Udommanetanakit, T. Rakthanmanon, and K. Waiyamai, "E-stream: Evolution-based technique for stream clustering," in *Advanced Data Mining and Applications*, vol. 4632, pp. 605–615, Springer Berlin Heidelberg, 08 2007.

[69] W. Meesuksabai, T. Kangkachit, and K. Waiyamai, "Hue-stream: Evolution-based clustering technique for heterogeneous data streams with uncertainty," in *Advanced Data Mining and Applications*, pp. 27–40, Springer Berlin Heidelberg, 12 2011.

[70] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha, "Streaming-data algorithms for high-quality clustering," in *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pp. 685–, 2002.

[71] C. Ordonez, "Clustering binary data streams with k-means," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, (New York, NY, USA), p. 12–19, Association for Computing Machinery, 2003.

[72] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A framework for projected clustering of high dimensional data streams," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, pp. 852–863, VLDB Endowment, 12 2004.

[73] A. Zhou, F. Cao, W. Qian, and C. Jin, "Tracking clusters in evolving data streams over sliding windows," *Knowl. Inf. Syst.*, vol. 15, pp. 181–214, May 2008.

[74] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "Streamkm++: A clustering algorithm for data streams," *J. Exp. Algorithmics*, vol. 17, pp. 2.4:2.1–2.4:2.30, May 2012.

[75] X. Zhang, C. Furtlehner, C. Germain-Renaud, and M. Sebag, "Data stream clustering with affinity propagation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 7, pp. 1644–1656, 2014.

[76] L. Kaufman and P. J. Rousseeuw, *Clustering Large Applications (Program CLARA)*, ch. 3, pp. 126–163. John Wiley & Sons, Ltd, 1990.

[77] Y. Lu, Y. Sun, G. Xu, and G. Liu, "A grid-based clustering algorithm for high-dimensional data streams," in *Advanced Data Mining and Applications* (X. Li, S. Wang, and Z. Y. Dong, eds.), (Berlin, Heidelberg), pp. 824–831, Springer Berlin Heidelberg, 2005.

[78] Y. Sun and Y. Lu, "A grid-based subspace clustering algorithm for high-dimensional data streams," in *Web Information Systems – WISE 2006 Workshops* (L. Feng, G. Wang, C. Zeng, and R. Huang, eds.), (Berlin, Heidelberg), pp. 37–48, Springer Berlin Heidelberg, 2006.

[79] J. a. Gama, P. P. Rodrigues, and L. Lopes, "Clustering distributed sensor data streams using local processing and reduced communication," *Intell. Data Anal.*, vol. 15, pp. 3–28, Jan. 2011.

[80] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, (New York, NY, USA), p. 94–105, Association for Computing Machinery, 1998.

[81] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "Wavecluster: A wavelet-based clustering approach for spatial data in very large databases," *The VLDB Journal*, vol. 8, p. 289–304, Feb. 2000.

[82] W. Wang, J. Yang, and R. R. Muntz, "Sting: A statistical information grid approach to spatial data mining," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, (San Francisco, CA, USA), p. 186–195, Morgan Kaufmann Publishers Inc., 1997.

[83] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in

Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’07, pp. 133–142, 2007.

[84] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, “Density-based clustering of data streams at multiple resolutions,” *ACM Trans. Knowl. Discov. Data*, vol. 3, July 2009.

[85] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu, “Incremental clustering for mining in a data warehousing environment,” in *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB ’98, (San Francisco, CA, USA), p. 323–333, Morgan Kaufmann Publishers Inc., 1998.

[86] L. Duan, D. Xiong, J. Lee, and F. Guo, “A local density based spatial clustering algorithm with noise,” in *2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 4061–4066, 2006.

[87] L. Liu, H. Huang, Y. Guo, and F. Chen, “rdenstream, a clustering algorithm over an evolving data stream,” in *2009 International Conference on Information Engineering and Computer Science*, pp. 1–4, 2009.

[88] A. Namadchian and G. Esfandani, “Dsclu: A new data stream clustring algorithm for multi density environments,” *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 83–88, 2012.

[89] H. Wang, Y. Yu, Q. Wang, and Y. Wan, “A density-based clustering structure mining algorithm for data streams,” in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, BigMine ’12, (New York, NY, USA), p. 69–76, Association for Computing Machinery, 2012.

[90] C. Isaksson, M. Dunham, and M. Hahsler, “Sostream: Self organizing density-based clustering over data stream,” in *Machine Learning and Data Mining in Pattern Recognition*, vol. 7376, Springer Berlin Heidelberg, 07 2012.

[91] D. Tasoulis, G. Ross, and N. Adams, “Visualising the cluster structure of data streams,” in *Advances in Intelligent Data Analysis VII*, vol. 4723, pp. 81–92, Springer Berlin Heidelberg, 09 2007.

[92] D. Fisher, “Iterative optimization and simplification of hierarchical clustering,” *J Artif Intell Res*, vol. 4, 05 1996.

[93] A. Zhou, F. Cao, Y. Yan, C. Sha, and X. He, “Distributed data stream clustering: A fast em-based approach,” in *2007 IEEE 23rd International Conference on Data Engineering*, pp. 736–745, 2007.

[94] X. H. Dang, V. C. S. Lee, W. K. Ng, and K. L. Ong, “Incremental and adaptive clustering stream data over sliding window,” in *Database and Expert Systems Applications* (S. S. Bhowmick, J. Küng, and R. Wagner, eds.), (Berlin, Heidelberg), pp. 660–674, Springer Berlin Heidelberg, 2009.

[95] S. U. Din, J. Shao, J. Kumar, W. Ali, J. Liu, and Y. Ye, “Online reliable semi-supervised learning on evolving data streams,” *Information Sciences*, vol. 525, pp. 153 – 171, 2020.

[96] C. G. Bezerra, B. S. J. Costa, L. A. Guedes, and P. P. Angelov, “An evolving approach to data streams clustering based on typicality and eccentricity data analytics,” *Information Sciences*, vol. 518, pp. 13 – 28, 2020.

[97] T. Kim and C. H. Park, “Anomaly pattern detection for streaming data,” *Expert Systems with Applications*, vol. 149, p. 113252, 2020.

[98] E. Parzen, “On estimation of a probability density function and mode,” *Ann. Math. Statist.*, vol. 33, pp. 1065–1076, 09 1962.

[99] M. Rosenblatt, “Remarks on some nonparametric estimates of a density function,” *Ann. Math. Statist.*, vol. 27, pp. 832–837, 09 1956.

[100] H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi, “Test of page-hinckley, an approach for fault detection in an agro-alimentary production system,” in *2004 5th Asian Control Conference (IEEE Cat. No.04EX904)*, vol. 2, pp. 815–818 Vol.2, July 2004.

[101] Q. Song and N. Kasabov, “Ecm - a novel on-line, evolving clustering method and its applications,” in *In M. I. Posner (Ed.), Foundations of cognitive science*, pp. 631–682, The MIT Press, 2001.

[102] M. Hassani, P. Spaus, and T. Seidl, "Adaptive multiple-resolution stream clustering," in *Machine Learning and Data Mining in Pattern Recognition*, pp. 134–148, 2014.

[103] H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer, "An effecive evaluation measure for clustering on evolving data streams," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pp. 868–876, 2011.

[104] Citi Bike NYC, "Citi Bike: NYC's Official Bike Sharing System." `https://www.citibikenyc.com/`, 2013. Accessed: 2018-03-25.

[105] Citi Bike System Data. `https://www.citibikenyc.com/system-data`, 2013. Accessed: 2018-03-25.

[106] Meetup, "We are what we do | Meetup." `https://www.meetup.com/`, 2002. Accessed: 2018-03-25.

[107] Meetup Stream, "Extend your community | Meetup." `https://www.meetup.com/meetup_api/docs/stream/2/rsvps/`, 2002. Accessed: 2018-03-25.

[108] National Weather Service (NWS), "National Weather Service." `https://www.weather.gov/`, 1870. Accessed: 2018-03-25.

[109] NWS Public Alerts, "NWS Public Alerts." `https://alerts.weather.gov/`, n.d. Accessed: 2018-03-25.

[110] X. H. Zhu, "Stream Data Mining Repository." `http://www.cse.fau.edu/~xqzhu/stream.html`, 2010. Accessed: 2018-03-25.

[111] Massive Online Analysis (MOA), "Moa - Machine Learning for Data Streams." `https://moa.cms.waikato.ac.nz/`, 2014. Accessed: 2018-03-25.

[112] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, Aug. 2010.

[113] Moa Stream Generators, "Moa: Package moa.stream.generators." `https://www.cs.waikato.ac.nz/~abifet/MOA/API/`

namespacemoa_1_1streams_1_1generators.html, 2014. Accessed: 2018-03-25.

[114] A Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD). https://crawdad.org/keyword-sensor-network.html, n.d. Accessed: 2018-03-25.

[115] Waikato Environment for Knowledge Analysis, "Weka 3 - Data Mining With Open Source Machine Learning Software in Java." https://www.cs.waikato.ac.nz/ml/weka/, 1993. Accessed: 2018-03-25.

[116] RapidMiner, "Data Sicence Platform - RapidMiner." https://rapidminer.com/, 2001. Accessed: 2018-03-25.

[117] C. Bockermann, "RapidMiner Streams Plugin." https://sfb876.de/streams/doc/rapidminer.html, 2018. Accessed: 2018-03-25.

[118] R, "R - The R Project for Statistical Computing." https://www.r-project.org/, 1993. Accessed: 2018-03-25.

[119] Janardan and S. Mehta, "Concept drift in streaming data classification: Algorithms, platforms and issues," *Procedia Computer Science*, vol. 122, pp. 804 – 811, 2017. 5th International Conference on Information Technology and Quantitative Management, ITQM 2017.

[120] B. R. Prasad and S. Agarwal, "Stream data mining: platforms, algorithms, performance evaluators and research trends," *International journal of database theory and application*, vol. 9, no. 9, pp. 201–218, 2016.

[121] Apache Storm. http://storm.apache.org/, 2011. Accessed: 2018-03-25.

[122] Apache Spark, "Apache Spark lightning-fast cluster computing." https://spark.apache.org/, 2012. Accessed: 2018-03-25.

[123] Spark Streaming, "Apache Spark Streaming." https://spark.apache.org/streaming/, 2012. Accessed: 2018-03-25.

[124] Apache Samza, "Samza." https://samza.apache.org/, 2013. Accessed: 2018-03-25.

[125] N. Ramesh, "Apache Samza, LinkedIn's Framework for Stream Processing - The New Stack." https://thenewstack.io/apache-samza-linkedins-framework-for-stream-processing/, 2013. Accessed: 2018-03-25.

[126] Apache Kafka. https://kafka.apache.org/, 2011. Accessed: 2018-03-25.

[127] AmazonKinesis, "Amazon Kinesis." https://aws.amazon.com/kinesis/, 2013. Accessed: 2018-03-25.

[128] IBM Infosphere, "Streaming Analytics - Overview - IBM Cloud." https://www.ibm.com/cloud/streaming-analytics, 1996. Accessed: 2018-03-25.

[129] B. Gedik and H. Andrade, "A model-based framework for building extensible, high performance stream processing middleware and programming language for ibm infosphere streams," *Softw. Pract. Exper.*, vol. 42, pp. 1363–1391, Nov. 2012.

[130] Google Cloud Stream, "Streaming Analytics for Real Time Insights - Google Cloud." https://cloud.google.com/solutions/big-data/stream-analytics/, 2012. Accessed: 2018-03-25.

[131] Microsoft Azure Stream Analytics, "Stream Analytics - Real Time Data Analytics - Microsoft Azure." https://azure.microsoft.com/en-us/services/stream-analytics/, 2012. Accessed: 2018-03-25.

[132] M. Hahsler, M. Bolanos, and J. Forrest, *streamMOA: Interface for MOA Stream Clustering Algorithms*, 2015. R package version 1.1-2.

[133] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[134] K. S. Killourhy and R. A. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pp. 125–134, 2009.

[135] K. Fukunaga and L. Hostetler, “The estimation of the gradient of a density function, with applications in pattern recognition,” *IEEE Transactions on Information Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[136] Y. Cheng, “Mean shift, mode seeking, and clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[137] R. L. Thorndike, “Who belongs in the family?,” *Psychometrika*, vol. 18, pp. 267–276, Dec 1953.

[138] K. E. O’grady, “Measures of explained variance: Cautions and limitations.,” *Psychological Bulletin*, vol. 92, pp. 766–777, 1982.

[139] S. K. Kingrani, M. Levene, and D. Zhang, “Estimating the number of clusters using diversity,” *Artif. Intell. Res.*, vol. 7, no. 1, p. 15, 2018.

[140] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction To Cluster Analysis*. Wiley, New York, 01 1990.

[141] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53 – 65, 1987.

[142] G. W. Milligan and M. C. Cooper, “An examination of procedures for determining the number of clusters in a data set,” *Psychometrika*, vol. 50, pp. 159–179, Jun 1985.

[143] R. Tibshirani, 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*, vol. 63, pp. 411–423, 02 2001.

[144] M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs, “Nbclust: An r package for determining the relevant number of clusters in a data set,” *Journal of Statistical Software*, vol. 61, no. 6, p. 1–36, 2014.

[145] F. Can and E. A. Ozkarahan, “Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases,” *ACM Trans. Database Syst.*, vol. 15, p. 483–517, dec 1990.

[146] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[147] J. E. Chacón and P. Monfort, “A comparison of bandwidth selectors for mean shift clustering,” 10 2013.

[148] B. Frey and D. Dueck, “Clustering by passing messages between data points,” *Science (New York, N.Y.)*, vol. 315, pp. 972–6, 03 2007.

[149] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD’96, p. 226–231, AAAI Press, 1996.

[150] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, “Dbscan revisited, revisited: Why and how you should (still) use dbscan,” *ACM Trans. Database Syst.*, vol. 42, jul 2017.

[151] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “Optics: Ordering points to identify the clustering structure,” *SIGMOD Rec.*, vol. 28, p. 49–60, jun 1999.

[152] L. E. Brito Da Silva, N. M. Melton, and D. C. Wunsch, “Incremental cluster validity indices for online learning of hard partitions: Extensions and comparative study,” *IEEE Access*, vol. 8, pp. 22025–22047, 2020.

[153] H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer, “An effective evaluation measure for clustering on evolving data streams,” in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’11, (New York, NY, USA), p. 868–876, Association for Computing Machinery, 2011.

[154] L. Hubert and P. Arabie, “Comparing partitions,” *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[155] W. M. Rand, “Objective criteria for the evaluation of clustering methods,” *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[156] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[157] A. Pothen, H. D. Simon, and K.-P. Liou, “Partitioning sparse matrices with eigenvectors of graphs,” *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 3, pp. 430–452, 1990.

[158] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[159] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.