Author/s:
Chenaghlou, Milad

Title:
Data stream clustering and anomaly detection

Date:
2019

Persistent Link:
https://hdl.handle.net/11343/237418

Data Stream Clustering and Anomaly Detection

Milad Chenaghlou

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

October 2019

Abstract

DATA stream clustering and anomaly detection have grown in importance with the advent of hardware and software technologies that capture and generate continuous streams of sensor data. Stream data mining problems are particularly important in application domains such as network intrusion detection, road traffic analysis, social media analysis and military surveillance systems. However, a number of open challenges need to be addressed in order for stream clustering and anomaly detection to be effectively used in those applications.

One of the main challenges regarding data stream clustering and anomaly detection is computational efficiency. In non-stationary data streams in which patterns change over time, algorithms need to identify and adapt to such changes. This requires the ability to test whether the current model accurately represents observed patterns in the stream in an efficient manner. To cope with this challenge, the processing time of the algorithm must scale linearly with the number of observed data points and the memory requirements should be constant. Accordingly, we propose an efficient data stream anomaly detection algorithm that scales linearly with the number of data points.

A second challenge is that in many application domains, it is desired that an online clustering algorithm be able to both update the model and identify anomalies in real-time. Current state-of-the-art online clustering algorithms either do not detect anomalies or detect them in a separate process when triggered by the user. Moreover, they only consider the spatial proximity of data points, rather than analyse the evolution of patterns in the stream. We propose an online clustering algorithm that considers the temporal proximity of observations as well as their spatial proximity to identify anomalies in real-time. It identifies the evolution of clusters in noisy streams, incrementally updates the model and calculates the minimum window length over the evolving data stream without jeopardizing performance.

Another challenge for clustering data streams is when the number of dimensions increases. In high-dimensional data, conventional distance measures become less meaningful, which limits the effectiveness of distance-based clustering methods. One approach to this challenge is the use of subspace clustering algorithms, which identify a small number of features that can best explain the clusters in the stream. Subspace clustering algorithms for streaming environments address this challenge by reducing the infinite search space of arbitrarily-oriented subspaces into a bounded number of axis-parallel (or projected) subspaces. Accordingly, we propose an arbitrarily oriented subspace clustering algorithm for time-series streams of unbounded length. This algorithm is incremental which makes it suitable for streaming environments, and has lower memory requirements compared to state-of-the-art subspace clustering techniques. In particular, our algorithm can identify emerging subspace clusters, as well as clusters that overlap but appear in different subspaces and timespans.

Finally, many data stream clustering algorithms require user-defined threshold parameters to identify and adapt to changes in non-stationary data streams. We propose a novel algorithm in the form of a control structure that can be mounted on other online clustering algorithms to guide them through the changes in the stream. This control structure uses an incremental cluster validity index as a basis for detecting and monitoring changes in the stream.

In summary, we propose a range of efficient online anomaly detection and online clustering algorithms for streaming data. These algorithms are suitable for unlabeled data streams, which arise in a variety of real-world applications, and have the flexibility to be used in non-stationary environments where patterns emerge and change over time.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Milad Chenaghlou, October 2019

Acknowledgements

I would like to express my special gratitude and appreciation to the many people who, in one way or another, have supported me in this journey. Throughout writing this thesis, I have received a great deal of support and assistance from my supervisors Prof. Christopher Leckie, Dr. Masud Moshtaghi, Dr. Mahsa Salehi and Dr. Zahra Ghafoori whose expertise have been invaluable in formulating of my research. They patiently and continuously supported my PhD study and research, and motivated me over this time. I am extremely grateful to Prof. James C. Bezdek for his insights and valuable guidance during his visits to the University of Melbourne. I would also like to express my gratitude to my advisory committee chair Dr. Sean Maynard for his guidance and support. I would like to thank Masud Moshtaghi again, for giving this opportunity to me and believing in me. You truly have inspired me by sharing your words of wisdom, to invoke change and share happiness. I am your Samurai.

I would also like to thank my fellow students at the University of Melbourne for their support and help throughout my study. And my friends Kamyar SabriLaghaee, Ehsan Shareghi, Emad Laghaee, Mojtaba Aryaee, the Chaman community including Masud Bahrami, Kaveh Kasebian, Fazli Hatamzadeh, Mansour Tabrizi, and John Walkrov, you have my deepest feeling of gratitude.

A special feeling of gratitude goes to my brother, Behzad, for his inspiration and support. I also would like to thank my entire family for their encouragement and support: my parents MohammadBagher and Parvin, my brothers Behrang and Masud, and my parents and brother in-law Fereydoun, Alieh and Behnoud.

Lastly, and most importantly, I wish to thank my loving wife, Farnaz, who has been with me all these years and has made them the best years of my life. She was beside me at every step of my journey, believed in me, supported me and loved me. To her I dedicate this thesis

Preface

The list of research papers that have arisen from the contribution chapters of this thesis, Chapters 3 to 6, are:

P1. **Milad Chenaghlou**, Masud Moshtaghi, Christopher Leckie, and Mahsa Salehi. "An efficient method for anomaly detection in non-stationary data streams", in IEEE Global Communications Conference, GLOBECOM (2017).

P2. **Milad Chenaghlou**, Masud Moshtaghi, Christopher Leckie, and Mahsa Salehi. "Online clustering for evolving data streams with online anomaly detection". Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2018.

P3. **Milad Chenaghlou**, Masud Moshtaghi, Mahsa Salehi, Zahra Ghafoori, and Christopher Leckie. "SCTS: An incremental subspace clustering algorithm for high-dimensional timeseries", *submitted* to IEEE Transactions on Cybernetics

P4. **Milad Chenaghlou**, Masud Moshtaghi, Mahsa Salehi, Zahra Ghafoori, James C. Bezdek, and Christopher Leckie. "Incremental Monitoring and Control of Online Clustering Algorithms", *submitted* to Pattern Recognition journal.

To my dearest wife and soulmate, Farnaz,

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

Symbol	Description
$X = \{x_1, x_2, \ldots, x_n\}$	dataset with n data points
$V = \{v_1, v_2, \ldots, v_k\}$	set of cluster prototypes produced by an algorithm
d	number of dimensions in a dataset
M_{fkn}	fuzzy membership matrix
M_{ckn}	crisp membership matrix
n	number of data points in a dataset
k	number of clusters in a dataset
wl	window length
κ	number of facilities in online facility location problem
LS	linear sum of data points
SS	sum of squared data points
LST	sum of time stamps of data points
SST	sum of squared time stamps of data points
t	time
w	weight of data points
μ	mean of a distribution
Σ	covariance matrix of a distribution
m	mean of a sample of a distribution
S	covariance of a sample of a distribution
z	level-set that sets the boundary of a hyper-ellipsoidal cluster
P	probability value
R	radius value around a cluster center
ε	distance to a data point
q	number of data points for identifying distance-based anomalies
$DB(q, \varepsilon)$	distance-based anomaly
L	number of buckets or parts into which a dimension is divided
λ	forgetting factor

$A = \{a_1, a_2, \ldots, a_l\}$	set of underlying mixture distributions
l	number of underlying distributions
$B = \{b_1, b_2, \ldots, b_{l'}\}$	set of subspaces
l'	number of subspaces
$C = \{c_1, c_2, \ldots, c_k\}$	set of clusters
E	entropy threshold value
W	window of data points
ζ	number of components in mixture of distributions
$\Theta = \{\ \theta_{ji}, i = 1, \ldots, \zeta_i\}$	set of parameters of a distribution with ζ_j components
$\Phi = \{\ \theta_{ji}, i = 1, \ldots, \zeta_i\}$	set of mixture weights of a distribution with ζ_i components
α	confidence level
CR	confidence region
$\tilde{d}$	dimension of a subspace of the ambient space

Chapter 1

Introduction

CLUSTER analysis is a cornerstone of exploratory data analysis, which aims to identify natural structure (clusters) in a dataset [57]. It facilitates an understanding of the dominant patterns and underlying behaviour in a system. Take a financial organization with a large customer base as an example. Clustering helps study the data by partitioning the customers into several groups with regard to specific features, such as gender, age, spending habits and so on. These partitions represent groups of customers with shared needs and characteristics which then can be used in formulating policies that are tailored to each group of customers.

A related task when analysing and exploring large datasets is detecting unexpected and unusual data points (anomalies). Anomalies (also known as outliers or exceptions) in data are referred to as data points or patterns that do not conform to the normal behaviour of a system [115]. Anomalies are interesting in many application domains because they often provide useful information about the dataset at hand. For instance, in a financial organization dataset, anomalous data points (customers) may correspond to illegitimate and fraudulent activities, where the customer behaviour does not conform to the normal behaviour of customers. Clustering and anomaly detection play crucial roles in exploratory data analysis, and there is a rich body of research on these two fields. However, the exponential growth in the volume of data that has to be analysed has made traditional techniques less effective and practical [127].

To illustrate the scale of the data that is available for analysis, consider that every day, 500 million tweets are sent on Twitter[1], 294 billion emails are sent[2], and 4 petabytes of data are created on Facebook[3]. Another example is the *Internet of Things* (IoT), which can potentially connect almost any device such as coffee makers, road traffic sensors, and wearable devices to the Internet for

[1] http://tiny.cc/z27f7y [2] http://tiny.cc/n47f7y [3] http://tiny.cc/n57f7y

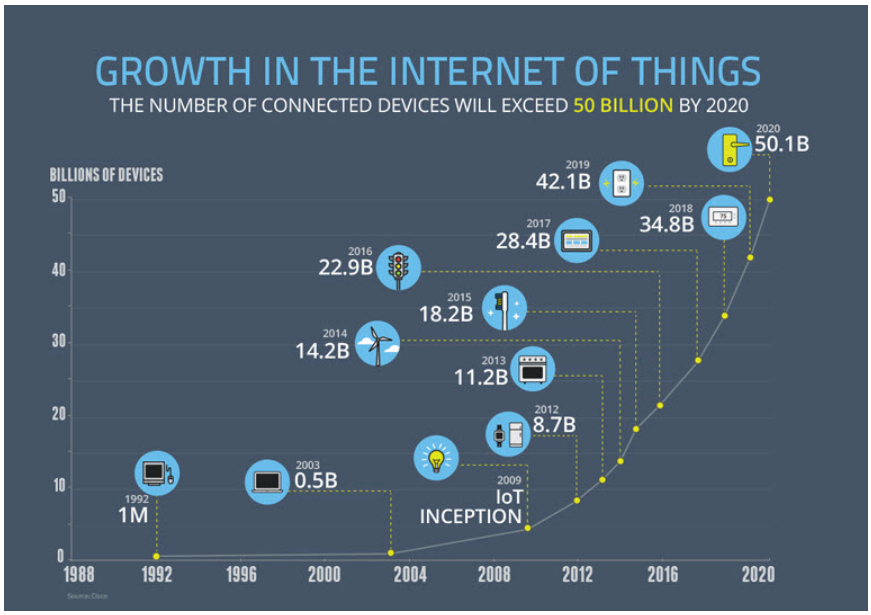


Figure 1.1: The exponential growth in the number of devices connected to the IoT [4]

better control, management or monitoring. In the past decade, companies have made substantial investments to continuously collect and process data for better understanding their businesses, and gaining a competitive edge over rival organizations. Figure 1.1 illustrates the rapid growth in the number of devices connected to the IoT from its inception in 2009. In 2017 it is estimated that there were 28.4 billion devices connected to the IoT, and by 2020 this number is projected to be more than 50 billion. A major challenge is that these devices can generate data as a continuous stream, which hinders the application of traditional clustering and anomaly detection techniques that are designed for batch analysis. In the next section, we illustrate the key requirements for new clustering and anomaly detection techniques within the context of IoT data analysis.

1.1 Motivating Example

IoT is providing an enabling platform for a new generation of services such as smart homes, wearable devices, connected cars, smart cities, and digital healthcare. These applications and services require a reliable high-performance Internet connection. However, low-cost IoT devices may provide limited support for security [146, 107]. The prevalence of IoT devices in our everyday

[4] https://goo.gl/e4a6DW

lives on one hand, and their vulnerability to cyber attacks and intrusions on the other hand, has created an interesting challenge in terms of detecting malicious behaviour. In 2015, a team of researchers were able to "hijack" a Jeep SUV through a vulnerability in a firmware update of the Controller Area Network (CAN bus) of the vehicle[5]. They could remotely speed up, slow down or control the steering wheel of the vehicle. Another example is the Mirai Botnet [9] in which a large number of online webcams were infected to form a "botnet" of devices that could launch a massive distributed denial-of-service attack. It is estimated that it infected over 600,000 IoT devices at its peak. Since detecting such intrusions is crucial for the success of IoT, intrusion detection systems must be designed to identify such abnormal activities in an efficient and accurate manner.

A common solution for securing a private network is using security products such as firewalls and intrusion detection systems. A firewall is a system that monitors each and every packet, inbound or outbound to the private network, and filters packets if they do not match the rules defined in the firewall [129] (Chapter 9). Signature based intrusion detection systems create a database of known attacks and scan the packets in the network for similar patterns [66]. A packet is flagged as an intrusion if it has high similarity to a known attack type in the database. Although these are the basic elements for securing a private network, they have limited effectiveness against new attacks that have not been seen before.

Moreover, these systems tend to detect intrusions at the packet level, and are less effective at identifying the combined behaviour of legitimate packets in the network. In other words, they do not consider the collective behaviour of a set of legitimate packets that may translate to an attack. For instance, the Mirai Botnet was a DDoS (Distributed Denial-of-Service) attack, where a large number of infected devices (a botnet) flooded the targeted servers with legitimate requests. This attack disrupted the content distribution network that hosts websites such as Twitter, The Guardian, Netflix, Reddit and CNN for almost a full day [9]. A major challenge in detecting this attack at the traffic source was how to detect the abnormal traffic patterns that were being generated by the IoT devices.

A second approach for detecting intrusions in a network is to monitor the data generated by a set of features defined for the network and construct a model of normal behaviour. For instance, recurring activities in the network such as video streaming, downloading files, or web surfing will result in clusters of data in the defined feature space. Therefore, clusters of data are assumed to

[5] http://tiny.cc/3e3h7y

be normal and data points that are far from these clusters represent rare activities (anomalies) that require further investigation [66]. In the Mirai Botnet attack, the sporadic change in bandwidth usage of those botnets is an example of a suspicious activity that does not conform to the normal behaviour of those IoT devices [9]. This motivates monitoring and detecting anomalies in network data to identify potential threats that may pass pre-defined security measures. However, a major problem for traditional clustering and anomaly detection algorithms is how to process the huge volume of data that is transferred in modern-day networks [127]. The main challenges for clustering and anomaly detection in such networks are as follows.

Challenge 1: Efficiency. IoT devices transmit data over the Internet for an indefinite period of time. Hence, this can be regarded as an unbounded stream of data, which is also referred to as a *data stream*. However, since these devices need to be cost effective, they often have limited memory and computational capabilities. Therefore, clustering and anomaly detection algorithms aimed at this kind of environment must be sufficiently efficient to (i) perform all calculations in limited memory, and (ii) have low computational complexity to keep up with the arrival rate of the data stream. This means that their time and memory complexity should not depend on retaining a long history of the observed data points, otherwise, they can become slow and inefficient as they process more data. Accordingly, such algorithms should process each data point (or a small batch of data points) quickly and discard it as soon as possible.

Challenge 2: Real-time Anomaly Detection. In many application domains it is imperative to identify intrusions as soon as they occur. This has a major impact on preventing or controlling their damage to the system. For instance, in the Mirai Botnet attack, if the attack was identified sooner, its spread could be prevented by rebooting the device or preventing it from accessing the Internet. The challenge for identifying anomalies in real-time is that such environments are often nonstationary, i.e., the underlying patterns in the data change over time. Such changes make it difficult to discern anomalies from evolving clusters that represent normal activities in the network. Current methods for stream clustering do not detect anomalies automatically, i.e., anomalies are detected in a separate process in an off-line manner. Hence, it is crucial for stream clustering algorithms to track and update evolving clusters in the stream and identify anomalies as soon as possible.

Challenge 3: High-Dimensional Data. The data in intrusion detection often has many attributes

(features). When dealing with high-dimensional data a range of problems arise that are referred to as *the curse of dimensionality* [128]. One problem is that distance measures become less informative when the computations are performed in high-dimensional spaces. Therefore dimensionality reduction techniques are used to reduce the number of features and define a subspace for each cluster. Another problem is that complex patterns may form in subspaces of the features, which makes identifying clusters in these subspaces more challenging. For instance, Figure 1.2 illustrates two complex scenarios that current state-of-the-art algorithms for subspace clustering have difficulty identifying. Timespans *A* and *B* in Figure 1.2 represent two 2-dimensional subspaces *F1* and *F2* with a single cluster that are attributed to normal activities. The first scenario comprises subspaces with multiple clusters, where each cluster may represent differing activities, which is depicted in timespan *C* of Figure 1.2. In this timespan, a red cluster that is attributed to an attack cluster has emerged in a subspace that is already identified as a subspace with a normal cluster, which may threaten the security of the network if it remains undetected. Another challenge is identifying overlapping clusters in different subspaces. This is depicted in timespan *D* of Figure 1.2 where the 1-dimensional cluster in Subspace *F3* that is attributed to an attack activity completely overlaps with the already identified normal cluster in Subspace *F2*. Although these data points overlap, they have occurred in two different timespans and two different subspaces. Current algorithms that address subspace clustering have difficulty identifying such clusters.

Challenge 4: Parameter Setting. A computer network is a dynamic environment where various patterns may emerge, evolve and die out in the stream. These patterns are not necessarily similar, e.g., the clusters in the data may differ in shape or size. Most state-of-the-art stream clustering algorithms require user-specified input parameters related to the shape, radius or density of clusters. For instance, ε for the radius of clusters, or *MinPts* for the minimum number of points around the ε neighbourhood of a data point to be regarded as a cluster [26]. These parameters serve as thresholds to identify the regions in the feature space with high densities of data points. However, finding "good" values for these parameters is challenging. Setting such threshold parameters usually requires expert knowledge which is expensive, or many passes over the data which is infeasible in a streaming environment.

The main contributions of this thesis address the above-mentioned challenges in data stream clustering and data stream anomaly detection. These challenges are addressed in the following

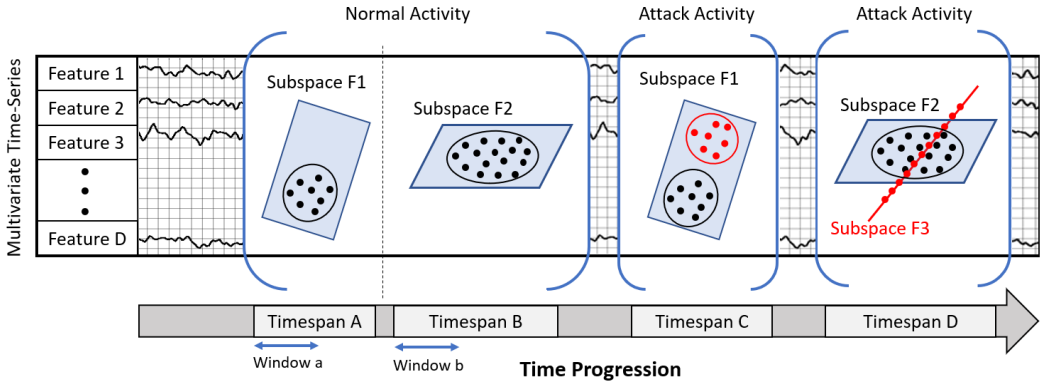


Figure 1.2: An illustration of the change of patterns in a high-dimensional multivariate streaming data for a hypothetical intrusion detection scenario. The data points in different timespans form different subspaces. The data points in timespan *A* form a 2-dimensional subspace *F1*. The data points in timespan *B* form a differently oriented subspace *F2*. In timespan *C* an attack cluster (the top cluster in red) emerges in a previously modelled subspace (an illustration of a multi-cluster subspace). The data points in timespan *D* represent an attack cluster (in Subspace *F3*) that overlaps with the normal cluster in *F2* (an illustration of overlapping clusters in different subspaces)

contribution chapters of the thesis as depicted in Figure 1.3.

1.2 Chapter 2 - Background and Literature Review

This chapter defines the preliminaries and main concepts of this thesis, and reviews the related literature. We first formally define data streams and explain the clustering problem. Then we review the current algorithms in the area of data stream clustering. We formally define anomalies and provide a thorough literature review of the methods and techniques for data stream anomaly detection. This chapter concludes by providing a taxonomy of algorithms for both data stream clustering and data stream anomaly detection, and analysing existing gaps in clustering and anomaly detection research for streaming data.

1.3 Chapter 3 - Efficient Anomaly Detection in Non-Stationary Data Streams

Anomaly detection techniques can be divided into two main categories: *proximity-based* methods, and *model-based* methods. Proximity-based algorithms find observations that are isolated from

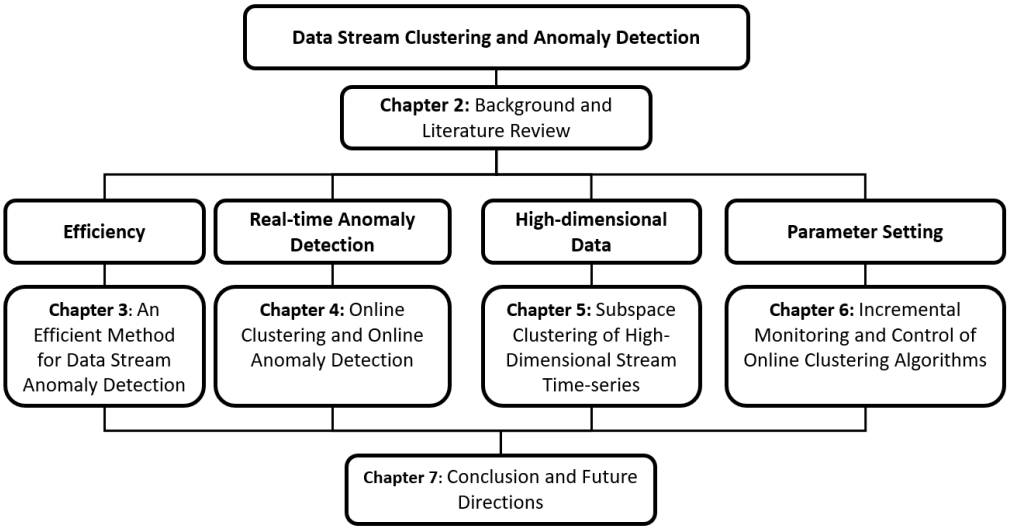


Figure 1.3: Organization of the thesis

the rest of the data and label them as anomalies. While these algorithms are generally accurate, their time and memory complexity restricts their practicality on unbounded data streams [122]. In contrast, model-based algorithms create a model of normal data and update it as the input data arrives. These algorithms generally use a window (a buffer) of data points that slides over the stream. As the window slides, subsequent data points are compared with the model, and labelled anomalous if they do not conform to it. In non-stationary environments, one challenge is to identify changes in normal patterns and update the model. However, this should be performed in an efficient manner, so it does not adversely affect the algorithm's time and memory complexity.

In this contribution chapter, we introduce a novel and time efficient approach called selective clustering to avoid unnecessary clustering and thus, increase computational efficiency on large data streams. Instead of building a new profile of clusters in each window, we check if any already existing clusters in the current model can explain the data in the window. Therefore, the time complexity of the proposed algorithm in processing a window of data is independent of previously observed data points, and only depends on the size of the model. In other words, the time required by the algorithm to detect anomalies after some initial time period remains constant for the rest of the stream, i.e., when all clusters have been modelled. Our algorithm has been tested on large-scale synthetic and real life datasets, and has comparable accuracy while being faster than current benchmarks. Accordingly, the contributions of this research are two-fold:

- We propose a novel method for selective clustering by reusing the current set of clusters rather than performing clustering on all of the data in each window
- We propose an algorithm with a memory complexity independent of the observed data, and time complexity are linear with respect to the number of data points

The outcomes of this contribution chapter are presented in paper P1 in the Preface.

1.4 Chapter 4 - Online Clustering for Evolving Data Streams with Online Anomaly Detection

Online clustering algorithms aim to find cluster structures in a data stream in real-time. They process newly arriving data points, update their model and then forget these data points as new inputs become available. A major challenge for these algorithms is to identify anomalies on the fly. Data stream clustering algorithms such as sequential k-means [93] and the *Adaptive Resonance Theory* (ART) [29] do not identify anomalies at all. Algorithms that do anomaly detection on data streams, such as [26, 81, 150], identify anomalies in a secondary process, i.e., they store the statistics of data points in specific data structures and perform anomaly detection in an off-line manner. Although this secondary process is useful, the significance of identifying anomalies deteriorates over time in many applications.

In this chapter, we propose an algorithm called OnCAD that considers the *temporal proximity* of observations as well as their spatial proximity to identify anomalies in an online manner. Our algorithm incrementally updates the model and identifies the evolution of clusters. It takes an input parameter to calculate the minimum window length without jeopardizing performance. The input parameter is the minimum mixture weight of a component in a mixture distribution that the user desires to be detected by the algorithm. The minimum window length is calculated using an *Elliptical Confidence Region* (ECR) [59] with a high confidence score. Accordingly, the contributions of this chapter are two-fold:

- We propose an online clustering algorithm with real-time anomaly detection
- We propose a method for calculating the minimum window length without sacrificing performance

The outcomes of this chapter are presented in paper P2 in the Preface.

1.5 Chapter 5 - Subspace Clustering of High-Dimensional Time-Series

When the number of features is large for a dataset, patterns of data often form in a subspace of the ambient space. *Subspace clustering* addresses the challenges of clustering in high-dimensional data as often a small set of features can accurately define the underlying clusters [139]. Subspace clustering algorithms often project (map) the data points into a subspace of the ambient space where distance measures work effectivly. The downside of the main subspace clustering algorithms [43, 65, 85, 88, 91, 132, 154] is that they work on small datasets. In a streaming environment, finding subspaces still remains a challenge. Subspace clustering algorithms for streaming environments alleviate this challenge by reducing the infinite search space of arbitrarily-oriented subspaces into a bounded number of axis-parallel (or projected) subspaces.

In this chapter we propose the first arbitrarily oriented subspace clustering algorithm for time-series streams of unbounded length. This method, called SCTS, is incremental (suitable for streams), and consumes significantly lower memory in comparison to state-of-the-art subspace clustering techniques. We employ a sliding window on the time-series that constantly calculates the principal components (PCs) in the window of fixed length (the PCs that define the subspace) and updates them. As time passes, new subspaces may emerge, evolve, or change in orientation. We present a novel technique to identify such changes in subspaces as the stream evolves. We identify two challenging scenarios for current state-of-the-art subspace clustering algorithms: (a) identifying subspaces with several clusters, and (b) differentiating overlapping clusters in dependent subspaces. SCTS uses the temporal dependency of data points in time-series to effectively identify these two cases. We demonstrate that identifying these two cases enhances the performance in an intrusion detection setting. Our experiments on synthetic as well as real-world datasets show that SCTS outperforms the current state-of-the-art subspace clustering algorithms. Accordingly, the contributions of this chapter are four-fold:

- SCTS is the first arbitrarily oriented subspace clustering algorithm for unbounded time series streams.
- SCTS identifies multi-cluster subspaces in the time-series data by setting a boundary for

each cluster using hyperellipsoidal cluster prototypes.

- SCTS identifies overlapping clusters in dependent subspaces by defining subspaces in local temporal neighbourhoods in the stream on a moving window.
- SCTS proposes a technique for detecting changes in subspaces in unbounded streaming time-series.

The outcomes of this chapter is presented in paper P3 in the Preface.

1.6 Chapter 6 - Incremental Monitoring and Control of Online Clustering Algorithms

Most state-of-the-art online clustering algorithms require user-specified input parameters related to the shape, radius, or density of clusters. These parameters set thresholds that are used to identify and adapt to changes in the data stream. Setting threshold parameters usually requires expert knowledge, or many passes over the data, which is not possible in streaming environments. Since no online clustering algorithm seems to address this issue, measuring their performance in various circumstances is important. Clustering performance is often evaluated with *cluster validity indices* (CVI). A CVI is a real-valued function that determines how well a clustering algorithm has partitioned data [10]. While *external* CVIs use some prior knowledge (e.g., groundtruth labels), *internal* CVIs use information intrinsic to partitions of the data (e.g., the cohesion of data in clusters or the separation of clusters from each other). A recent study [102] provided methods for incremental calculation of two internal CVIs. They showed that changes in the data stream were signaled by certain patterns in these CVI values.

The main contribution of this paper is circumventing such threshold parameters by calculating and utilizing CVIs in real-time. The algorithm we propose is not an online clustering algorithm; rather it is a control structure that is mountable on an underlying online clustering algorithm that guides it through changes in the stream. This control structure utilizes an incremental form of the well-known Xie-Beni cluster validity index [147] values in real-time while incorporating a forgetting factor, to signal changes in the underlying structures in the stream. Our experiments on synthetic and real-world datasets show that the proposed control structure improves the performance of two online clustering algorithms.

Accordingly, the contributions of this chapter are two-fold:

- We propose the first control structure to identify changes over a data stream using incremental Xie-Beni index values
- The proposed control structure can be mounted on other online clustering algorithms, making them capable of adapting to the changes over time in the data stream

The outcomes of this chapter is presented in paper P4 in the Preface.

1.7 Chapter 7 - Conclusions and Future Work

This chapter summarizes the main contributions of this thesis, reviews the techniques and summarizes the results obtained. It concludes with the impact these contributions had on real-life datasets, and proposes several directions for future research.

Chapter 2

Background and Survey of Data Stream Clustering and Data Stream Anomaly Detection

Continuous and large-scale data collection has enabled companies to better understand their business and gain a competitive edge over rival organizations. Since the majority of this data is unlabelled, unsupervised machine learning techniques such as clustering and unsupervised anomaly detection have become popular in the past decades. These tasks become more challenging when the assumptions made by traditional techniques are violated by the emergence of data streaming environments. Data streams have a wide range of applications in fields such as network intrusion detection, fraud detection, wireless sensor networks, and the Internet of Things. This chapter provides a detailed review of data stream clustering and anomaly detection techniques in the literature. First, the data stream model is introduced. In Section 2.2 clustering is defined and techniques in data stream clustering techniques are surveyed. In Section 2.3 anomalies are defined and data stream anomaly detection techniques are discussed. We conclude the chapter by highlighting the open questions that we address in this thesis.

2.1 Data Stream Model

Data streams were introduced in late twentieth century with the advancements in hardware and software technologies that enabled rapid and large-scale data acquisition [155]. Since then, there have been thousands of articles and hundreds of book on this subject [95, 127, 104, 110, 113, 50, 51, 19]. The *data stream* model attracted attention because several assumptions made by traditional *static* data models was violated in many application domains. Static data models assumed that (i) *data is finite*, (ii) *memory is unlimited* and (iii) *data can be accessed randomly*. The benefits of these assumptions are well-known; algorithms could read every data point several times,

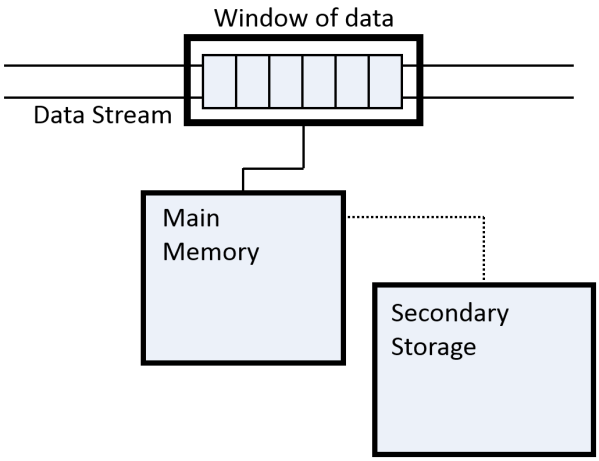


Figure 2.1: The data stream model. The limited memory creates a partial view of the data with a window that slides over the stream

processing time was not strictly bounded, and data points could be accessed in any order, e.g., finding the two closest data points in the whole dataset.

The amount of data generated in many applications inhibited using traditional machine learning techniques. Take a financial organization where millions of transactions are performed on a daily basis as an example. In this example, the data is too big to be loaded into main memory, and traditional data processing techniques that have a high complexity cannot be applied on the data. Since memory is limited, certain tasks, such as calculating a dissimilarity matrix for the whole dataset, are not feasible. This raises challenges for researchers when applying traditional machine learning tasks such as clustering and classification in these scenarios.

The data stream model addresses these challenges. In this model (i) *data can be infinite*, (i.e., unbounded), (ii) *memory is limited*, and (iii) *data becomes available sequentially* (i.e., there is no control on the order in which data becomes accessible) [30]. Algorithms emerged rapidly for this kind of environment because of a high demand in industry [1, 12, 26, 31, 81, 101, 136]. A schematic diagram of a typical data stream processing model is presented in Figure 2.1. In this figure, data arrives as an unbounded stream of data points that become available sequentially, where limited memory allows only a partial view of the data within a window that slides over the stream. The differences between the static data model and the data stream model are presented in Table 2.1.

Since the data window is an integral part of the data stream model, there have been several

Table 2.1: The characteristics of traditional data model and the data stream model

	Traditional Data Model	**Data Stream Model**
Memory	Unlimited	Limited
Data Access	Random Access	Sequential
Data Size	Bounded	Unbounded
Processing Time	Unrestricted	Restricted
Number of Passes	Several Passes	One Pass

windowing techniques proposed: landmark, sliding and damped.

1. **Landmark windows**: Landmark windowing partitions the stream into disjoint portions (chunks), where data processing is performed on each chunk. After processing each chunk, a data summary of the current chunk is created and merged with the data summary in past chunks. This windowing system leverages the solid background of static data processing in the literature (which is performed on each chunk). However, it suffers from the high complexity of static data processing techniques that require processing each data point more than once [101].

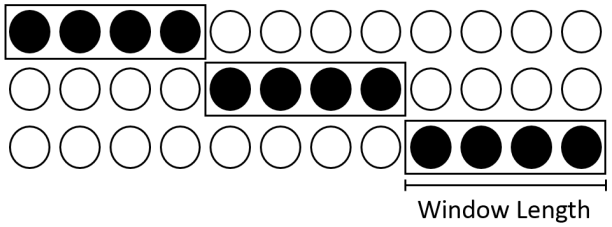


Figure 2.2: The landmark windowing system. In this windowing system, data points are processed in chunks.

2. **Sliding windows**: Sliding windows store the most recent data points in the stream for a specific length. When a new data point becomes available, the oldest data point in the window is discarded, and the newly arrived data point is added to the window. This system requires incremental data processing techniques to update the model with each data point, however proposing such algorithms is not trivial [127].

3. **Damped windows**: Damped windowing generally assigns an exponential forgetting factor to the data in the stream, where the most recent data points have higher weights. As more

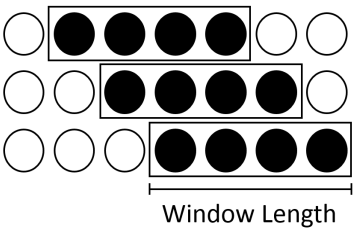


Figure 2.3: The sliding windowing system. In this windowing system, when a data point becomes available, the last data point in the window is discarded

data points arrive, the weights of the older points decrease over time. This windowing system is similar to the sliding window, where algorithms forget the past data points by using a weighting coefficient, rather than having a window length.

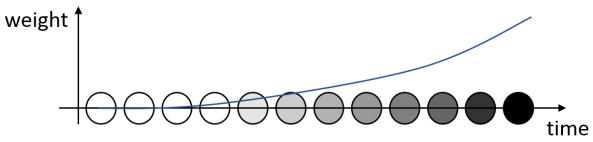


Figure 2.4: The damped windowing system. In this windowing system, the most recent data points in the stream have more weight compared to the older data points

The data stream model addresses several challenges for large-scale data. Windowing systems provide a partial view of the whole data stream that can be used for performing various machine learning tasks. In the next section, we describe the clustering problem and the major data stream clustering approaches in the literature.

2.2 Data Stream Clustering

Data clustering is the cornerstone of exploratory data analysis in machine learning. It involves partitioning data points into clusters where data points in the same clusters are more similar to each other than the data points in other clusters. Clustering is an unsupervised technique that is generally performed when there is no (or little) knowledge about the data. It aims at simplifying data analysis by finding and studying representatives of the clusters, rather than each and every data point. An example of data clustering is illustrated in Figure 2.5. In this figure, three clusters are identified. The data points in each cluster can be represented by a single point, e.g., the mean

of the data points in the cluster, which simplifies studying and explaining the dataset.

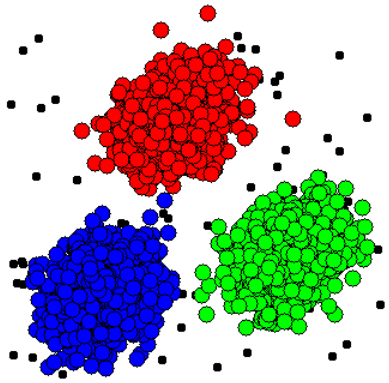


Figure 2.5: An example of data clustering where three clusters are identified

We present a formal definition for the clustering problem based on [84]. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of d-dimensional feature vectors (data points) and let $V = \{v_1, \ldots, v_k\} \subset \mathbb{R}^{kd}$ be a set of k prototypes (cluster centres) in $\mathbb{R}^d$. The aim of clustering is finding the membership of each data point to each cluster centre. The sets of fuzzy and crisp *k-partitions* of X are the sets of matrices

$$M_{fkn} = \Big\{ U \in \mathbb{R}^{kn} : u_{ij} \in [0,1] \; \forall i, j; \; \sum_{i=1}^{k} u_{ij} = 1 \; \forall j; \sum_{j=1}^{n} u_{ij} > 0 \; \forall i \Big\} \qquad (\text{f} \Rightarrow \text{fuzzy}) \qquad (2.1a)$$

$$M_{ckn} = \{ U \in M_{fkn} : u_{ij} \in \{0,1\} \; \forall i, j \} \qquad (\text{c} \Rightarrow \text{crisp}) \qquad (2.1b)$$

M_{fkn} is the fuzzy membership matrix with k rows and n columns, where each $u_{ij} \in [0,1]$ determines the degree of membership of x_j to the i^{th} cluster centre. The condition $\sum_{i=1}^{k} u_{ij} = 1 \; \forall j$ constrains each data point to be fully absorbed by all cluster centres, and the second condition $\sum_{j=1}^{n} u_{ij} > 0$ constrains each cluster centre to have at least one member. The crisp membership matrix (M_{ckn}) is a subset of M_{fkn} where the degrees of membership are binary, i.e., from the set $\{0,1\}$. Data clustering aims to find the number of underlying clusters and determine the values of the matrices M_{fkn} for fuzzy memberships, and M_{ckn} for crisp memberships.

There are many approaches for clustering since there is generally no an optimal clustering algorithm [127]. Before surveying these techniques, we briefly discuss a special case that requires separate treatment. When d (the number of dimensions) is high, several problems arise that prevent using ordinary distance measurement techniques in the full-space. Therefore, using standard

clustering techniques for high-dimensional data is impractical, and other techniques are developed that are known as *subspace clustering* [139]. Subspace clustering aims at identifying a set of features that can describe the underlying clusters accurately. The number of these features is often much fewer than d which makes using standard measuring techniques in the subspace feasible. A detailed description of these methods is provided in Chapter 6.

Aggrawal et al. [2] divide the majority of data stream clustering algorithms into four categories: partitioning methods, density-based methods, density grid-based methods and model-based methods. In the next few sections, we briefly discuss the major data stream clustering algorithms in each category, as depicted in Figure 2.6.

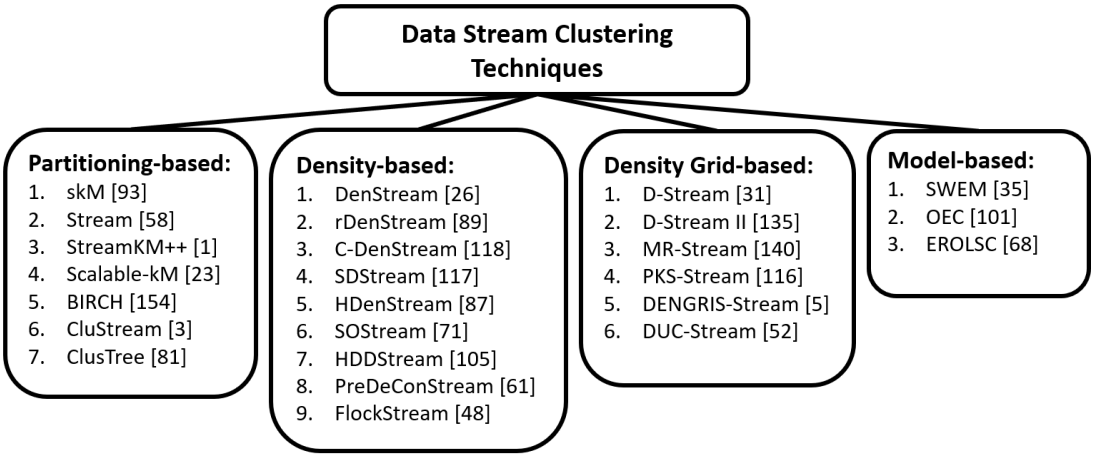


Figure 2.6: Overview of the data stream clustering methods that are discussed in this survey

2.2.1 Partitioning Methods

Partitioning based methods aim at minimizing an objective function by spreading k cluster centres in the feature space. In these techniques the membership of each data point to the clusters is determined by calculating their distance to each cluster. The earliest methods in this field are k-means (kM) [90], k-medians [13, 11] and k-medoids [109]. We first present the mathematical background of the kM algorithm and then discuss its extensions to the streaming environment.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of d-dimensional feature vectors and let $V = \{v_1, \ldots, v_k\} \subset \mathbb{R}^{kd}$ be a set of k prototypes (cluster centers) in $\mathbb{R}^d$. The *within groups sum of squared errors* (WGSS)

objective function is:

$$J(U,V;X) = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} u_{ij} \left\| x_j - v_i \right\|_A^2 \right) \tag{2.2}$$

where A induces the inner product norm $\left\| x_j - v_i \right\|_A^2 = (x_j - v_i)^T A (x_j - v_i)$. Function J lies at the heart of kM, which is the optimization problem (aka the *minimum variance partitioning problem*)

$$\underset{U \in M_{ckn}, V \in \mathbb{R}^{kd}}{\text{minimize}} \{J(U,V;X)\} \tag{2.3}$$

The necessary conditions for solutions to 6.3 are well known:

$$u_{ij} = \begin{cases} 1; \left\| x_j - v_i \right\|_A^2 \leq \left\| x_j - v_l \right\|_A^2, i \neq l & \text{for } 1 \leq i \leq k \\ 0; \text{otherwise} & \text{for } 1 \leq j \leq n \end{cases} \tag{2.4}$$

$$v_i = \Big(\sum_{x_j \in X} x_j \Big) \Big/ n_i = \sum_{j=1}^{n} u_{ij} x_j \Big/ \sum_{j=1}^{n} u_{ij}; \quad \text{for } 1 \leq i \leq k \tag{2.5}$$

There are many models related to the optimization problem in Equation 6.3 but the most commonly documented and widely used of these are the kM model, and the sequential kM model (skM) [93]. The main difference between kM and skM is that the sequential version considers local structure in the data one point at a time, whereas kM tries to capture the global relationship between all the vectors in X during alternating estimation to find a pair (U,V) that might solve Equation 6.3.

skM can easily be applied on streaming data but its performance deteriorates in non-stationary environments. This is because of the fact that it cannot adapt well to the emergence and evolution of clusters in the stream. There are many algorithms based on kM and skM for very large datasets, such as [1, 28, 72, 86, 145] just to name a few.

skM can be extended to a competitive neural network where the output layer neurons are the k centres of the clusters and by the arrival of a new data point, the nearest neuron (winner) is identified and gets updated. One network structure to choose the winner is through lateral inhibition [45] where each neuron reinforces itself and undermines others. Another competitive network is Hebbian learning [83] where correlated neurons are updated to reinforce the correlation. In these networks only the winner neuron gets updated which usually results in having dead neurons in the

network and one way to avoid it is to update a group of neurons in a neighborhood of the winner neuron, i.e., self-organizing maps (SOM) [77].

The mentioned algorithms require the number of clusters in advance. A neural network that automatically identifies the number of clusters is presented in [29] named *adaptive resonance theory* (ART-2). In this algorithm, clusters are defined with hyper-spheres where the radius is calculated from an input parameter named *vigilance*. The algorithm starts with a single cluster (neuron) and if a new observation does not belong to any clusters, a new cluster is defined. In this network, setting a suitable value for the vigilance parameter is difficult because it defines the radius of the clusters and if it is not set correctly, the algorithm will have a poor performance.

Guha et al. [58] proposed Stream for data stream clustering. It assumes data becomes available in chunks (windows) of size wl which is calculated based on the memory budget. Having each chunk available in memory, it performs k-medians on the first chunk. After the first step, the k prototypes of the first window are stored along with their weight, which is the number of data points assigned to them. Then the second chunk is processed, which creates k prototypes. At the end of the second step, there are $2 \times k$ prototypes in memory. This process continues r times with $r \times k$ being the memory budget. At this point, these $r \times k$ prototypes are clustered into k prototypes, where the weights assigned to each prototype are considered in the clustering process. The final clustering can be achieved at any time by clustering the current set of prototypes. Later [106] extended the Stream algorithm by proving that the final quality of the output cannot be arbitrarily worse than the particular subroutine that is used at the intermediate stage for k-medians clustering.

StreamKM++ [1] has a different strategy for storing the summaries of data in the stream. It stores the data summaries in a data structure called a *coreset tree*. A coreset tree is a binary tree in which each node is the union of all elements that descend from it. StreamKM++ has two steps, one for merge and the other for reduce. The reduce step is performed by the coreset tree where $2 \times g$ data points are reduced to g data points (note that g is defined in the next step). The merge step is performed by a data structure named the bucket set. The algorithm has L buckets of size g. When a new data point arrives, it is stored in the first bucket. If it is full, all of its data is moved to the second bucket. If the second bucket is also full, a reduce step is performed to reduce $2 \times g$ elements to g elements and this process continues indefinitely. A shortcoming of StreamKM++ is its poor performance when memory is constrained [126].

Another series of studies worked on proposing approximate algorithms for the kM and k-median algorithms. The work in [21] employs "coresets", but adds a guaranteed approximation of the optimal solution by solving the optimization problem on the dataset. They initially "guess" the cost of the optimum solution, and then run an "online facility location" [96] algorithm on the data. It processes data to the point that either the current cost exceeds a constant times the initial guessed value, or the number of facilities exceeds a parameter κ. At this point, they declare an end of a "phase" where they increase the guessed optimal solution, reduce the κ facilities into k and continue for the rest of the stream. The final clustering is achieved by running kM on the current set of facilities.

The work in [126] extends [21] to achieve greater efficiency and a better approximation guarantee. They remove some unnecessary checks, improve the cost of the facility location problem, simplify the transition between phases, and bound the number of phases to $\log OPT$ where OPT is the optimum solution of kM. A shortcoming of this method is that it scales poorly with k, the number of cluster centres.

Broder et al. [23] address the scalability of kM over millions of data points and thousands of cluster prototypes. They speed-up the nearest neighbour search by indexing the data points with a data structure called an "inverted file structure". The natural process is to index the cluster centres, so that for each arriving data point the closest cluster centre can be identified easily. However, they proposed the converse approach where the data points are indexed and at each iteration each prototype is queried against all the indexed data to find neighbouring data points. Since they do not need to rebuild the index for each input data point, they show an improvement in efficiency of up to one to two orders of magnitude.

There is another group of partitioning methods for indefinite data streams that first create a summary of the observed data in secondary storage, then perform a batch clusteing algorithm (such as kM) on the data summary. This method was first used by Zhang et al. [155] in the algorithm named *BIRCH* (Balanced Iterative Reducing and Clustering using Hierarchies). They used a feature vector for summarizing the statistics of large amounts of data in the stream and named it a *Cluster Feature* (CF) vector. A CF is denoted by $< n, LS, SS >$ and where n is the number of data points, LS is the linear sum of the data points, and SS is the sum of the squared data points. These features allow efficient calculation of the cluster mean, cluster radius and cluster

diameter:

$$mean = \frac{LS}{n} \tag{2.6}$$

$$radius: \sqrt{\left(\frac{SS}{n} - \left(\frac{LS}{n}\right)^2\right)} \tag{2.7}$$

$$diameter: \sqrt{\left(\frac{2n \times SS - 2 \times LS^2}{n(n-1)}\right)} \tag{2.8}$$

Moreover, the CF vector has useful properties for efficiently updating or merging several CFs by simple mathematical operators. Another data structure used in this work is a height-balanced tree (CF tree) whose leaf nodes contain CFs. The CFs in leaf nodes are aggregated in their parents, so every node in this tree provides a partial view over a part of the observed data in the stream. When a new data points arrives, it descends this tree to the leaves by choosing the child whose CF mean is closest to the data point. Each leaf node has a radius that determines if the newly arrived data point can be absorbed by this CF or a new CF must be defined. If there is no space in the leaf for a new CF, the leaf splits and the farthest pairs of CFs are put in the new leaf as the seed.

The CF vector was extended by Aggarwal et al. [3] who introduced *CluStream* to capture the temporal characteristics of data points in a stream. The new data structure was called a "microcluster" denoted by $< n, LS, SS, LST, SST >$ where LST is the sum of the timestamps of the data points in the CF, and SST is the sum of the squares of the timestamps of the data points in the CF. CluStream comprises an online and an offline phase. In the online phase, a summary of the observed data is stored in secondary storage as micro-clusters, and in the offline phase a clustering algorithm such as kM is performed on the data summary. Having stored the temporal characteristics of data points in the stream enabled CluStream to perform clustering on any given horizon.

Kranen et al. [81] later proposed *ClusTree* that uses CF vectors, but each CF is associated with a weight that indicates an importance based on temporality. The weight is calculated such that CFs with no recent objects lose their importance. It also provides solutions for anytime clustering, i.e., interrupting the insertion process of newly arrived data points. These interrupted data points are stored in a sub-tree buffer, and when this sub-tree is accessed, they descend through to the leaves

as “hitchhikers”. An important characteristic of ClusTree is that it can be adapted to fast and slow streams.

2.2.2 Density-Based Methods

Density-based algorithms identify changes in a stream by monitoring the density of data in the feature space. These methods can identify arbitrarily shaped clusters and are capable of detecting anomalies. However, they require parameters that are related to the density of the neighbourhoods of data points, which can be difficult to set.

Cao et al. [26] proposed *DenStream*, which extended the micro-cluster structure introduced by CluStream for density-based clustering. The density of micro-clusters is calculated with an input parameter for a radius threshold. In this approach, three variations of micro-clusters were proposed: *core-micro-clusters* (c-micro-cluster), *potential core-micro-cluster* (p-micro-cluster), and *outlier micro-cluster* (o-micro-cluster).

A c-micro-cluster at time t for a group of data points $X = (x_{t_1}, x_{t_2}, x_{t_3}, ..., x_{t_n})$ is defined as CMC(w, c, r). In this definition, w is the weight of the CMC and is defined as

$$w = \Sigma_{j=1}^{n} f(t - t_j) \tag{2.9}$$

where

$$f(t) = 2^{-\lambda \times t}$$

The second element c is the centroid of the CMC and is defined as:

$$c = \frac{\Sigma_{j=1}^{n} f(t - t_j) \times x_{t_j}}{w} \tag{2.10}$$

Finally, the third element (r) is the radius of the CMC and is defined as:

$$r = \frac{\Sigma_{j=1}^{n} f(t - T_{ij}) dist(x_j, c)}{w} \tag{2.11}$$

where $dist(x_j, c)$ denotes the Euclidean distance between the points x_j and c. A c-micro-cluster has the property that its weight is greater than an input parameter ϕ, and its radius is less than

the radius threshold ε. This indicates that the micro-cluster has many recent data points with a small radius. A p-micro-cluster is a micro-cluster such that $w \geq \beta \times \phi$ where $0 < \beta < 1$ (a value close to 1). This parameter indicates the fact that c-micro-clusters turn into p-micro-clusters if they do not receive data points for a period of time. An o-micro-cluster is a p-micro-cluster such that $w < \beta \times \phi$ which means it has not received data points for an extended period of time.

These structures are used to identify dense regions as clusters and detect anomalies in the stream. DenStream performs clustering in two phases, similar to [3, 81, 155]. In the first phase (micro-cluster maintenance) it captures the dense and anomaly regions in the stream. In the second phase (generating clusters) it performs a batch clustering algorithm such as DBScan [44] on the data summary. One shortcoming of this algorithm is its unintuitive input parameters.

Liu et al. [89] proposed *rDenStream* as an extension of DenStream for achieving better accuracy. They add another phase to the two-phased framework of DenStream and named it as *"retrospect"*. In this phase, the discarded data points are put in a anomaly buffer and have another chance to be added to micro-clusters that may form later in the stream. In their performance evaluation against DenStream, they reported higher clustering accuracy and purity in their experiments.

Another extension of DenStream is *C-DenStream* proposed by Ruiz et al. [119] which adapts C-DBSCAN [120] for streaming environments. C-DBSCAN is a density based clustering algorithm with constraints. These constraints can be Must-Link constraints, which indicate that two data points belong to the same cluster, and Cannot-Link constraints, which indicate that two data points do not belong to the same cluster. In C-DenStream, these constraints are applied on micro-clusters and the final clustering is generated by applying C-DBSCAN on the data summary. C-DenStream can be helpful in cases where some background information is available and can prevent generation of unacceptable clusterings. However, it needs expert knowledge that can be potentially expensive to acquire.

Ren and Ma [117] developed *SDStream*, which is another extension of DenStream that uses a sliding window over the stream. It stores micro-clusters in "Exponential Histograms" (EH) introduced in [37], which are designed to maintain approximate statistics with low memory complexity. In its offline phase, it applies DBSCAN on the data summary for generating clusters.

HDenStream inherits potential micro-clusters and outlier micro-clusters from DenStream and employs the distance measure proposed for static categorical data in [150]. It extends the micro-

clusters by adding another entry, which is a two-dimensional array that keeps the frequency of categorical data. It calculates the distance between two micro-clusters by measuring their distance for continuous and categorical attributes separately, and combines them as a distance between these two micro-clusters. HDenStream has the capability of handling data streams with categorical attributes, but it does not provide efficient methods for storing them for indefinite streams.

SOStream (Self Organizing Density-Based Clustering Over Data Streams) was proposed by Isaksson et al. [71]. Rather than manually setting a static threshold for identifying dense regions in the feature space, it learns a dynamic threshold value for each cluster with a minimum number of points. It employs competitive learning, which was introduced in [78] where a winner microcluster influences its overlapping micro-clusters. When a new data point does not belong to any micro-clusters, a new micro-cluster is added for it. Although SOStream learns the radius threshold for each micro-cluster automatically, its high computational complexity inhibits using it in real life applications.

HDDStream (High-Dimensional Data Stream) [105] is a density-based clustering algorithm that employs the micro-cluster structures of DenStream for high-dimensional data. It adds a *prefer vector* to micro-clusters, which indicates the variance of each dimension. Generally, dimensions of lower variance (that are dense) are preferred to better explain the data, and the micro-cluster with the preferred dimensions is a projected micro-cluster. It generalizes the outlier and potential micro-clusters to projected outlier and projected potential micro-clusters respectively. In its offline phase it applies the PreDeCon [20] clustering algorithm on the data summary, which is a subspace clustering algorithm for static data.

PreDeConStream [61] is similar to HDDStream, but it increases its accuracy in the offline phase by maintaining another list as *affected micro-clusters*. Although PreDeConStream has greater accuracy, maintaining the affected micro-clusters may be time consuming. FlockStream [48] is another density-based stream clustering algorithm that has a bio-inspired model. It considers each data point in the stream as an agent, where each agent moves in its predefined neighbourhood for a fixed amount of time. Agents that come across each other join and make clusters. The fact that every data point is only compared with the data points in its neighbourhood, makes FlockStream more efficient compared to DenStream. However, it does not provide any explicit method for identifying anomalies.

2.2.3 Density Grid-Based Methods

Density grid-based methods are a combination of grid-based methods and density-based methods. Grid-based methods partition the feature space into small cells known as grids, and explore the distribution of data points in each grid for creating clusters. They were introduced in algorithms such as [142], which stores and processes the statistical information of data points in grid cells, WaveCluster [125], which clusters data points using wavelet transform methods, and CLIQUE [4], which merges grid-based and density-based methods. In density-grid based algorithms, data points in the stream are mapped into grids, then the densities of the grids are used for clustering.

This subclass of stream clustering algorithms are capable of detecting arbitrary shaped clusterings and identifying anomalies. However, their time complexity often increases exponentially with the number of dimensions. Gao et al. [52] proposed DUCstream (Dense Units Clustering for data Streams), which is a single pass algorithm that uses dense units for stream clustering. It calculates the density of each grid cell and if it is greater than a threshold, considers them as dense units. It only retains the grid cells that most probably will be dense units for efficiency and represents the grids as a graph. In this graph, vertices are the dense units and edges show their relation. If a new dense unit is identified, it is either absorbed by an existing cluster, otherwise a new cluster is defined for it. DUCstream employs the *density coefficient* concept, which is a decay factor for the density of grids that is decreased over time. This coefficient has a higher value for grids with more recent data points and a lower value for grids with no recent data points.

Another example is D-Stream [31], a two phased data stream clustering algorithm where in the first phase each grid cell is used to store synopsis information of the data points falling into it in secondary storage. Then, in the second phase the clustering is performed in an offline manner. An important part of this algorithm is that anomalies are detected as sporadic grids. However, it requires several input parameters for doing so, which can be difficult to set in many application domains. Later, this algorithm was extended by Tu and Chen [136], where the clustering process is similar to D-Stream, but it also takes grid attraction into account. Grid attraction determines the proximity of data points in two neighbouring grids, and two grids are merged only if their grid attraction is higher than a predefined threshold. Although this algorithm is more accurate than D-Stream, it requires several unintuitive parameters to be set by the user.

MR-Stream was proposed by Wan et al. [141], which uses a tree structure to store and main-

tain the data summary in constant time. It also enables users to discover clusters at different resolutions. However, its performance deteriorates as the number of dimensions increases. Ren et al. [116] proposed *PKS-Stream*, which addresses high-dimensional data stream clustering. Later, *DCUStream* was proposed by Yang et al. [152] for clustering of uncertain data streams. Amini et al. [5] proposed *DENGRIS-Stream* that uses a sliding window to capture the distribution of data points in the stream. Bhatnagar et al. [17] proposed *ExCC* (Exclusive and Complete Clustering) for heterogeneous data streams with mixed (numeric and categorical) data streams.

2.2.4 Model-Based Methods

Model-based methods generally use a sliding window over the stream and fit a model to the patterns observed in the data stream. SWEM was proposed by Dang et al. [35], which represents clusters with micro-components comprising three entities: a weight, a mean, and a covariance matrix. In the first window, the *Expectation Maximisation* (EM) algorithm is applied on the window for obtaining the learned parameters. In subsequent windows, it uses the previous window parameter as the initial values for the mixture models' parameters. If the difference of the two sets of parameters in neighbouring windows is high, SWEM redistributes the micro-components in the whole feature space by splitting micro-components with large variance and merging close micro-components. It also applies a forgetting factor as a weight for micro-components, so past data points have less effect on the current state of clustering. Finally, it employs *Mahalanobis* distance as a similarity measure for micro-components. Mahalanobis distance is defined as:

$$(x-\mu)^T\Sigma^{-1}(x-\mu) \leq z^2 \tag{2.12}$$

In this formula, x is a given data point, μ is the center of the micro-component (the mean of the Gaussian distribution), Σ^{-1} is the inverse of the covariance matrix and z^2 is a constant that marks a specific level set of the distribution. When z^2 is chosen from the cumulative inverted chi-squared distribution with d degrees of freedom for probability value $P((\chi^2_d)^{-1}_P)$, the probability of an observation falling outside the boundary will be $1-P$.

Moshtaghi et al. [101] proposed the OEC (Online Elliptical Clustering) algorithm for online clustering of multivariate time-series. This paper presents an incremental clustering algorithm

where every incoming data point was processed and discarded before the arrival of the next data point. It comprises a set of hyper-ellipsoidal clusterings and a state-tracker as a change-point detection technique that captures statistics of the most recent data with a forgetting factor. It provided incremental update formulas for updating the mean and the covariance matrix of hyperellipsoidal clusterings with and without the forgetting factor and used Mahalanobis distance to define the boundary of its clusters.

EROLSC (Extended Robust Online Streaming Clustering) was proposed by Ibrahim et al. [68], which is based on integrating Gaussian Mixture Models (GMM) with possibilistic fuzzy clustering. They apply Possibilistic C-Means (PCM) [82] on the initial window, which is mainly for identifying anomalies. Then an *Automatic Merging Possibilistic Clustering Method* (AMPCM) is applied to the clean data, where the final clustering result is used to initialize a GMM. The GMM is updated by every data point that arrived using the formulas provided in [101].

2.2.5 Summary of Data Stream Clustering Algorithms

The main characteristics that differentiate data stream clustering algorithms are the windowing model that they employ, the data structure they use to store the synopsis information of data, the geometrical representation of clusters, and their ability to detect anomalies in data streams. A summary of the methods discussed in the previous section is presented in Table 2.2.

Traditional data stream clustering algorithms [106, 155] employed the landmark windowing system (which divides the stream into chunks), performed a batch clustering algorithm on each chunk, and then merged the information acquired by clustering each chunk. These methods leverage the vast literature on batch clustering algorithms, but the high time complexity of those methods and the fact that those algorithms are executed on each chunk, results in high time complexities for these stream clustering methods. Newer methods [45, 50, 63] use online or real-time learning techniques and process data points one at a time. They either use the damped window or the sliding window model, process incoming data points and discard. Since the incremental update does not relate to the size of the dataset, their time complexity is linear w.r.t the stream size, which makes them more suitable for streaming environments.

One challenge for incremental learning techniques is identifying and adapting to changes in non-stationary environments [3]. When an input data point does not belong to the existing model of

clusters, it is either an anomaly, or a cluster that is emerging. Since identifying between these two cases is challenging, many stream clustering algorithms perform anomaly detection as a separate process (when demanded by the user) in an offline manner. However, online clustering algorithms that automatically identify anomalies in real-time are needed.

Another challenge for data stream clustering is when the number of features in a data stream is high [139]. Subspace clustering addresses such environments, but the literature of subspace clustering tends to focus on small datasets where all data can be stored in memory [43, 85, 88, 91, 132]. Finding subspaces in a streaming environment still remains a challenge, and the subspace clustering algorithms in the streaming environment alleviate this challenge by reducing the infinite search space of arbitrarily-oriented subspaces into a bounded number of axis-parallel (or projected) subspaces. Therefore, identifying arbitrarily-oriented subspaces in streaming environments still remain a challenging and open problem.

Finally, most state-of-the-art online clustering algorithms require user-specified input parameters related to the shape, radius, or density of clusters. These parameters set thresholds, which are used to identify and adapt to changes in the data stream. Setting threshold parameters usually requires expert knowledge, which can be expensive, or requires many passes over the data, which is not possible in streaming environments. Therefore, a control structure is needed that can be mounted on online clustering algorithms to guide them through the changes in a stream. We conclude that there are specific challenges in online clustering algorithms that are not addressed in the literature. Although there are algorithms that work well under certain conditions, new techniques should be developed that are more practical in real-life application domains.

Table 2.2: Data stream clustering related work summary- AD: Anomaly Detection, OAD: Online Anomaly Detection, PCS: Parameters related to Cluster Shape

Approach	Algorithm	Window Model	Data Summary	Processing	Cluster Shape	AD	OAD	PCS
Partitioning	Stream [58]	Landmark	Cluster centre	Single-pass	Spherical	✓	✗	✗
	StreamKM++ [1]	Incremental	Core-set tree	Merge-Reduce	Spherical	✗	✗	✗
	skM [93]	Incremental	Cluster centre	Single-pass	Spherical	✗	✗	✗
	Scalable kM [23]	Incremental	Cluster centre	Single-pass	Spherical	✓	✗	✗
	Scalable kM++ [12]	Incremental	Cluster centre	Several passes	Spherical	✗	✗	✗
	BIRCH [155]	Landmark	Cluster feature	Two-phased	Spherical	✓	✗	✗
	CluStream [3]	Damped	Micro-cluster	Two-phased	Spherical	✗	✗	✗
	ClusTree [81]	Damped	Micro-cluster	Two-phased	Spherical	✓	✗	✗
Density-based	DenStream [26]	Damped	Micro-cluster	Two-phased	Arbitrary	✓	✗	✗
	rDenStream [89]	Damped	Micro-cluster	Two-phased	Arbitrary	✓	✗	✗
	C-DenStream [119]	Sliding	Micro-cluster	Two-phased	Arbitrary	✓	✗	✗
	SDStream [117]	Damped	EHCF	Two-phased	Arbitrary	✓	✗	✗
	HDenStream [87]	Damped	Core-micro-cluster	Two-phased	Arbitrary	✓	✗	✗
	SOSTream [71]	Damped	Micro-cluster	Single-pass	Arbitrary	✓	✗	✗
	HDDStream [105]	Damped	Core-micro-cluster	Two-phased	Arbitrary	✓	✗	✗
	PreDeConStream [61]	Damped	Core-micro-cluster	Two-phased	Arbitrary	✓	✗	✗
	FlockStream [48]	Damped	Micro-cluster	Two-phased	Arbitrary	✗	✗	✗
Density Grid-based	D-Stream [31]	Damped	Grid-hashtable	Two-phased	Arbitrary	✓	✗	✗
	DUC-Stream [52]	Landmark	Grid-graph	Two-phased	Arbitrary	✓	✗	✗
	D-Stream II [136]	Damped	Grid	Two-phased	Arbitrary	✓	✗	✗
	MR-Stream [141]	Damped	Grid-tree	Two-phased	Arbitrary	✓	✗	✗
	PKStream [116]	Damped	Grid-PKS-tree	Two-phased	Arbitrary	✓	✗	✗
	DENGRIS-Stream [5]	Sliding	Grid	Single-pass	Arbitrary	✗	✗	✗
Model-based	SWEM [35]	Sliding	GMM	Single-pass	Elliptical	✓	✗	✗
	OEC [101]	Incremental	GMM	Single-pass	Elliptical	✓	✗	✗
	EROLSC [68]	Sliding	GMM	Single-pass	Elliptical	✓	✗	✗

2.3 Data Stream Anomaly Detection

Anomalies in data are referred to as data points or patterns that do not conform to the normal behaviour of a system [115]. Detecting anomalies is particularly important because they often indicate a problem, fault or a abnormal behaviour in the system. Detecting anomalies has a wide range of applications such as fraud detection in financial organizations [80], intrusion detection in computer networks [73], fault detection in industrial machinery [94], and military surveillance for enemy activities [75]. In such applications, anomalies often represent a failure, suspicious activity or an intrusion in the system that needs to be studied by an expert [30]. In this section, we briefly explain what anomalies are, different types of anomalies and their importance. We then present a survey of data stream anomaly detection techniques in the literature.

2.3.1 What Are Anomalies?

An anomaly is formally defined as "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" [62]. Anomalies are important in many applications. For instance, in network security, the network traffic of computers, routers, mobile phones, printers, etc., form various patterns in the data. Sometimes an intrusion in the network can be identified by monitoring the patterns in the network traffic as a burst of a specific kind of pattern. Monitoring this data can protect against intruders by taking security measures after detecting an anomaly. Another example is in financial organizations where millions of transactions are performed on a daily basis. Normal transactions often create various patterns in the data but fraudulent transactions generally deviate from this normal behaviour.

The literature on anomaly detection has mainly focused on three types of anomalies [30]:

1. **Point anomalies**: This type of anomaly is sometimes referred to as noise and is defined as the data points that reside far away from the majority of the data. This is the easiest type of anomaly where there are effective algorithms for identifying them.

2. **Contextual anomalies**: Data points that are considered anomalies because of the context (or neighbourhood) of those data points. This type of anomaly happens when a data point that is considered normal with respect to the whole dataset, is labelled as anomalous because of the patterns in its local temporal neighbourhood [30]. For instance, in Figure 2.7 the

temperature at time t_2 is considered anomalous because in June, a low temperature is not expected to be observed. An individual data point is considered an anomaly with respect to the context in a dataset. For example, the same temperature at time t_1 is not considered an anomaly, because low temperatures are expected at that time of year.

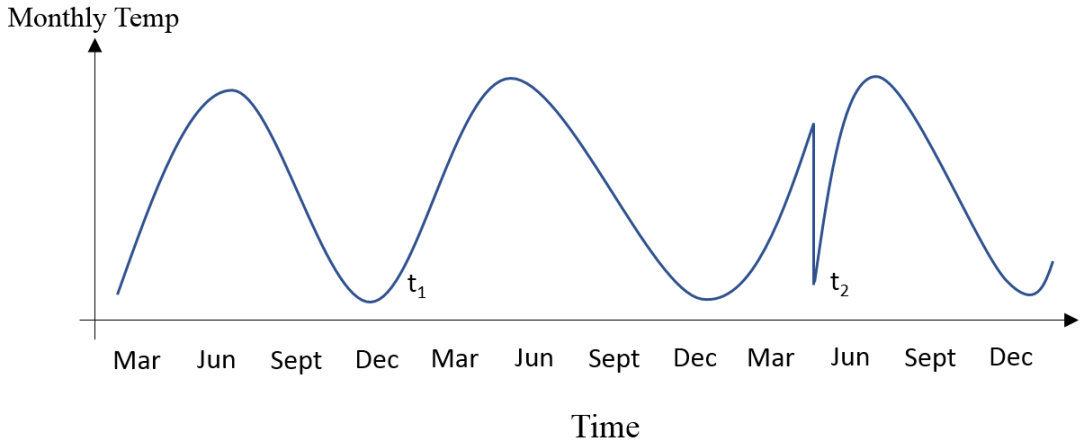


Figure 2.7: Example of contextual anomaly at time t_2

3. **Collective anomalies**: A collection of data points that are considered anomalies compared with the entire dataset. The data points in this collection may be normal by themselves, but when they are observed in a collection they are labelled as anomalous. For instance, Figure 2.8 [30] represents the Atrial Premature Contraction in a human electrocardiogram where the collection of data points from time 1000 to 1500 are considered anomalous.

Depending on the availability of labels for data, there are generally three training scenarios for anomaly detection. First, *supervised* techniques assume that a training dataset is available along with the labels that can be used for modelling the "normal" and "anomalous" data points. Although these methods can achieve high accuracy, they are limited to cases where labels of data points are available [55]. Second, *semi-supervised* techniques assume in the training dataset only the normal data points exist, which are used for building a model for the normal patterns. In this scenario, the data points in the test set that do not conform to the model of normal are labelled as anomalous. Finally, *unsupervised* techniques are used where there are no labels for the dataset and both normal and anomalous classes might exist in the data. This is often a necessity because many application domains do not provide labels for the data points and cannot provide a training dataset

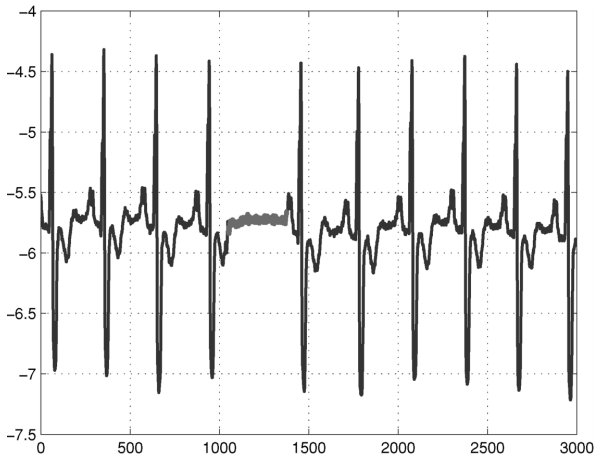


Figure 2.8: Collective anomaly

that only has normal data points in it [101, 122]. Since the focus of this thesis is on unsupervised anomaly detection, we mainly focus on this category.

The research on anomaly detection in data streams can be divided into two major categories. The first category is *proximity-based* methods, where observations that are isolated from the rest of the data are detected as anomalies. The second category is *model-based* algorithms, where a model of normal data is created and updated as the input data arrives. These methods are outlined in Figure 2.9 and are discussed in the next few sections.

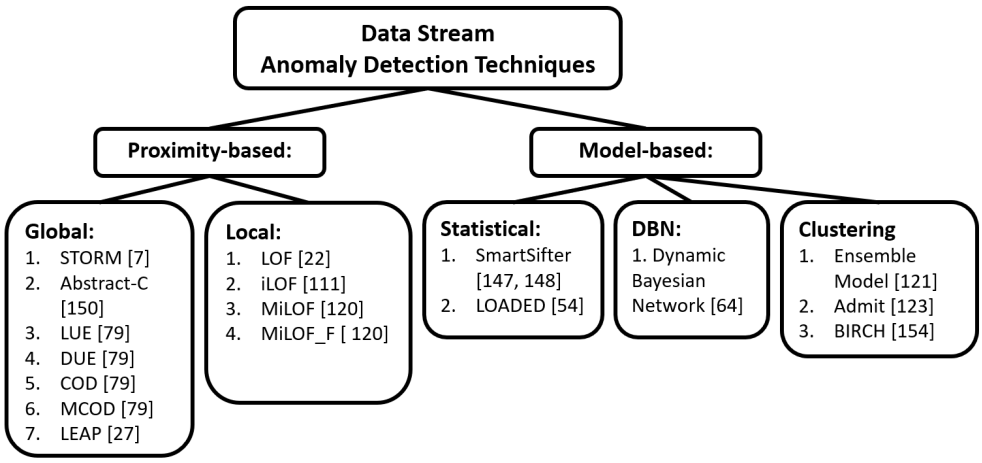


Figure 2.9: Overview of the data stream anomaly detection techniques discussed in this survey

2.3.2 Proximity-Based Anomaly Detection Methods

Proximity-based methods detect anomalies by calculating the distances of data points from each other, and labelling those that are relatively far from the majority of data points as anomalies. There are two main definitions of anomalies in this field: *global* and *local*. Global methods can only identify certain types of anomalies with similar densities over the whole stream, while local anomalies are identified by measuring the local density of a data point with respect to its neighbours. We elaborate more on the two following sections on global and local anomaly detection methods.

Global Anomalies

The most recent distance based algorithms for anomaly detection in data streams are Storm [7], Abstract-C [151], LUE [79], DUE [79], COD [79], MCOD [79], and Thresh_LEAP [27]. These algorithms use the anomaly definition introduced by Knorr in [76]. Based on this definition, a data point is called a DB(q, R) anomaly (Distance Based(q, R) anomaly) if there are less than q data points within radius R from it. Figure 2.10 represents a 1-dimensional data stream with two time windows. The first time window is t_3 to t_{18} and the second time window is t_7 to t_{22}. In the first window, if q is set to 3, o_9 with four neighbours (o_5, o_{10}, o_{14} and o_{15}) and o_{11} with four neighbours (o_3, o_4, o_6 and o_{13}) are considered to be normal. However, o_8 is an anomaly with only two neighbours (o_7 and o_{16}).

Angiulli et al. [7] proposed an exact algorithm, called STORM, for efficiently calculating distance-based anomalies with a data structure called an *Indexed Stream Buffer* (ISB). They also proposed an approximate algorithm, called approx-Storm, that requires much less memory compared to STORM. Approx-Storm is based on two observations: (i) only a fraction of normal observations is sufficient to be stored in ISB, (ii) instead of storing the list of k most recent preceding neighbours, it is sufficient to store only a fraction of preceding neighbours.

Yang et al. [151] proposed the Abstract-C algorithm, which stores abstracted neighbour relationships rather than all neighbour relationships. They also exploit the "predictability" of the expiration of data points in the window for predicting the pattern structures for the current window. One advantage of Abstract-C over STORM is that it does not find all neighbouring data

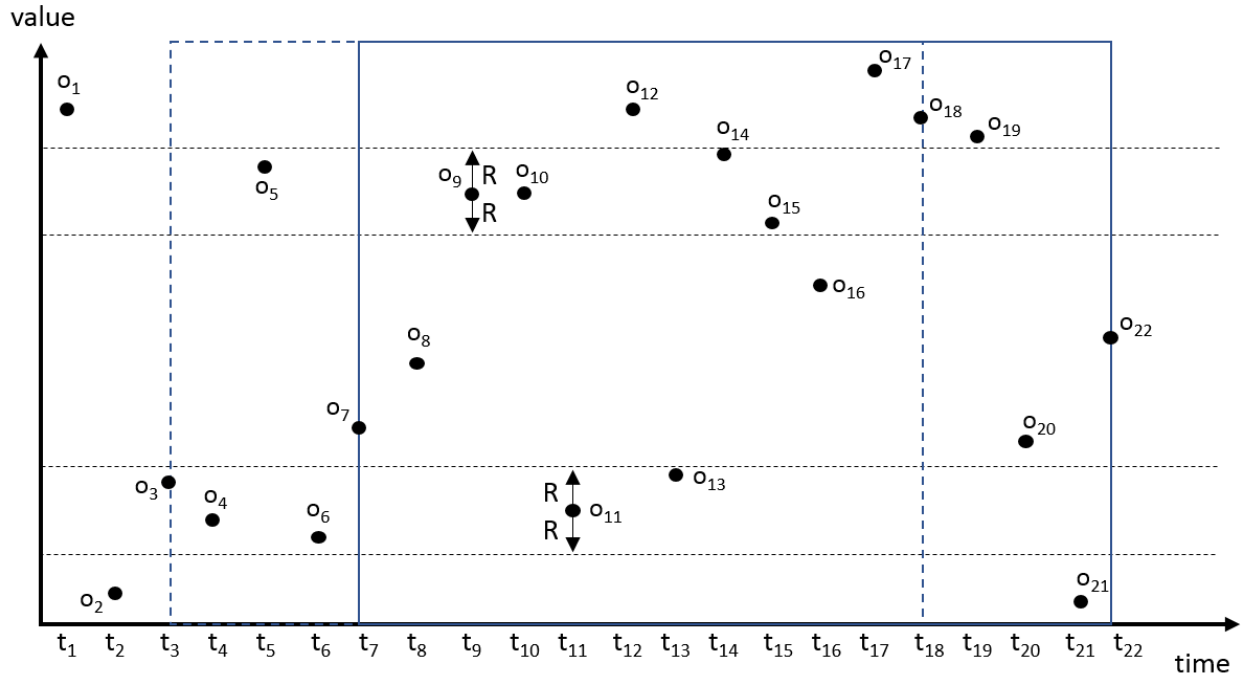


Figure 2.10: Example of the evolution of a 1-dimensional data stream

points for each data point in the window. However, its memory consumption heavily depends on the window length parameter.

In DUE [79] the temporal dependencies of data points are considered, where data points are updated only by their neighbours in adjacent windows. DUE also employs a priority queue called an *event queue* that stores data points that are likely to become anomalies in the future. Data points in event queue are sorted in the order of increasing expiration time. Finding all neighbours for each data point in the stream can be time consuming, therefore MCOD [79] stores neighbouring data points in micro-clusters with at least $q+1$ data points. The radius of micro-clusters is $R/2$ and based on the triangle inequality in metric space, so the distance between every pair of data points in a micro-cluster is less than R. This observation makes all data points in micro-clusters to be labelled as normal. As the window progresses, the data points that do not fall into any micro-clusters are stored in a list, and the data points that expire and have less than k neighbours are labelled as an anomaly.

Cao et al. [27] proposed the LEAP framework, which aims to optimize the three main distance based anomaly definitions in the literature. These three definitions are:

1. $O_{thresh}^{(q,R)}$: anomalies are the data points with fewer than q neighbours within radius R from them
2. $O_{qmax}^{(q,n)}$: anomalies are the n data points with the highest distance values to their q^{th} nearest neighbour
3. $O_{qavg}^{(q,n)}$: anomalies are the n data points with the highest average distance values to their q^{th} nearest neighbour

The LEAP framework optimizes these methods based on two observations: (i) *minimal probing* and (ii) *lifespan-aware prioritization*. The minimal probing principle is based on the fact that anomalies constitute a very small fraction of the data stream. In a data stream, all data points are either normal points or anomalies. By minimal probing, they look for the minimal evidence that shows a data point is normal. Therefore, rather than performing a complete neighbourhood search for each data point, as soon as sufficient evidence is found that a data point is normal, it is removed from the set of all anomaly candidates. This way, a large number of data points are removed, which provides a considerable speed up for identifying all of the three anomaly definitions. The lifespan-aware prioritization principle is based on the fact that later data points in the stream have a more decisive impact on the anomaly detection process rather than earlier data points.

One common shortcoming of the algorithms for identifying global anomalies is that they cannot identify more complex cases where clusters have different radii. More specifically, no parameter setting of global anomalies can identify anomalies from clusters with a very large radius as well as clusters with a very small radius in streaming environments.

Local Anomalies

To address the shortcomings of global anomaly detection techniques, Breuning et al. [22] proposed the Local Outlier Factor (LOF) measure. They considered detecting anomalies in a local manner, relative to their local neighbourhood as represented in Figure 2.11. In this figure, there are two clusters of different densities where o_1 and o_2 are anomalies. Global anomaly detection techniques cannot identify o_2 as anomalous. The reason is o_2 has the same distance to its nearest neighbours as the data points in cluster $C1$ have to their nearest neighbours.

To identify o_2 as an anomaly, they introduced the following concepts:

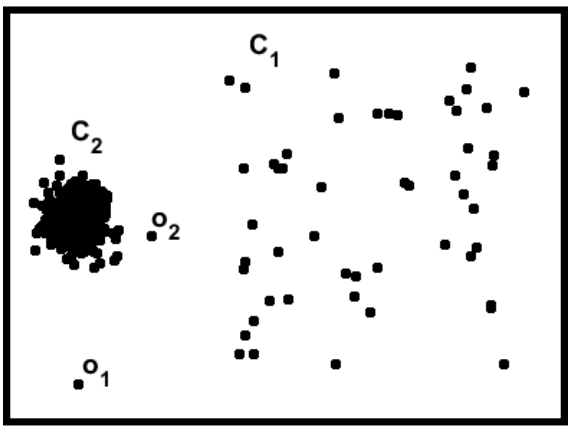


Figure 2.11: An example of local anomalies. o_2 is a local anomaly because the density of data points in its local neighbourhood is different from its own local density

1. *q-distance*(o): the distance between a data point o to its q^{th}nearest neighbour.

2. *Reachability distance* (reash-dist): The distance between a data point o_1 and another data point o_2 is

$$reach\text{-}dist_q(o_1,o_2) = max\{q\text{-}distance(o_1), d(o_2,o_1)\}$$

3. *Local reachability density* (lrd) of a data point o_1 is

$$lrd_q(o_1) = \left(\frac{1}{q}\Sigma_{o\in N_{(o_1,q)}} reach\text{-}dist(o_1,o)\right)^{-1} \tag{2.13}$$

   where $N_{(o_1,q)}$ is the set of q nearest neighbors of o_1.

4. *Local outlier factor* of a data point o_1 is defined as:

$$LOF(o_1) = \frac{1}{q}\Sigma_{o\in N_{(o1,q)}} \frac{lrd_q(o)}{lrd_q(o_1)} \tag{2.14}$$

The LOF of each data point is calculated based on the number of observations in its local neighbourhood. Although this algorithm was designed to detect anomalies in static data, it can be used to detect anomalies in streaming environments using a sliding window. Later, an incremental version of [22] was introduced in [111] named *Incremental Local Outlier Factor* (iLOF). iLOF calculates the exact score values as LOF does, but incrementally with intermediate LOF scores.

The main problem with iLOF is that it requires all the data points to be in memory, which makes it unsuitable for a true streaming environment.

Salehi et al. [121] proposed a *memory efficient incremental local outlier factor* (MiLOF) algorithm to address this problem, which produces similar LOF values to the outputs of iLOF within a fixed memory bound. Given the memory bound b, when the number of data points in memory reaches b it performs a k-means clustering on the first $b/2$ data points in memory. The second half is retained intact because the most recent data points are better representatives of the recent evolution of the stream. The k cluster centres become an average of the data points they represent in the stream, and this process continues indefinitely. They also extended MiLOF to MiLOF_F, which dynamically finds the number of summaries to be kept in memory. It was found to be more stable to various environments with different memory sizes compared to MiLOF and iLOF.

2.3.3 Model-Based Anomaly Detection Methods

Model-based methods usually define a sliding window over the stream and create a model that represents the behaviour of the majority of data points, then identify those patterns that deviate from the model. They incrementally update the parameters of their model as data points arrive in the stream to capture the trends in the stream, and can be further divided into three categories: (i) Statistical methods, (ii) Dynamic Bayesian Networks, and (iii) Dynamic Cluster Maintenance.

Statistical Methods

Statistical methods aim to learn a statistical model for the normal patterns in the dataset. The distribution parameters are adjusted to best represent the data in the stream. Thereafter, the data points that do not follow the learned model are referred to as anomalies.

Yamanishi et al. [148, 149] proposed an online sequential discounting method called *SmartSifter*. SmartSifter is an online learning algorithm for learning a probabilistic mixture model that represents the behaviour of the majority of the data. This model has a decay factor to cope with the drift in the stream. The score that a data point receives is based on the probabilistic fit value of it to the current model. This method is illustrated in Figure 2.12. They proposed the Sequen-

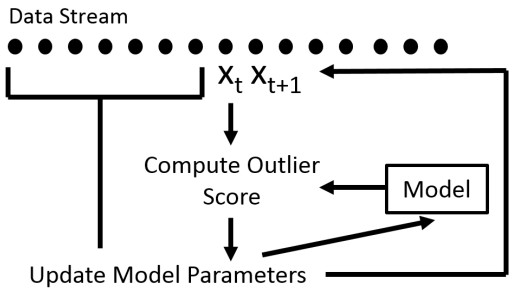


Figure 2.12: Online sequential discounting

tially *Discounting Laplace Estimation* (SDLE) for categorical variables, and *Gaussian Mixture Model* (GMM) and *Sequentially Discounting Expextation-Maximization* (SDEM) for continuous variables.

Yamanishi et al. [148, 149] proposed and online sequential discounting method *SmartSifter*. SmartSifter is an online learning algorithm for learning a probabilistic mixture model that represents the behaviour of the majority of the data. This model has a decay factor to cope with the drift in the stream. The score that a data point receives is based on the probabilistic fit value of it to the current model. This method is illustrated in Figure 2.12. They proposed the Sequentially Discounting Laplace Estimation (SDLE) for categorical variables, and Gaussian mixture model (GMM) and Sequentially Discounting Expextation-Maximization (SDEM) for continuous variables.

For data streams with a mixture of categorial and continuous attributes, Ghoting et al. [54] proposed LOADED, which has a metric for efficient determination of dependencies between categorical and continuous attributes. For categorical attributes, two data points are considered linked if they have at least one common attribute-value pair. For continuous attributes, a covariance matrix is created to capture the dependencies between them. The dependence between the data points with mixed continuous and categorical attributes is calculated by incremental maintenance of the covariance matrix. However, calculating a covariance matrix has an implicit assumption that the data share the same distribution, which may not hold for some real-life applications.

Dynamic Bayesian Networks

Dynamic Bayesian Networks (DBN) are employed by Hill et al. [64] to incorporates drifts in a data stream. In their method, a network topology is created and as the stream evolves the DBN observes new system states, and defines new state variables to represent them. Anomalies are identified by two methods: *Bayesian credible interval* (BCI) and *maximum a posteriori measurement status* (MAP-ms).

BCI employs a *Hidden Markov Model* (HMM) and uses Kalman Filtering to sequentially infer the posterior distributions of hidden and observed states as new data points arrive. These posteriors are then used to construct a Bayesian credible interval for the most recent data points. Any data point that falls outside of the $p\%$ Bayesian credible interval is classified as anomalous. In MAPms, a 2-layered DBN is used. The label of each data point is modelled as a hidden state and the maximum a posteriori estimate of the hidden state variable is used to classify the data point as normal or anomalous. The main drawback of using this model is its unintuitive input parameters.

Clustering Based Methods

One of the main approaches for anomaly detection in data streams is clustering. The goal of clustering is to partition data into several clusters where the data points in the same cluster are more similar to each other than the data points in other clusters. As a by-product of this process, some clustering algorithms [101, 70] label data points that do not belong to the identified clusters as anomalies. This membership is usually calculated using Euclidean or Mahalanobis distance.

The clustering algorithms such as [155, 26, 31, 101] were discussed in Section 2.2 and we avoid duplicating them here. Some algorithms use clustering as a tool to score the anomaliness of data points in the stream. For instance, Salehi et al. [122] proposed a relevance weighted ensemble model for anomaly detection in data streams, which scores the anomaliness of data points as the main output. The work proposed in [122] is a window based method that clusters each window and stores them in a pool of clusters. It uses a relevance function to compare the clusters in each window with the clusters of previous windows, and if a data point tends to be an anomaly in relevant windows, it is considered anomalous. Although this algorithm has been shown to have high accuracy, the computational complexity of the algorithm increases with time, limiting its

practicality for long data streams.

2.3.4 Summary of Data Stream Anomaly Detection Techniques

Data stream anomaly detection algorithms can be broadly divided into proximity-based and model-based algorithms as depicted in Table 2.3. A subset of proximity-based anomaly detection techniques can only detect global anomalies. Anomalies are defined by having less than q data points in their R proximity. Since this definition cannot identify more complex patterns, viz., clusters with varying densities in the stream, other proximity based algorithms are developed to address this challenge. These algorithms calculate the local density of clusters to identify local anomalies. The latter subset of methods generally have higher time-complexity compared to the global anomaly detection techniques, which makes them impractical for long data streams.

Model-based algorithms can be divided into statistical methods, DBNs, and clustering based methods. These techniques can identify local anomalies and also are generally more efficient than proximity-based methods. Statistical methods assume that the data in the stream is coming from a set of distributions that may not be the case in real-life applications. DBNs make no assumption on the distribution of data, but require user-input parameters that are not easily interpretable for users. On the other hand, clustering based methods are more intuitive and require more interpretable user input parameters compared to DBNs. However, their time complexity limits their practicality for long data streams.

2.4 Conclusion

In this chapter we reviewed the data stream model, data stream clustering and data stream anomaly detection. A summary of the reviewed articles can be found in Table 2.2 and Table 2.3. According to this table, data stream clustering algorithms do not identify anomalies in real-time. Identifying anomalies in real-time has a great impact on clustering performance, while having a high utility in various application domains.

For the rest of this thesis, we address the problem of online anomaly detection for data stream clustering. We also demonstrate that efficiency is an important feature for data stream anomaly detection. Then, we study data stream clustering in a high-dimensional scenario where clusters of

data tend to form in a subspace of the feature space. Finally, we propose a method for clustering that does not require a parameter relating to the shape or the number of clusters believed to exist in the stream.

Table 2.3: Data stream anomaly detection related work summary- IP: Interpretability of Parameters, LCD: Linear Complexity with Data

Approach		Algorithm	Global Outliers	Local Outliers	IP	LCD
Proximity-Based Methods	Global Anomalies	STORM [7]	✓	✗	✗	✗
		Abstract-C [151]	✓	✗	✗	✗
		LUE [79]	✓	✗	✓	✗
		DUE [79]	✓	✗	✓	✗
		COD [79]	✓	✗	✓	✗
		MCOD [79]	✓	✗	✓	✗
		LEAP [27]	✓	✗	✓	✗
	Local Density Methods	LOF [22]	✓	✓	✗	✗
		iLOF [111]	✓	✓	✗	✗
		MiLOF [121]	✓	✓	✗	✗
		MiLOFF [121]	✓	✓	✗	✗
Model-Based Methods	Statistical	SmartSifter [148, 149]	✓	✓	✗	✗
		LOADED [54]	✓	✓	✗	✗
	Dynamic Bayesian Networks	DBN [64]	✓	✓	✗	✗
	Clustering Based Methods	Ensemble Model [122]	✓	✓	✓	✗
		BIRCH [155]	✓	✓	✗	✗
		Admit [124]	✓	✓	✗	✗

Chapter 3

An Efficient Method for Anomaly Detection in Non-stationary Data Streams

Traditional anomaly detection methods are not suitable for processing potentially unbounded streams of data collected in non-stationary environments, in which the normal and anomalous behavior is subject to change over time. They require all data points reside in memory before processing. Current state-of-the-art stream anomaly detection algorithms are computationally expensive where their time-complexity grows as more data points arrive. In this chapter, we propose a cluster-based algorithm for modeling normal behavior in non-stationary data streams and detecting anomalous data points. We show that our method scales linearly with the number of observed data points, while its complexity is independent of the size of the data stream. We opt for a selective clustering approach to optimize the computation time needed to model the normal data, where the computational time for processing each window remains constant after some initial time period. Our experiments on large-scale synthetic and real-life datasets show that the accuracy of the proposed algorithm is comparable to the state-of-the-art techniques reported in the literature while providing substantial improvements in terms of computation time, which makes our method more practical in data streaming use cases.

3.1 Introduction

AN important challenge in detecting anomalies (also known as outliers) in data streams is the unbounded size of the data. Additionally, normal (i.e., non-anomalous) patterns in many streaming applications are non-stationary, which introduces further challenges for modeling normal behavior. Our focus in this chapter is on the IoT systems where a sensor node is deployed to constantly capture some features of the environment. In this context, the number of features that sensors can capture is typically in the order of 10 dimensions per sensor node. While the dimensionality of such data is relatively low, each sensor provides a potentially unbounded data

stream of measurements.

Techniques proposed for anomaly detection in non-stationary data streams tend to either have high time-complexity [122, 111], or do not detect complex patterns where clusters have varying densities [8]. Pokrajac et al. [111] proposed the iLOF algorithm for incrementally calculating LOF values for a stream of data points. The high processing time of this algorithm makes it impractical for large data streams. Salehi et al. [122] proposed an ensemble learning method for data stream anomaly detection, which applies a clustering algorithm on each window of the data stream and creates a history of clusters. A weight is calculated for the data points in an incoming window based on the relevance of previous clusters to the clusters in the new window. A problem with this method is that the weight calculation time increases as the history of clusters grows by processing each new window. The algorithm proposed by Angiulli et al. [8] has faster processing time, but it is designed for identifying global anomalies and cannot identify anomalies when clusters have different densities.

In this chapter, we introduce an unsupervised algorithm to detect anomalies in non-stationary data streams, which maintains a cluster model of normal data by a selective clustering approach. It comprises two steps, where in the first step we check if the current set of clusters can accurately explain a portion of data in the window, which is used for weighting the data points. In the second step, we perform clustering on the window if needed, viz., a portion of data in the window does not belong to any already defined clusters. The main contribution of this chapter is that its time complexity for processing a window of data is independent of previously observed data points. In other words, the time required by the algorithm to detect anomalies after some initial time period remains constant for the rest of the stream. Our experimental evaluation on large-scale synthetic and real-life datasets shows that the proposed algorithm is substantially faster than existing benchmark algorithms with no loss in accuracy.

3.2 Problem Definition

First we formally define the problem of anomaly detection in data streams. Suppose a system is being monitored and several features of interest are being measured at each time interval t, forming a data stream $X = \{x_1, x_2, \ldots, x_n, \ldots\}$ where each $x_i \in X$ is a d-dimensional data vector. We assume

Figure 3.1: Summary of the workflow of our algorithm.

a lower bound and an upper bound is available for each variable, and use them to normalise the data using min-max scaling.

The data points form a stream, where they mainly come from a set of l unknown distributions denoted by $A = \{a_1, a_2, \ldots, a_l\}$. At any time interval, data is drawn from a subset of the distributions in A. We call this subset the *active* distributions. For instance, initially, observations come from a subset $A_1 \subseteq A$ distributions. After a while, at time t_1 the data comes from another subset $A_2 \subseteq A$ where $A_1 \neq A_2$ and over time these subsets keep changing. In this problem, data is presented in sliding *windows* on the stream and the aim is to detect anomalies, which are the observations that are very unlikely to come from any distributions in A.

3.3 Methodology

We create a model of the distributions in A using Gaussian clusters, and then score observations based on those clusters which is schematically depicted in Figure 3.1. We also use Figure 3.2 as a running example, which consists of a window of 2000 observations forming three clusters as indicated by ellipses. To score observations in this window, we first identify the active distributions (in this case, Cluster 1 is active since the observations are mainly coming from its underlying distribution). Additionally, we need to identify emerging distributions, which are the distributions that are becoming active for the first time (red points at the bottom of the Figure 3.2a that are not in any cluster). Correspondingly, there are two important questions that we need to answer: (i) How to identify active distributions? (ii) How to detect emerging distributions? Before proposing a method for each question, the model that is maintained by this algorithm is presented.

The model is a set of clusters $C = \{c_1, c_2, \ldots, c_k\}$ where each c_i, $0 \leq i \leq k$ is a cluster created by a sample of an underlying distribution in A. We define a hyperellipsoidal boundary for each cluster as [114]:

$$(x-m)^T S^{-1} (x-m) \leq z^2 \tag{3.1}$$

In this formula (known as Mahalanobis distance), x is a given point (an observation), m is the center of the hyperellipsoid (the mean of the sample cluster), S^{-1} is a coefficient of the characteristic matrix of the hyperellipsoid (the inverse of covariance matrix of the sample cluster) and z^2 is a constant that marks a specific level set of the distribution. When z^2 is chosen from the cumulative inverted chi-squared distribution with d degrees of freedom for probability value $P((\chi^2_d)^{-1}_P)$, the probability of an observation falling outside the boundary will be $1-P$.

Upon arrival of a window, our algorithm performs four main steps. First, active clusters in C are identified. Second, the algorithm looks for any emerging distributions that need to be modelled and added to C. Third, if there are emerging distributions in the window then the clustering algorithm is called to update the model, and finally, based on the identified active clusters a score is calculated for each observation. This score determines to what extent an observation is anomalous. The pseudo code of the algorithm is provided in Algorithm 1.

1. Active Cluster Identification: In this step, we utilize the knowledge obtained by previous windows by testing whether a subset of observations in a window can be explained by the existing clusters in the model. As a result, instead of re-clustering each window, we check the model to find clusters that reflect the observed distribution of data, i.e., active clusters.

We consider a cluster to be active if it meets two criteria. First, it should be *feasible*, i.e., a reasonable number of observations in the window should reside in the cluster. Second, the observations in the cluster should be dispersed over the entire volume of the cluster. Based on this, we have devised a two stage mechanism to identify active clusters.

In the first stage, by finding feasible clusters we aim to quickly filter out clusters with few observations inside them. Feasible clusters are identified using the *Cumulative Binomial Probability* (CBP) function. If the CBP of the number of observations in a cluster is greater than 0.5, it is considered feasible. The reason for using the CBP value is that if the observations had been distributed randomly over the instance space, the CBP value of the observations in each cluster would have been 0.5. Hence, if this value for a cluster is greater than 0.5, it indicates that there is a concentration of observations in the cluster. In Figure 3.2a, Cluster 1 with area 0.03 and 1771 observations yields a CBP value of 1, Cluster 2 (which has an overlap with Cluster1) with area 0.04 and 120 observations yields a CBP value of 0.99, and Cluster 3 with area 0.03 and 31 observations yields a CBP value of 4.59e-9. As a result, Cluster 1 and 2 are considered feasible but Cluster 3 is

not feasible since it does not reflect the main density of the data.

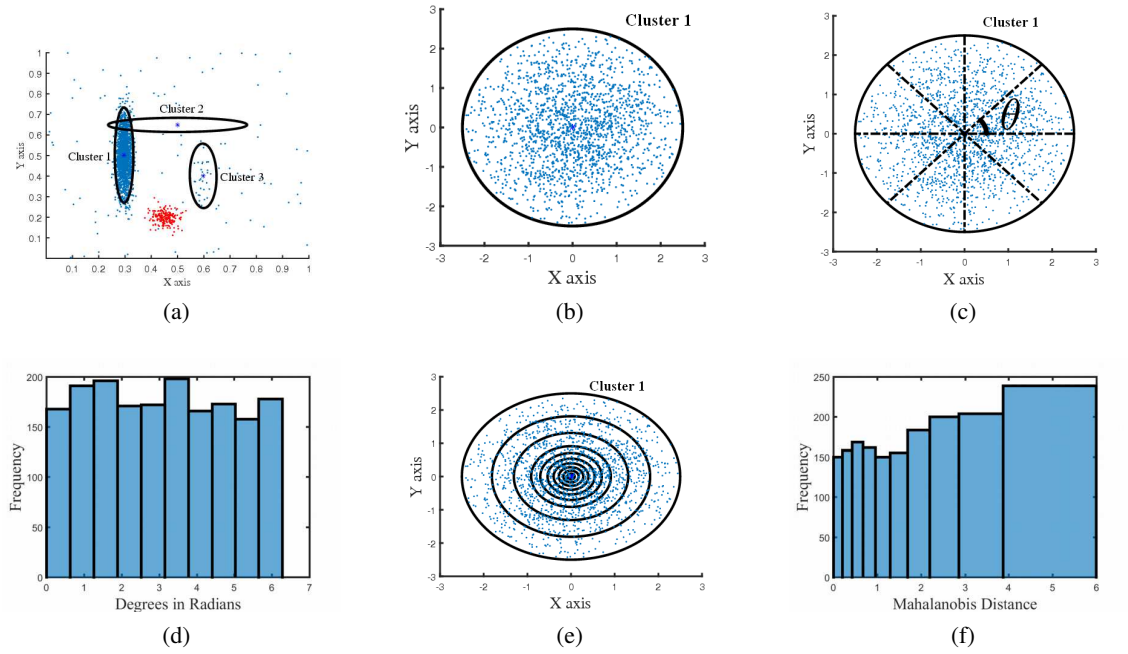


Figure 3.2: Detecting active clusters. (a) A set of clusters and a window of observations. (b) Cluster 1 is transformed into a standard Gaussian distribution. (c) Dimension ϕ_1 in the spherical coordinate system is divided to k parts. (d) Histogram of the frequency of observations in dimension ϕ_1 (e) Dimension r is divided to k parts. (f) Histogram of the frequency of observations in dimension r.

In the second stage, we quantify the validity of feasible clusters (i.e., their quality of fit) for a given set of observations, by determining if the observations are dispersed over their entire volume. By doing so, we filter out situations like Cluster 2 in Figure 3.2a where there is a large number of observations inside a cluster, but it is very unlikely that they are drawn from the underlying distribution of the cluster. To do so, they are first transformed into standard Gaussian distributions [59], then transformed into a spherical coordinate system [63] $(r, \phi_1, \phi_2, \ldots, \phi_{d-1})$ with the origin of the coordinate system at the center of the cluster. We can easily show that Gaussian distributions follow a uniform distribution with respect to the angular coordinates. In other words, if observations fit the cluster, we expect to see a uniform distribution of observations with respect to angular coordinates. Accordingly, the angular coordinates are divided into L equal parts (buckets), but since the distribution of data with respect to dimension r is not uniform, the value of $((\chi^2_d)^{-1}_P)$ in Equation 5.1 is used to divide this dimension. The parameter P, which determines the percentage

of observations to reside in the cluster is divided into L buckets (so in each interval we expect to see $(P/L)\%$ of observations). For each dimension a histogram is created and the entropy of the histogram is calculated. The entropy value quantifies the dispersion (uniformity) of observations in the cluster with respect to that dimension. For instance, Figure 3.2c and 3.2e show the partition of dimensions into L parts and Figure 3.2d and 3.2f show the histogram of each dimension. In this example, the entropy of the histogram in Figure 3.2d is 0.99, and for the histogram in Figure 3.2f it is 0.92. If the entropy of all dimensions are above a threshold value E, it means observations are dispersed evenly over all dimensions and the cluster is considered active.

Algorithm 1 Processing a window of observations

```
1:  procedure ProcessWindow(W_t, C)        ▷ W_t : current window of observations. C : set of clusters modelled so far
2:      for all C_i ∈ C do
3:          mems ← FindMembersOfCluster(C_i, W_t)
4:          CBP ← FindCumBinProb(mems, W_t, V_i)
5:          if CBP > 0.5 then
6:              stanNorm ← standardize(mems, C_i)
7:              spherical ← toSphericalCoords(stanNorm)
8:              active ← sweepSphericalDims(spherical)
9:              if active is true then C ← C ∪ C_i
10:             end if
11:         end if
12:     end for
13:     C ← C ∪ IdentifyEmergDistributions(C, W_t)
14:     scores ← ComputeGaussProbabDensity(C, W_t)
15: end procedure
```

2. Emerging Distribution Identification: In a window of observations, after detecting active clusters in C, there might be distributions in A that have become active for the first time. Therefore, there are no clusters in C to model them. The aim of this step is to look for these distributions.

To detect emerging distributions, all observations belonging to active clusters are removed from the instance-space, and the remaining observations are projected into the existing dimensions in the Cartesian coordinate system. Each dimension, which is in the range $[0,1]$, is divided into L' equal intervals and the number of observations in each interval is calculated. If the CBP of the number of observations in an interval is greater than 0.5, it shows that there is a concentration of data that has not been modelled in C, and the clustering algorithm is called. Otherwise our algorithm proceeds to the scoring step.

Note that the value of L' determines the rate of observations in a window belonging to an emerging distribution that we want to detect at this step. In a window of length wl, there could be at most $wl/2$ anomalies (since the number of normal data points is greater than the number of anomalies). If anomalies were distributed randomly over the instance-space, we would expect to see at most $wl/2L'$ anomalies in each interval. On the other hand, suppose an emerging distribution with n observations is partitioned into L' intervals in the scanning stage. Therefore, there would be n/L' observations in each interval (since we are considering a worst case scenario, suppose there is no concentration of data in any of those L' intervals and the observations are evenly distributed in them). In order to detect this emerging distribution, we should have: $n/L' > wl/2L$, therefore $n/wl > L'/2L$. As a result, if the rate of observations belonging to an emerging cluster in a window (n/wl) is greater than ($L'/2L$), it is detected by the algorithm.

Moreover, there tends to be a relation between L' and L. If L is set to a large value, L' would increase because by increasing the number of intervals, observations would be distributed over more intervals. As a result, setting L to a large value does not necessarily increase the accuracy of this step. Accordingly, in the parameter setting section we analytically set this parameter to an appropriate value.

Algorithm 2 Detecting emerging distributions

```
1: procedure IDENTIFYEMERGDISTRIBUTIONS(C, W_t)
2:     C' ← ∅
3:     outsiders ← RemoveObsInClusters(C, W_t)
4:     flag ← SweepCartesianDims(outsiders)
5:     if flag is true then
6:         C' ← HyCARCE(W_t)
7:     end if
8: end procedure
```

3. Clustering: For clustering, we have employed HyCARCE [103], which is a near-linear density-based algorithm. The reason for choosing HyCARCE is its accuracy and efficiency in clustering. Given a window of observations, it approximates underlying distributions with mean μ and covariance matrix Σ with clusters where the mean is estimated as m and the covariance matrix as S for the cluster. An advantage of this algorithm is that any distribution can be approximated by a mixture of Gaussian distributions. At this step, given a window of observations, the whole window is clustered with HyCARCE and the clusters are added to the model. Note that when the

whole window of observations is clustered and clusters are added to the model, there might be a cluster that is already in the model. This scenario happens when a fraction of a window is from an emerging distribution. In this case, there would be multiple clusters of a single distribution, which might affect the computation time in a negative way, but note that it is limited to the number of distributions in A.

4. Scoring: By approximating the distributions in A with clusters and identifying active clusters, we detect and track active distributions in the stream which can be used to calculate the likelihood of observing a data point, given a distribution. Let $C' = \{c'_1, \ldots, c'_{l'}\}$ be the set of active clusters and x_t be an observation at time t. The value $P(x_t \mid c'_i), i \in \{1,2,\ldots,l'\}$ is the probability of observing x_t given the cluster c'_i, which is calculated using the Gaussian probability density function of the cluster. As a result, the score of an arbitrary observation x_t in a window is calculated as:

$$A(x_t) = max(\ \{P(x_t \mid c'_i) :\ \ i = 1,2,\ldots,l'\}).$$

At this stage, the algorithm terminates for the current window, the scores are normalized with the MinMax algorithm, and the observations having a score near to 1 are most normal and the ones having a score near to 0 are most anomalous. This algorithm is implemented in Matlab[1] and publicly available[2].

Time Complexity: In order to model the time complexity of our algorithm, let wl be the window length, l be the size of the cluster model, d be the dimension of the data and L and L' be the number of parts into which the hyperellipsoids and the instance-space are divided, respectively.

For the first step, calculating the number of observations in each cluster is computed in $O(l \times wl \times d^2)$ as we need to calculate the Mahalanobis distance of each data point from each cluster. Transforming clusters into the standard Gaussian distribution is computed in $O(l \times d^3)$ because we need to calculate the covariance decomposition (in our algorithm we used Cholesky covariance decomposition) which has a time complexity of d^3 for each of the l clusters. Scanning the hyperellipsoids is done in $O(l \times wl \times d \times L)$ since each cluster needs to be divided into L parts, and the data points in each part should be calculated. This all leads to a total time complexity of $O(l \times wl \times d^2 + l \times wl \times d \times L + l \times d^3)$. For the second step, scanning the instance-space and

[1] https://mathworks.com/ [2] https://github.com/mchenaghlou/EfficientAnomalyDetectionAlgorithm-GlobeCom2017

counting the number of observations in each part is done in $O(wl \times L' \times d)$. For the third step, calling HyCARCE to cluster the window has time complexity $O(wl)$, since HyCARCE is near linear, and the final step is calculated in $O(l \times wl)$. Altogether, the time complexity of our algorithm is $O(l \times wl \times d^2 + l \times d^3 + l \times wl \times L \times d + wl \times L' \times d)$ in the worst case, which is linear with respect to the size of the model and the window length. These characteristics make this algorithm a suitable solution for anomaly detection in streaming data.

3.4 Evaluation

In this section we aim to evaluate our algorithm in terms of its accuracy and efficiency, and compare its performance with other algorithms. We have compared our algorithm with two state-of-the-art algorithms, i.e., iLOF [111], which is widely used in the literature, and a more recent method called the Ensemble model [122]. We describe our evaluation setup by detailing our parameter selection, comparison measures and experimental datasets, and finally, the results.

Parameter Setting: There are four parameters in this algorithm, namely the entropy threshold (E), the spherical coordinate system division number (L), the Cartesian coordinate system division number (L') and the window length (wl).

The goodness of fit of observations in clusters can be tuned by E. When E is set to a high value (near to 1), observations should perfectly fit the clusters, for clusters to be considered active. As a result, more clusters are formed in the model, which imposes a high computational cost but results in greater accuracy. On the other hand, if E is set to a low value (near to 0), clusters that the observations do not follow the underlying distribution are detected as active. As a result, fewer clusters are formed in the model, and the results would be less accurate but the algorithm would be faster. In our experiments, we have set this value to 0.75 to provide a trade-off between the accuracy and the speed of the algorithm, and we recommend a value from the range $[0.7, 0.85]$ for this parameter.

The range of values for the parameter L is the set of positive integers, and it represents the number of bins in a histogram. Therefore, to set a value to L we extend the well-known Freedman–Diaconis rule [49] for choosing the optimal bin-width, i.e., $2 \times IQR(X') / (n')^{1/3}$, where $IQR(X')$ is the interquartile range for the set of observations residing in the cluster (X') and n' is the size of

X'. If we assume data is evenly distributed in the cluster, $IQR(X') = r'/2$, where r' is the range of values in the designated dimension, then

$$L = r'/(2 \times \frac{IQR(X')}{(n')^{1/3}}) = r'/(2 \times \frac{r'/2}{(n')^{1/3}}) = (n')^{1/3}. \tag{3.2}$$

The parameter L' can also be set by Equation 3.2, but in this case since the observations are in the original coordinate system the value of r' is 1 (observations are normalized) and n' is the number of observations that do not belong to any active clusters. We observed that these parameters are mostly chosen from the range [6, 12].

Window length is a parameter that is inherent to the problem and has been tested with a variety of values. In our experiments, we observed that having smaller window lengths results in having a larger model, which increases the computation time. The reason for this is that having a smaller window length means having fewer observations of a distribution in each window. As a result, in order to have an accurate model of the distribution, more windows need to be clustered, which results in a larger model. On the other hand, having a large window length means having more observations of a distribution in a window, which can be used to model the distribution with fewer windows.

Performance Measure: We have implemented our algorithm in MatLab[3], and evaluated the algorithms in terms of computation time and the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC). These are common measures in the literature to compare the effectiveness of algorithms in this field. The tests are conducted on an Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz with 16.0 GB main memory and a 64-bit Windows 7 operating system.

Datasets: Our tests are conducted both on synthetic and real world datasets and we have made our synthetic datasets available for interested readers[4]. Since iLOF is computationally expensive on large datasets, we have compared the *accuracy* of our algorithm with iLOF and the Ensemble model on small datasets, and compared the *computation time* and *accuracy* of our algorithm with the Ensemble model on larger datasets.

We have provided 18 synthetic datasets divided into 2 parts. In the first part there are 9 small datasets, each consisting of around 5000 observations, and in the second part there are 9 large datasets, each consisting of around 150,000 to 250,000 observations. In each part, there are

[3] https://au.mathworks.com/products/matlab/ [4] https://goo.gl/SDJ6Wh

Table 3.1: A summary of the structure of the datasets. We have provided 18 synthetic datasets divided into 2 parts. In the first part there are 9 small datasets, each consisting of around 5000 observations, and in the second part there are 9 large datasets, each consisting of around 150,000 to 250,000 observations. In each part, there are streams in 2D, 3D and 4D and in each dimension there are 3 types of streams.

Small Datasets	2D	Stream Type 1
		Stream Type 2
		Stream Type 3
	3D	Stream Type 1
		Stream Type 2
		Stream Type 3
	4D	Stream Type 1
		Stream Type 2
		Stream Type 3
Large Datasets	2D	Stream Type 1
		Stream Type 2
		Stream Type 3
	3D	Stream Type 1
		Stream Type 2
		Stream Type 3
	4D	Stream Type 1
		Stream Type 2
		Stream Type 3

streams in 2D, 3D and 4D and in each dimension there are 3 types of streams. This structure is illustrated in Table 3.1. To identify streams in the results section, each stream is labelled, where T1-2D denotes the 2 dimensional data stream of type 1. For each type, there are 10 streams with different random rates of anomalies from 1% to 10% of the size of the dataset. Anomalies are drawn from a uniform distribution over the instance space and are randomly inserted into the stream.

For real world datasets, we have employed the TAO[5] dataset and LWSNDR[6] which are publicly available. For TAO, we have chosen the site (5°S, 156 °E) with features Air Temperature, Relative Humidity and Shortwave Radiation measured every 10 minutes from 1 January 2011 until 1 February 2013. This dataset consists of 18,312 observations, which are rated with quality codes that determine the quality of the measured values. The second dataset consists of measurements of humidity and temperature, made every 5 seconds during a 6 hour interval with TelosB motes.

[5] http://www.pmel.noaa.gov/tao [6] http://www.uncg.edu/cmp/downloads/

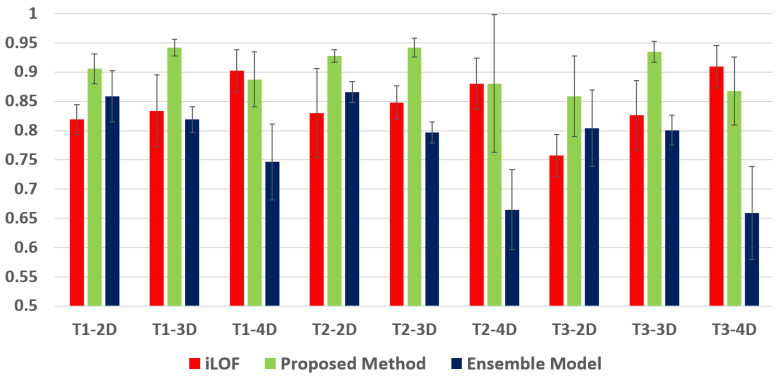


Figure 3.3: The AUC values for small synthetic datasets with error bars. The proposed method has a comparable, and in many cases, a better performance compared to the Ensemble model and iLOF in all datasets.

Table 3.2: The AUC values for the small real world dataset, with different settings for the window length parameter.

Window Length	200	400	600	800	1000
Proposed Method	**0.94**	**0.91**	**0.89**	**0.85**	0.80
Ensemble model	0.89	0.87	0.79	**0.85**	**0.89**
iLOF	0.64	0.64	0.64	0.64	0.64

The environment is altered manually to generate artificial anomalies. In our experiments, we have used the *singlehop_indoor_moteid1* dataset which consists of 4,417 observations.

Results and Discussion: Having various window lengths for our algorithm and the Ensemble model, the iLOF performance is reported for the best neighbour parameter in the range of [5, 15]. It is important to note that for each stream type, the designated algorithms have been executed 10 times, each time having a random uniform rate of anomalies from 1% to 10%, and the mean of the results is presented with error bars.

Figure 3.3 and Table 3.2 show the AUC values of the algorithms on the small synthetic and real world datasets, respectively, where the results of our algorithm are comparable with the other two algorithms and in many cases outperforms them.

Figure 3.4 illustrates the AUC value and computation time of the Ensemble model and our algorithm on large synthetic datasets. Note that for brevity, only the execution time for the dataset type 1 is presented in Table 3.3. The execution time of the algorithms on datasets type 2 and type 3 is similar to type 1. Large datasets vary from 150,000 to 250,000 observations and as can be seen, our algorithm outperforms the Ensemble model in all dimensions and data types (in terms of

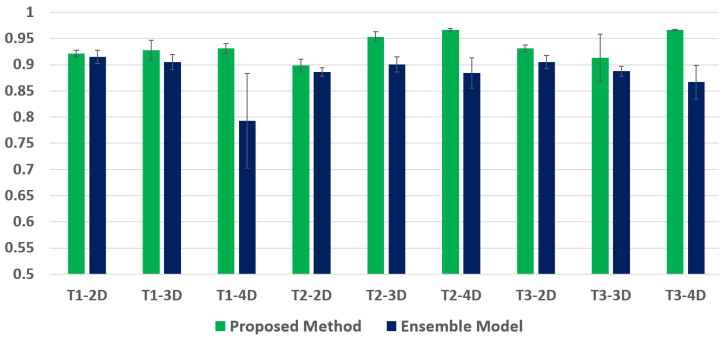


Figure 3.4: AUC values with error bars for large synthetic datasets. The proposed method has a better performance compared to the Ensemble model in all datasets.

Table 3.3: Execution times on large synthetic datasets (seconds) with various settings for the window length parameter.

Window Length	200	400	600	800	1000
Proposed Method 2D	**3.39**	**2.40**	**2.15**	**1.74**	**1.38**
Ensemble model 2D	783	374	230	135	100
Proposed Method 3D	**27.16**	**19.96**	**16.61**	**10.93**	**7.99**
Ensemble model 3D	777	351	217	115	63
Proposed Method 4D	**35**	**33**	**31**	**25**	**20**
Ensemble model 4D	401	377	260	162	101

both time and accuracy), and proves to be the fastest in all cases.

The reason for the speed up of our algorithm is that HyCARCE is called a limited number of times to model the distributions in S. Therefore, the complexity of the model is independent of the size of the data stream, and when all the distributions are learned by our algorithm, the computation time would remain constant for the rest of the stream. However, in the Ensemble model algorithm, since the complexity of the model grows with the number of observed windows, the computation time of processing a single window increases as windows arrive. Figure 3.5 illustrates that the computation time of processing windows of length 200 on the T3-3D dataset for the Ensemble model grows steadily with time, but the computation time of our algorithm grows for a certain time and remains relatively constant for the rest of the stream. Other datasets show similar patterns.

Table 3.4 and Table 3.5 present the computation time and AUC value of the Ensemble model and our algorithm on the TAO dataset. Our algorithm outperforms the Ensemble model for all window lengths, and has substantially lower computation time. There are two points that need to

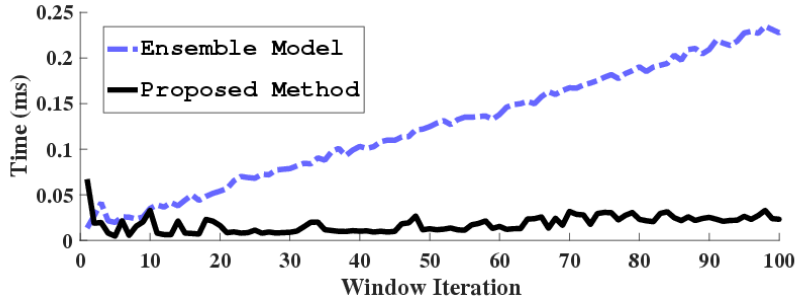


Figure 3.5: Computation times of the Ensemble model and our algorithm with window length set to 200 over successive windows on T3-3D. This shows that for the Ensemble model the computation time grows steadily with time, but for our algorithm it grows for a certain time and remains relatively constant for the rest of the stream.

Table 3.4: Execution times on TAO dataset (seconds) with various settings for the window length parameter.

Window Length	200	400	600	800	1000
Proposed Method	**2.22**	**0.88**	**0.53**	**0.37**	**0.33**
Ensemble model	38.78	12.62	5.94	3.29	2.26

be explained. First, since the size of datasets are fixed, having larger windows means having fewer windows. When the number of windows is small, there is not much difference in the execution time of these algorithms. This is illustrated in Figure 3.5 in the first few iterations which is because the size of their model is similar. This is also observed in Table 3.4 where the window length is 1000. In this case, there are about 20 windows and the computation times of both algorithms are similar, but as the number of windows increases, the improvement is more substantial. Second, in Table 3.4, as mentioned in the window length parameter setting section, it is observed that the computation time increases as the window length decreases. This is because with smaller window lengths the size of the model grows, which makes the algorithm slightly slower.

Table 3.5: AUC values on TAO dataset with various settings for the window length parameter.

Window Length	200	400	600	800	1000
Proposed Method	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
Ensemble model	0.97	0.97	0.97	0.96	0.95

3.5 Conclusion and Future Work

In unbounded data streams where underlying patterns change over time, detecting anomalies is a major challenge. State-of-the-art algorithms either have high computational time or only identify global anomalies. In this chapter, we proposed an efficient unsupervised clustering-based algorithm to detect anomalies in non-stationary data streams. The computational time of our algorithm is optimized by selective clustering in a targeted manner where clustering is performed a limited number of times. This algorithm comprises two steps; In the first step, it identifies active clusters by calculating the dispersion of data points over the volume of clusters. Clusters that have a large number of data points over their volume are considered active clusters and are used to score those data points. In the second step, if there are data points that are not inside any already defined clusters, it applies a clustering algorithm on them to model any possible emerging clusters. When all clusters are identified by the algorithm, then it only performs the first step for the rest of the stream. We showed that the computational complexity of our algorithm is linear w.r.t the observed data points, and the processing time of each window remains constant after some initial time period. However, this algorithm works on windows of fixed length, which may be difficult to set in various application domains. Moreover, some applications such as intrusion detection and fraud detection require detecting anomalies as early as possible, therefore real-time anomaly detection methods are required. In the next chapter, we propose an online clustering algorithm that performs anomaly detection in real-time and automatically sets the length of the window.

Chapter 4

Online Clustering for Evolving Data Streams with Online Anomaly Detection

In the previous chapter, we proposed an efficient anomaly detection algorithm that processed data in batches, i.e., windows of fixed length. In some application domains, such as intrusion detection, real-time anomaly detection is desired. This requires data points to be processed one at a time, and then incrementally update the model to adapt to changes in the data streaming environment. In an evolving data stream, online clustering algorithms are required to both (a) assign observations to clusters incrementally and (b) identify anomalies in real-time. Current state-of-the-art algorithms in the literature do not address feature (b) as they only consider the spatial proximity of data, which results in poor clustering and poor demonstration of the temporal evolution of data in noisy environments. In this chapter, we propose an online clustering algorithm that considers the temporal proximity of observations as well as their spatial proximity to identify clusters and anomalies in data streams in real-time. It identifies the evolution of clusters in noisy streams, incrementally updates the model and calculates the minimum required window length over the evolving data stream to avoid jeopardizing performance when using our method for memory constrained applications. To the best of our knowledge, this is the first online clustering algorithm that identifies anomalies in real-time and discovers the temporal evolution of clusters. Our findings are supported by experiments on synthetic as well as real-world data.

4.1 Introduction

TRADITIONAL clustering differs from stream clustering in several important ways. First, traditional algorithms have a *global* view of the data, i.e., all observations are randomly accessible whereas in the streaming environment a window is defined that slides over the stream, creating a *partial* view of the data. Second, in streaming environments, clusters of data emerge,

evolve and change over time but static datasets lack this kind of temporal evolution within the current data window and therefore an effective approach to stream clustering should discover the temporal evolution of clusters as well.

Current approaches for data stream clustering are: (i) data summarization clustering [81, 26], (ii) online (real-time) clustering [29, 93], and (iii) time-series clustering [101, 6]. The first approach creates a statistical summary of observed data with the help of a sliding window to give an approximate global view of the stream [81, 26]. In the second approach (online clustering), model construction and data labeling are performed simultaneously, i.e., as observations arrive into memory (or while present in memory) they are assigned to clusters. This category imposes the fewest constraints on the type of data stream, which makes it the most general approach and most challenging. Only a handful of methods have been proposed in this category [29, 93]. The third approach (time-series clustering) exploits the *principle of locality* assumption, which states that observations close in time are usually close in value, which arises in various practical application domains [101, 6]. The primary focus of this chapter is online clustering where the main challenges are (i) the stream *evolves* gradually and identifying the evolution of patterns in noisy environments is not trivial, and (ii) algorithms do not have a global view of the data but instead a partial view, i.e., a window that slides over the stream. Current state-of-the-art algorithms in this category are sequential K-means and competitive neural network based algorithms such as *Adaptive Resonance Theory* (ART-2) [29] and *Self-Organizing Map* (SOM) [77].

Online clustering is quite similar to online classification with novelty detection. In online clustering, at the beginning of the stream it is assumed that data comes from a single underlying cluster for a few timestamps, so the algorithm can create a model for it, while in online classification [118] there are two phases. In the initial training phase, they build a model based on a labelled dataset that is available. In the second phase, new data points are classified using this model, and new classes that deviate substantially from known classes are identified as novelties. In other words, online clustering is similar to online classification with novelty detection where in the first phase, there is only one class in the labelled dataset.

A limitation of current state-of-the-art online clustering algorithms is that they do not identify the evolution of clusters in streams effectively, mainly because they only consider the *spatial proximity* of data in the stream. In this chapter, we propose an algorithm that considers the *temporal*

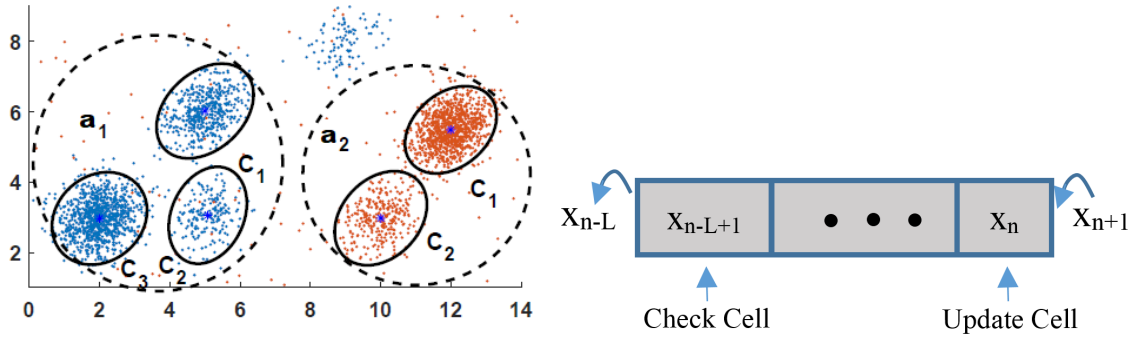
proximity of observations as well as their spatial proximity to identify anomalies in real-time. Our algorithm incrementally updates the model and identifies the evolution of clusters. It takes an input parameter to calculate the minimum window length required to avoid jeopardizing performance and detects the number of clusters automatically.

4.2 Problem Definition

In this section, the problem of evolving data stream clustering is defined. Suppose a stream of observations $X = \{x_1, x_2, x_3, \ldots, x_n, \ldots\}$ are generated from a set of l unknown distributions denoted by $A = \{a_1, a_2, \ldots, a_l\}$ where each $a_i \in A$ is a mixture of ζ_i components. Let $\Theta = \{\theta_{i,j}, j = 1, \ldots, \zeta_i\}$ denote the parameters corresponding to the components of a_i, e.g., if a_i is a multivariate Gaussian distribution, $\theta_{i,j} = \{\mu_{i,j}, \Sigma_{i,j}\}$ determines the mean and the covariance matrix of the j^{th} component and the mixture weights for $a_i \in A$ are denoted by $\Phi = \{\phi_{i,j}, j = 1, \ldots, \zeta_i\}$.

Each observation $x_i \in X$ (drawn from a random $a_i \in A$) is a feature vector $(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ where each $x_i, 1 \leqslant i \leqslant d$ is an arbitrary type of variable such as temperature in x_i. A small subset of observations are i.i.d random anomalies, i.e., do not come from any distributions in H. X is potentially unbounded and there is no control over the order of which observations arrive. At any time interval, one of the distributions in H is active (contributing to the stream) and active distributions change over time. When a distribution becomes active for the first time it is called an *emerging* distribution We aim to cluster X into several dense and separated partitions and identify the temporal evolution of data. A good clustering algorithm should reflect the structure imposed by $\theta_{i,j}$ in the input space, and filter out anomalies.

Data Presentation: Observations become available to the algorithm one at a time and data is presented in a window of length *wl* illustrated in Figure 4.1b. When a new observation arrives, all observations in the window move one cell to the left (the observation at the leftmost cell is discarded) and the new observation is put in the rightmost cell.



(a) Underlying distributions

(b) Sliding window (partial view of data) of length L

Figure 4.1: Motivating example of clusters in an evolving data stream

4.3 Methodology

In this section we propose OnCAD (Online Clustering and Anomaly Detection). In a non-stationary environment where clusters of data emerge, evolve and change over time, a new observation may: (1) *belong to an existing cluster in the model*, (2) *be an anomaly*, or (3) *be part of an emerging cluster*. Our goal is to determine the fate of an observation, update the model, and represent each underlying component with a cluster. We name the leftmost cell in the window as the *check cell* and the rightmost cell as the *update cell* because of their importance.

When a new observation x_{n+1} arrives, the algorithm performs two main steps:

Step 1: It checks if x_{n+1} belongs to case 1 by determining its membership to the clusters in the model and updating them.

Step 2: If x_{n+1} does not belong to any clusters in the model, it should be assigned to either case 2 or 3. Distinguishing between these two cases is impossible at this stage, therefore we postpone the assignment of x_{n+1} until it is in the check cell where either a new cluster is formed or the observation is labelled as anomalous. At this stage one iteration of the algorithm terminates and the algorithm proceeds to the next iteration. However, to perform these steps the main questions that we need to answer are:

1. How to define the membership of observations to clusters?
2. How to distinguish between anomalies and emerging clusters?
3. How to calculate the minimum window length (wl)?

Answering these questions is not trivial because the algorithm has only a partial view of the data that evolves over time. The pseudo-code of OnCAD is presented in Algorithm 3.

4.3.1 Step 1 (Cluster Update Rule):

In this section, we answer question 1 by building a cluster model for the underlying components in the stream using Gaussian clusters $C = \{c_1, c_2, \ldots, c_k\}$ where the cluster prototypes are hyper-ellipsoidal and each cluster is represented by its mean and the inverse of its covariance matrix. To determine the membership of x_{n+1} w.r.t the clusters in the model, we employ the boundary definition in [114] for hyper-ellipsoids:

$$(x-m)^T S^{-1}(x-m) \leq z^2. \tag{4.1}$$

In this formula, x is an observation, m is the center of the hyper-ellipsoid (the mean of the cluster), S^{-1} is the inverse of covariance matrix of the cluster and z^2 is a constant that marks a specific level set of the cluster. When z^2 is chosen from the cumulative inverted chi-squared distribution with d degrees of freedom for a probability value P, $((\chi_d^2)_P^{-1})$ sets the boundary of the cluster such that the probability of an observation falling outside the boundary is $1-P$. In our experiments, we set the value of P to 0.99.

If x_{n+1} falls inside the boundary of a cluster, we update it by the weighted incremental update formulas in [100] (Equations 20 and 23). These formulas are used to update $m_{n,i}$ (the mean of the cluster at time n) and $S_{n,i}^{-1}$ (the inverse of covariance matrix of the cluster at time n) with x_{n+1}. By determining the membership and updating the selected clusters step 1 terminates.

4.3.2 Step 2 (Detecting Emerging Clusters)

The key to identifying emerging clusters from anomalies is the window length *wl* that defines the temporal locality of observations. If *wl* is large enough to fit the whole dataset, it is identical to batch-mode clustering, and if *wl* is too small the algorithm would not be able to capture any statistically significant patterns. Hence, our aim is to calculate the smallest *wl* that enables emerging clusters to be distinguished from anomalies.

The prominence of the ζ_i components of a distribution in the window can be described by the

Algorithm 3 One iteration of OnCAD

Inputs
    W current window
    x_{n+1} the newly arrived data point
    C set of identified clusters
    q minimum sample size
    ε radius for DBScan
    cbt cluster boundary threshold
procedure ONCAD(W, x_{n+1}, C, q, ε, cbt)
    member_clusters $\leftarrow \emptyset$
    member_dists $\leftarrow \emptyset$
    $M_{x_{n+1}} \leftarrow 0$
    ▷ Step 1 operations
    for each $c \in C$ **do** ▷ Calculate Mahalanobis distances
        cur_dist $\leftarrow$ Cal_Mahal_dist(x_{n+1}, c)
        if cur_dist $\leq cbt$ **then**
            $M_{x_{n+1}} \leftarrow 1$
            member_clusters $\leftarrow$ member_clusters $\cup\, c$
            member_dists $\leftarrow$ member_dists $\cup$ cur_dist
        end if
    end for
    membership_weights $\leftarrow$ cal_weights(member_dists) ▷ Calculate weights with Mahalanobis distances
    if $M_{x_{n+1}} == 1$ **then**
        for each $c \in$ member_clusters **do**
            update_clusters(x_{n+1}, c, membership_weights)
        end for
    end if
    ▷ Step 2 operations
    if $M_{x_{n-wl+1}} == 0$ **then**
        non_members $\leftarrow$ find_all_non_members_in_M(M)
        if length(non_members) $\geq q$ **then**
            $C' \leftarrow$ DBScan(non_members, ε, q)
            for each $c' \in C'$ **do**
                $c \leftarrow$ calculate_mean_covariance_matrix(c') ▷ Create hyperellipsoidal cluster prototype
                $C \leftarrow C \cup c$
            end for
        end if
    end if
end procedure

mixture-weight and *sample size*. Therefore, we set a threshold ϕ_{min} (parameter set by the user) on mixture-weights to cap the number of emerging components and calculate the minimum number of observations to model each component with a cluster (sample size). As a result, the value of wl is calculated such that the algorithm is capable of detecting an emerging mixture distribution $a_i \in A$ with ζ_i components where their mixture-weights are greater than ϕ_{min}. For instance, if ϕ_{min} is set to 0.2, the algorithm can detect any emerging $a_i \in A$ where $\phi_{i,j} \geq 0.2$ for all $j = 1, \ldots, \zeta_i$, i.e., it can detect at most 5 simultaneous emerging components.

Assume $X_s = \{x_1, x_2, \ldots, x_{n'}\}$ is a sample of size n' from a single random component with mean μ and covariance matrix Σ. Also, let m and S be the maximum likelihood estimates (unbiased) of μ and Σ based on X_s, respectively. We compute n' such that the distance between m and μ is less than a threshold by calculating an elliptical confidence region (ECR) for μ with a high confidence level.

A confidence region $CR \subset \mathbb{R}^d$ for $\mu \in \mathbb{R}^d$ with confidence level $1-\alpha \in (0,1)$ is a region where $P(\mu \in CR) \geq 1-\alpha$ and it is defined as [59]:

$$CR = \left\{ \mu \in \mathbb{R}^d : (m-\mu)S^{-1}(m-\mu)' < \frac{d}{n'-d} F_{1-\alpha;d,n'-d} \right\} \tag{4.2}$$

where d is the dimensionality of the data, n' is the sample size, F is the F-inverse cumulative distribution and α is the confidence level. Here, the term $(m-\mu)S^{-1}(m-\mu)'$ is the Mahalanobis distance between m and μ based on S^{-1} and the term $\frac{d}{n'-d}F_{1-\alpha;d,n'-d}$ sets the boundary of the confidence region where the user usually sets α to a fixed value as the confidence level. Therefore, it is the value of n' that determines the boundary of the ECR and as n' grows, the area (volume) of the ECR shrinks and the cluster becomes a more accurate representation of the component.

To model a component with a cluster, Equation 4.2 gives us an effective tool for achieving any desired precision. Our aim is to minimize n' but the question is how small n' should be. The criterion that we define for minimizing n' is that n' should be set to a value such that only one cluster is formed for each component. Having X_s in memory and creating the cluster C_s based on it, we need the majority of the subsequent observations of the component to fall inside C_s. Otherwise, the algorithm may form another cluster for the same underlying component. Therefore, the ECR should be small enough (or n' should be large enough) such that C_s covers at least half of the component. To do so, we combine the boundary definitions in Equations 5.1 and 4.2 to extract the

maximum n'. Therefore, we find the maximum n' such that

$$\frac{d}{n'-d}F_{1-\alpha;d,n'-d} \leqslant P((\chi^2_d)^{-1}_P). \tag{4.3}$$

This equation means that the true mean μ is within the boundary of the cluster C_s that covers at least $P\%$ of the observations of C_s. Accordingly, by having P set to 0.5, μ would be within the boundary of C_s such that it covers at least half of the observations in the component. In practice, the value of n' with a 95% confidence level in a two dimensional environment is 11 and it grows to 69, in a 20 dimensional environment. Finally, the value of wl is calculated as $wl = \lfloor 1/\phi_{min} \rfloor \times n'$.

Detecting Emerging Clusters: Detecting emerging clusters is achieved by performing a batch clustering algorithm on the window. In the second step, if the observation at the check cell is not assigned to case 1, we extract all such observations from the window and apply DBScan [44] to them. DBScan is a well-known clustering algorithm on static data that requires two parameters $MinPts$ and ε that we extract in the initialization phase. Accordingly, any calculated cluster by DBScan is added to the model. Although using DBScan limits the practicality of the algorithm to data streams where clusters have similar densities, it can be replaced by any other batch clustering algorithm. This concludes one iteration of the algorithm for updating the current model and identifying anomalies from emerging clusters.

Initialization: We have demonstrated that n' observations are sufficient to represent a cluster in this methodology, so we set $MinPts$ to n' and we assume that the first n' observations in the stream are coming from a single component without anomalies. Hence, with the first n' observations we form the first cluster by calculating the mean and the inverse of the covariance matrix. Then, we calculate ε as the distance between the farthest and the closest observation from the mean.

Time Complexity: In the initialization phase, the mean and the inverse of the covariance matrix of the first n' observations are calculated with complexity $O(n'd^2)$. The first step of the algorithm includes calculating the Mahalanobis distance of the new observation from all the clusters in the model in $O(|C|d^2)$ and updating the mean and the covariance matrix of the selected clusters (we assume all of the clusters) in $O(|C|d^2)$. This step applies to any observation that belongs to at least one cluster. Therefore, for a stream of size n, the total time complexity is $O(n|C|d^2)$.

In the second step, the time complexity of calling DBScan on the window (we assume the whole window) is $O(wl^2)$. At this point, since DBScan is called for each underlying component

(in the worst-case), we can rewrite the time complexity as $O(|C|wl^2)$. Calculating the mean and covariance matrix for each new cluster is $O(d^2)$, and we can rewrite it as $O(|C|d^2)$ for all components. As a result, the total time complexity of OnCAD is $O(n'd^2 + n|C|d^2 + |C|wl^2)$ in the worst-case. When the algorithm models all the underlying components, the time complexity for the rest of the stream is $O(n|C|d^2)$, which is linear in terms of the number of observations and underlying components.

This algorithm is implemented in Matlab[1] and publicly available[2].

4.4 Evaluation

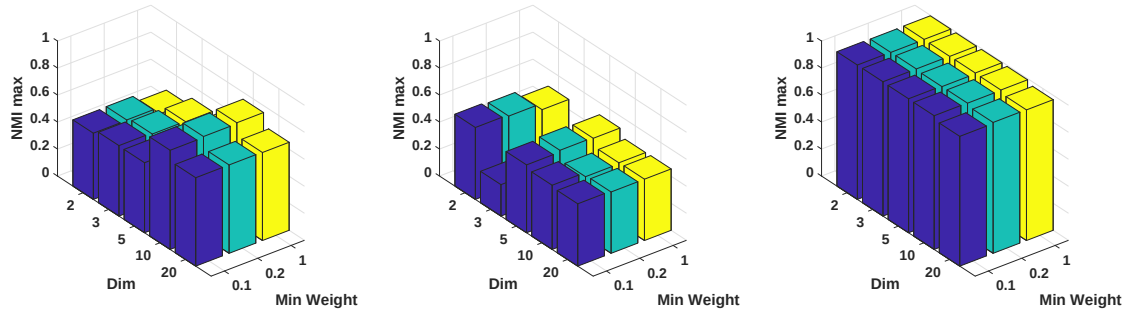
We have conducted an extensive experimental evaluation on both synthetic and real-world datasets. We generated streams of data in various numbers of dimensions in the range $[2,20]$, in both clean and noisy environments (with 5% noise) and various minimum mixture-weights $(0.1,0.2,0.5,1)$ for distributions. The size of the streams is at least 500,000 with a random number of underlying components in the range $[100,110]$.

For comparison, we used two state-of-the-art online clustering algorithms, i.e., online KMeans and the ART-2 network. We implemented online KMeans in MatLab and employed the ART-2 network implementation in [18]. We first present the parameter settings for each algorithm, then we present the measures that are employed for comparison, and finally, the results are presented.

Parameter Setting: Online KMeans takes a single parameter that determines the number of clusters in the data stream. While setting this number in static datasets is not trivial, in evolving streams it is even more challenging. Consequently, we set the KMeans parameter to the true number of clusters for each stream. When the vigilance parameter of ART-2 is set close to 1, ART-2 creates many small clusters (more vigilant), and when it is close to 0, it creates a few big clusters (less vigilant). In order to get the best performance of this algorithm, the first 500 observations of each data stream are from a randomly selected single component without anomalies. We start this algorithm with the value 0.999995 and reduce this parameter by the value 0.000003 until it forms a single cluster for the first 500 observations to calculate the optimal vigilance value.

Finally, OnCAD takes a parameter $\phi_{min} \in (0,1]$ that determines the minimum weight of a

[1] https://mathworks.com/ [2] https://github.com/mchenaghlou/OnCAD-PAKDD-2017



(a) Online KMeans (b) ART-2 Network (c) OnCAD

Figure 4.2: NMI_{max} values on clean synthetic data streams.

component in a mixture distribution that the algorithm is able to detect. Having ϕ_{min} set to a value close to 1 makes the algorithm a time-series clustering algorithm where there is one emerging cluster at any time. In contrast, setting ϕ_{min} to a value in the range $(0, 0.5]$ causes the algorithm to detect mixture distributions with various numbers of components. In our experiments, we set this parameter to 0.1 and recommend a value in the range $[0.1, 0.15]$ to be chosen for this parameter.

Performance Measure: To compare the performance of the different methods we used the Normalized Mutual Information (NMI_{max}) [84], which is a well-known information theoretic cluster validity index (higher values determine better clusterings), and computation time.

Results on Synthetic Datasets: The results on synthetic clean data streams over different data dimensions are presented in Figure 4.2. The results on noisy datasets are omitted as they are similar to clean datasets. Figure 4.2a illustrates the NMI_{max} value of the online KMeans in clean data streams, which is modest for all dimensions. The reason for this is that it generally converges to a local minimum by partitioning the input space into Voronoi cells. We observed that the noise in the stream does not affect the results greatly, but surprisingly the NMI_{max} value improves slightly by a few percent. The reason for this behavior is that the noise in the stream causes the cluster centers to be distributed better in the input space.

The results of the ART-2 network on clean data are illustrated in Figure 4.2b. This network normalizes the input points to unit length and employs hyper-spherical cluster prototypes. When the number of dimensions is 3, an abrupt drop of NMI_{max} value is observed. A close investigation revealed that when the number of dimensions is 3, the components are densely packed and it is difficult for ART-2 to characterize all of these densely packed components. In contrast, in other

Table 4.1: Anomaly detection rate on synthetic data

Algorithms	Online KMeans [93]	ART-2 net. [29]	Eff. method Chapter 3	Ens. model [122]	OnCAD
Sensitivity	0%	0%	86%	82%	**93**%
Specificity	**100**%	**100**%	99%	99%	98%
Accuracy	95%	95%	**98**%	**98**%	97%

numbers of dimensions the individual component distributions are more clearly separated and it is easier for them to be covered by ART-2. The behavior of this algorithm is similar in noisy environments. Finally, Figure 4.2c illustrates the results of OnCAD, where it outperforms the existing state-of-the-art algorithms. The main reason for this is that OnCAD identifies emerging clusters from anomalies in real-time and filters them out of subsequent clusterings.

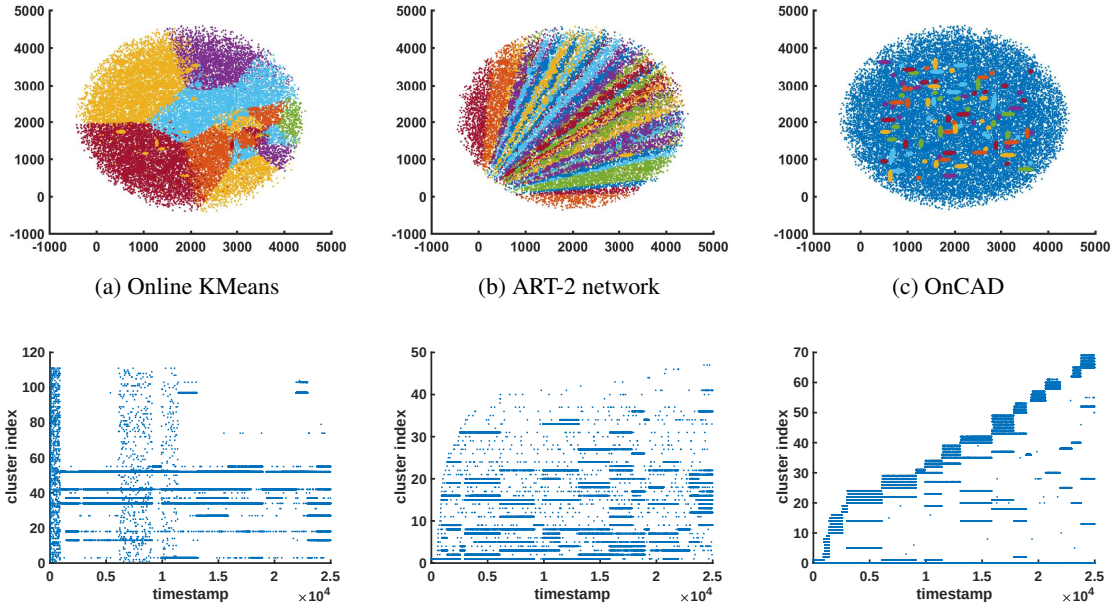
In order to demonstrate how real-time anomaly detection contributes to the results of OnCAD, in Table 4.1 we briefly compare this capability of OnCAD with an ensemble model anomaly detection algorithm [122] and the efficient anomaly detection algorithm in Chapter 3 on a randomly generated noisy stream. Online KMeans and ART-2 label all observations normal, achieving an accuracy of 95%. The methods in [122] and the efficient anomaly detection algorithm achieve the best accuracy of 98% (rounded to two decimal places) and OnCAD has the best anomaly detection rate of 93% with a slightly lower accuracy.

To elaborate the results in Table 4.1, the clustering results and the cluster membership figures of the algorithms are presented in Figure 4.3. The cluster membership figures (second row of Figure 4.3) show the evolution of clusters over time, where the x-axis is the timestamp of an observation and the y-axis is its cluster label. In these figures, we present the first 25,000 observations of the stream for better visualization.

The clustering of online KMeans (Figure 4.3a) shows the Voronoi cells created by this algorithm. Moreover, Figure 4.3d shows that this algorithm exhibits hardly any knowledge about the evolution of clusters. The ART-2 network performs poorly as well, as illustrated in Figure 4.3b since it cannot identify anomalies in the stream. Besides, the similarity measure that this algorithm employs is the dot product between the network pattern and the input point, therefore, maximizing this measure is equivalent to minimizing the angle between them. The cluster membership (Figure 4.3e) is not informative as this algorithm does not identify anomalies in real-time.

The results of OnCAD are presented in Figure 4.3c where anomalies are represented by blue points and clusters are represented by various colors. Since OnCAD identifies anomalies in real-

time, it detects emerging clusters, and from the cluster membership of OnCAD in Figure 4.3f, the time of emergence of each cluster is clearly apparent. Moreover, the times when a cluster becomes active and inactive can be determined.



(a) Online KMeans (b) ART-2 network (c) OnCAD

(d) Cluster membership of Online KMeans (e) Cluster membership of ART-2 network (f) Cluster membership of OnCAD

Figure 4.3: The clustering results and the evolution of clusters in synthetic data

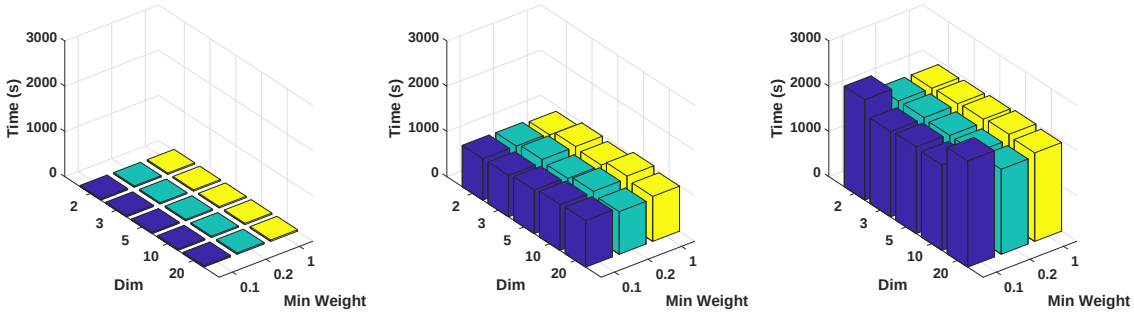
The second row of Figure 4.3 provides useful insights into how these algorithms work. For instance, the first few hundred observations in online KMeans (Figure 4.3d) are assigned to all cluster centers, but then a few clusters dominate the input space and the majority of observations are assigned to them. Figure 4.3e shows that the ART-2 network is unable to identify emerging clusters because of the noise in the stream. However, the results of OnCAD in Figure 4.3f demonstrate that OnCAD identifies emerging clusters by filtering out anomalies (anomalies are assigned to cluster 0).

The computation times of the algorithms on clean synthetic datasets are illustrated in Figure 4.4. Since the computation times on noisy datasets are similar to these figures, they are not included here. Figure 4.4a shows the computation times of the online KMeans, which is the fastest algorithm and takes about 30 seconds for each stream. Although online KMeans is fast, its anomaly detection performance is limited. The computation times of ART-2 are illustrated in

Table 4.2: NMI_{max} values on the gas sensor array dataset.

Algorithm Name	online KMeans	ART-2	OnCAD
NMI_{max}	0.47	0.79	**0.92**

Figure 4.4b where it generally takes about 15 to 20 minutes for each stream. Finally, the computation times of OnCAD are illustrated in Figure 4.4c. Although OnCAD is slower than the others, it models the evolution of clusters (second row of Figure 4.3) better than the others and has better clustering results, i.e., the NMI_{max} value of OnCAD is the highest compared to other algorithms.



(a) Online KMeans (b) ART-2 Network (c) OnCAD

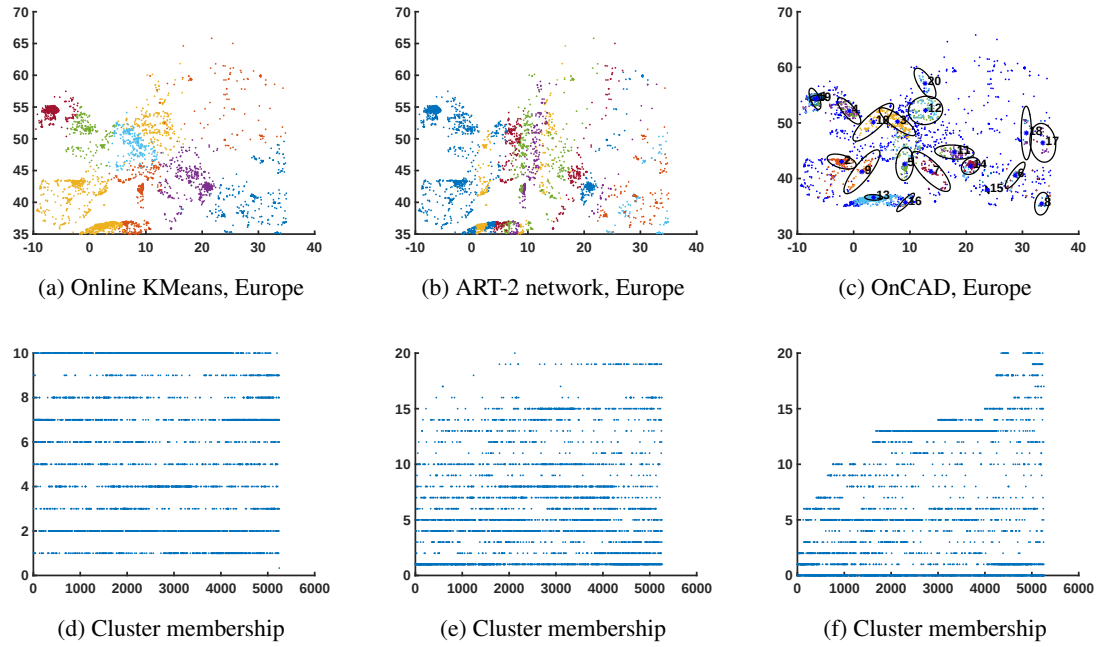
Figure 4.4: The computation times of clustering algorithms over clean synthetic datasets

Results on real-world datasets: For the real-world datasets, we used two publicly available datasets. The first one is the *gas sensor array under dynamic gas mixtures* [3] where we compare the clustering qualities in terms of NMI_{max} values. In this labeled dataset gas mixtures of varying concentration levels were exposed to 16 chemical sensors. We chose 12 of these 16 sensors (i.e., a 12-dimensional dataset) along with the data acquired from Ethylene and CO mixtures. After exposing the gas to the environment, it takes around 4 seconds for the sensors to capture the change. Therefore, we chose a subset of the data (consisting of more than 64,000 data points) after around 4 seconds of exposing a new mixture where it reaches an equilibrium, considering them as clusters. The NMI_{max} values on this dataset are presented in Table 4.2 where OnCAD outperforms the other algorithms. Note that since the clusters are well-separated in this real-world dataset, ART-2 outperforms online KMeans.

For the second real-world dataset we evaluate the capabilities of the algorithms to identify the

[3] https://goo.gl/zcAijP

temporal evolution of clusters on the *global terrorist attack* dataset [4] which contains the coordinates of the terror attacks around the world from 1970 through 2016. In this context, detecting emerging clusters from anomalies is crucial as an anomaly represents a single attack but an emerging cluster is a new group of attacks.



(a) Online KMeans, Europe (b) ART-2 network, Europe (c) OnCAD, Europe

(d) Cluster membership (e) Cluster membership (f) Cluster membership

Figure 4.5: The evolution of clusters in Europe on Global Terrorist Attack dataset

We chose Europe as the area of interest (latitude in range (35,70) and longitude in range (-10,35)) and the cluster membership of online KMeans are illustrated in Figures 4.5a and 4.5d. We set the parameter of this algorithm to 10 using the elbow method to determine the optimal number of clusters. This algorithm fails to show the evolution of the clusters in Figure 4.5d because it only considers the spatial proximity of observations.

The results of ART-2 are presented in Figure 4.5e. Since it does not identify anomalies in real-time, it is not clear when a new cluster emerges. As an example, clusters like 2, 9, 11, 15, 18, 19 are formed with a single observation (an anomaly), which is not effective. However, with OnCAD (Figure 4.5f) the emergence of the clusters can be determined as it filters out anomalies in real-time. For instance, clusters 11 (Hungary, Serbia, and Croatia), 12 (East Germany, west Poland) are formed later than clusters 1 (Ireland), 2 (north of Spain), 3 (West Germany and Netherlands),

[4] https://www.kaggle.com/START-UMD/gtd

and clusters 16 (North Africa), 17 (Ukraine) and 20 (Denmark) are formed towards the end of the stream. Meanwhile, some clusters become inactive towards the end of the stream such as cluster 2 (north of Spain) as there have been few terror attacks in that region in recent times.

4.5 Conclusion and Future Directions

In evolving data streams where patterns of data emerge, mature and evolve, identifying the temporal evolution of data and clusters can greatly help to understand the processes in the monitored system. In this chapter, we proposed an algorithm, named OnCAD, which performs online clustering and real-time anomaly detection on evolving data streams. OnCAD employs a sliding window that incrementally progresses over the stream and comprises two steps. In the first step, it checks the membership of a newly arrived data point to the set of already identified clusters and updates the relevant clusters. In the second step, when there is an emerging cluster and the data points in the window do not belong to any already defined clusters, it performs a batch clustering algorithm on the window to model the emerging cluster. However, the window length plays a major role, as if it is set to a small number it may not be able to capture any significant patterns in the stream, and if it is too large, it becomes more similar to batch clustering. We calculate the minimum window length by calculating a minimum sample size based on Elliptical Confidence Region where the mean of the sample is within a certain distance of the mean of the underlying distribution so that subsequent data points update the cluster and grow it to a more accurate representation of the underlying distribution.

A limitation of the current algorithm is that when the number of dimensions in a dataset increases, normal distance measures lose their accuracy. This results in a few problems that prevent the use of common clustering algorithms. OnCAD cannot be used for high-dimensional data because the covariance matrix of a cluster in a subspace of the ambient space is singular. Therefore, the inverse of the covariance matrix cannot be calculated. In the next chapter, we propose a subspace clustering algorithm for streaming time-series data. This algorithm uses the same incremental update formulae as in this chapter, but incrementally updates clusters formed in a subspace of the ambient space.

Chapter 5

An Incremental Subspace Clustering Algorithm for High-dimensional Time-series Data

In the previous chapters, we mainly addressed the challenges of online (incremental) clustering and identifying anomalies in real-time. However, analyzing high-dimensional data was not addressed. Subspace clustering aims to find patterns in a subspace of the ambient space in which the number of dimensions is high. Existing algorithms for subspace clustering mainly work on small datasets as they require $O(n^2)$ memory for a dataset of size n, and finding subspaces in streaming environments remains a challenge. Algorithms designed for unbounded streams alleviate this challenge by reducing the infinite search space of arbitrarily oriented subspaces into a bounded number of axis-parallel (or projected) subspaces. In this chapter, we propose the first arbitrarily oriented subspace clustering algorithm (SCTS) for time-series streams of unbounded length. SCTS is incremental (suitable for streams), and consumes substantially lower memory compared to state-of-the-art subspace clustering techniques. SCTS uses the temporal dependency of data points to identify emerging clusters in subspaces and overlapping clusters in dependent subspaces. We demonstrate the importance of these features in an intrusion detection scenario and its time complexity is linear with respect to the size of the stream, linear with respect to the ambient space dimensionality, and squared with respect to the number of dimensions of subspaces. Our synthetic as well as real-world experiments show that SCTS outperforms the current state-of-the-art subspace clustering algorithms in terms of clustering quality.

5.1 Introduction

THE performance of clustering algorithms often deteriorates as the number of dimensions increases [139]. Subspace clustering addresses the challenges in high-dimensional data as it has been shown that often a small set of features can accurately define underlying clusters in

highdimensional data [139]. Subspace clustering algorithms usually project (map) the data points into a subspace of the ambient space where distance measures work effectivly. The downside of the main subspace clustering algorithms [85, 132, 43, 65, 154, 88, 91] is that they work on small datasets.

In this chapter, we propose a novel subspace clustering technique for modeling arbitrarily oriented subspaces of streaming time-series data, which is an important subcategory of the data streams. In time-series, it is assumed that the data has a natural ordering in time, and observations close in time are closely related in value [101]. Time-series data have been widely studied as they are observed in many applications such as finance [134] and speech recognition [39]. Time-series are often assumed to be of fixed length, where certain features are extracted for classification or clustering. In contrast, streaming time-series data is generated continuously, and patterns in the data emerge and evolve over time.

As a motivating application, we take an example of time-series data from a computer network for intrusion detection throughout this chapter. In this scenario, certain features (such as protocol type, service, duration, number of data bytes from source to destination) of a computer network traffic data are monitored continuously to detect suspicious activities or intrusions forming a time-series. Normal daily network activities such as sending emails, web surfing or downloading files may form varying patterns (i.e., subspaces and clusters) in the time-series data. These patterns have a temporal aspect such that they dominate the stream for a period of time. For example in timespan A in Figure 5.1, the Subspace F1 dominates the stream. After a while, in timespan B, the subspace changes in orientation and Subspace F2 dominates the stream. This characteristic is different from batch datasets where data in memory without any ordering and can be shuffled randomly.

Identifying changes in subspaces and clusters is challenging. The first challenge is that since memory is limited, complex techniques [43], [88] cannot be applied. Second, it is possible that two phenomena are sufficiently similar to fall into a single subspace, but different enough to form two distinct clusters. We call them multi-cluster subspaces. This is represented in timespan C of Figure 5.1. In this timespan, the red cluster (the top cluster) in Subspace F1 has emerged which is attributed to an attack activity. Since this attack cluster resides in a subspace that has been already identified as normal, it may remain undetected, posing a threat to the security of the

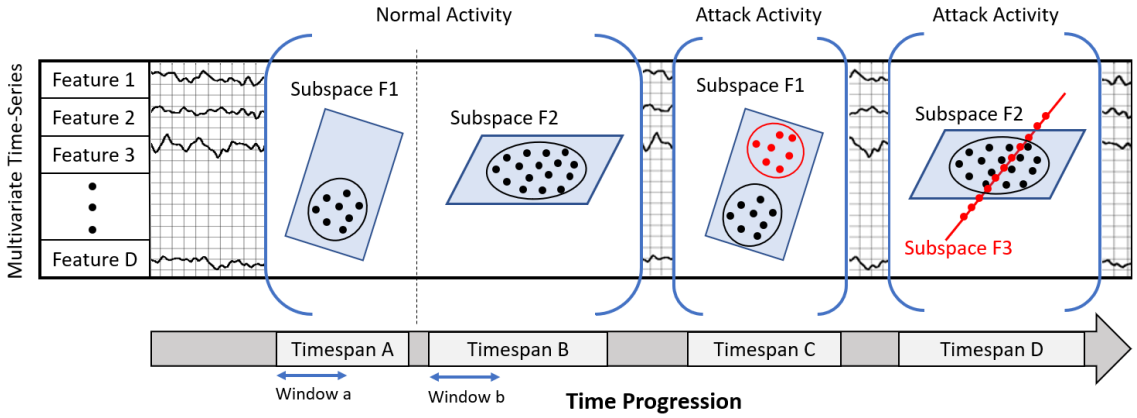


Figure 5.1: An illustration of the change of patterns in a high-dimensional multivariate time-series for intrusion detection. The data points in different timespans form different subspaces. In timespan A, data points form a 2-dimensional subspace. In timespan B the orientation of the subspace changes. In timespan C an attack cluster (the top cluster) emerges in a previously modelled subspace (an illustration a multi-cluster subspace). In timespan D an attack cluster (in Subspace F3) overlaps with the data points in a dependent Subspace F2 (an illustration of overlapping clusters in dependent subspaces)

network. Several algorithms such as [65], [154] are unable to identify them as separate clusters. The final challenge is identifying overlapping clusters in dependent subspaces. Two subspaces are dependent, if the union of their bases are linearly dependent. For example in timespan *D* of Figure 5.1, Subspaces F3 and F2 are dependent because the basis of Subspace F3 is a linear combination of elements in the basis of Subspace F2. The data points in F3 that are attributed to an attack activity overlapping with the data points in F2 that are attributed to normal activity, however, they have happened in two different timespans. This is particularly important because the aim of clustering is to identify clusters of similar data, and in this case, although data points are similar, they represent two different activities (normal versus attack) and are in two different time-frames and subspaces. Current state-of-the-art methods in subspace clustering [85, 43, 65, 154, 88] cannot identify this case because they ignore patterns that occur in temporal neighbourhoods.

In this chapter, we propose an incremental subspace clustering algorithm for streaming time-series data (SCTS), which processes data points one at a time, consumes low memory, can process infinite streams, and can identify the evolution of subspaces and clusters over time. SCTS uses a moving window to identify subspaces in local temporal neighbourhoods at any time (for detecting dependent subspaces), and updates the clusters in the subspace incrementally along with setting

a boundary for hyperellipsoidal cluster prototypes (to detect multi-cluster subspaces). The main contributions of this chapter are fourfold:

- SCTS to the best of our knowledge is the first arbitrarily oriented subspace clustering algorithm for unbounded time-series streams.
- SCTS identifies multi-cluster subspaces in the time-series data by setting a boundary for each cluster using hyperellipsoidal cluster prototypes.
- SCTS identifies overlapping clusters in dependent subspaces by defining subspaces in local temporal neighbourhoods in the stream on a moving window.
- SCTS proposes a new technique for detecting changes in subspaces in unbounded streaming time-series.

5.2 Related Work

Prior work on subspace clustering can be divided into five categories: statistical methods, algebraic methods, iterative methods, spectral clustering-based methods, and temporal and sequential methods.

Statistical Methods. Iterative statistical methods [133] are based on the assumption that the distribution of the data points in each subspace is Gaussian, and use Expectation Maximization to alternate between performing clustering and subspace approximation. The main drawback of these methods is that they generally need the number of dimensions of subspaces as an input parameter. Robust statistical approaches [46] randomly choose n data points and fit a subspace of dimension l' until the number of data points residing in the subspace is large. Then they remove those data points and repeat the process to find a second subspace, and so on. The main drawback of these methods is that besides knowing the number and dimensions of subspaces, the dimension of all subspaces must be equal.

Algebraic Methods. Algebraic methods that are based on matrix factorization [33, 74, 53], first create a similarity matrix by factorizing the data matrix, then find a segmentation by thresholding the entries in that matrix. These methods are sensitive to noise and when the subspaces are independent they are provably correct, however they fail when this assumption is violated.

Algebraic-geometric algorithms such as Generalized Principal Component Analysis (GPCA) [140, 92] fit the data with a polynomial. In this technique the normal vector of the subspace at a point is given by its gradient at that point. While these methods can identify subspaces of various dimensions, they are sensitive to anomalies and their complexity increases exponentially by the number and dimensions of the subspaces. These shortcomings make these methods unsuitable for streaming environments.

Iterative Methods. Iterative methods [65, 154, 135] alternate between assigning data points to subspaces and fitting a subspace to each cluster. Although these methods are fast, they require the number and dimensions of subspaces as an input parameter which are often unknown a priori. These methods can process larger datasets than the algebraic methods, but they require all the data to be stored in memory in order to have random access to arbitrary data points.

Spectral Methods. Spectral clustering methods first create a similarity matrix for the data points, then apply spectral clustering on the matrix to find clusters. Therefore, building the similarity matrix is at the heart of these methods. Algorithms such as Sparse Subspace Clustering (SSC) [43] and Low-Rank Recovery (LRR) [88] use global information for creating the similarity matrix and aim at finding a sparse or low-rank representation for the data points. The main logic in these methods is that the sparsest or lowest-rank representation of each data point is a basis for the subspace in which the data point resides. The main drawback of spectral methods is that building the dissimilarity matrix is time-consuming and requires $O(n^2)$ memory where n is the dataset size.

Temporal and Sequential Methods. This category is the closest one to our problem. *Ordered Subspace Clustering* (OSC) [132] is similar to SSC at finding a sparse representation for data points, but uses a similarity measure that directly enforces sequential similarity. Although this method is for sequential data, it requires $O(n^2)$ memory (n is the dataset size) making it unsuitable for streaming environments. Temporal Subspace Clustering (TSC) [85] takes a slightly different approach by designing a temporal Laplacian regularization for encoding the sequential relationships in time-series data. Then, they learn a non-negative dictionary from data to obtain expressive codings. These codings are used to build an affinity graph, where spectral clustering algorithms can be used for segmentation. The main drawback of this method is its memory requirement ($O(n^2)$ for n data points) which makes it infeasible for streaming time-series data.

5.3 Problem Definition

In this section, we formally define the subspace clustering of streaming time-series data. Let $X = \{x_1, x_2, x_3, \ldots, x_n, \ldots\}$ be a d-dimensional time-series where each $x_i \in X$ is a feature vector $(x_1, x_2, \ldots, x_d)^T \in \mathbb{R}^d$. Suppose $x_i s$ are coming from a set of k unknown clusters denoted by $C = \{c_1, c_2, \ldots, c_k\}$. These clusters lie in l' unknown linear or affine subspaces of the ambient space denoted by $B = \{b_1, b_2, \ldots, b_{l'}\}$. This is in contrast with the problem definition in the past two chapters, where the distributions resided in the ambient space. For example, in the past two chapters in a 3-dimensional dataset, data were drawn from a 3-dimensional distribution, whereas in this chapter, the dimensionality of the distributions could be much lower than the D-dimensional ambient space Moreover, it is possible that some of the subspaces in B are multi-cluster subspaces, or there exists some overlapping clusters in dependent subspaces. Each subspace $b_i \in B$ has an unknown dimension $\tilde{d}_i = \dim(b_i)$, $0 < \tilde{d}_i < d$, and can be described as

$$b_i = \{x \in \mathbb{R}^d : x = \rho_i + \Delta_i y\}, \;\; i = 1, \ldots, l'$$

where x is a random observation in X, $\rho_i \in \mathbb{R}^d$ is the affinity vector (it can be set to $\mathbf{0}$ for linear subspaces), $\Delta_i \in \mathbb{R}^{\tilde{d}_i \times d}$ is a basis for b_i and y is a low-dimensional representation for x.

At each timespan of the time-series, data points come from a single underlying cluster in C, which means at any point in time a single cluster contributes to the stream. The underlying cluster that contributes to the stream changes over time and it is possible that either the new cluster is from the same subspace or another subspace in B. This process continues indefinitely and the aim is to identify the clusters in C, label the incoming data points in an online manner, and identify the subspaces in B by calculating a basis for them. Data points become available one at a time in a sliding window of length wl, where wl is a small integer as the available memory for storing the data is limited. When a new data point arrives and the sliding window is full, the data point at the leftmost cell is discarded and the new data point is put in the rightmost cell.

5.4 SCTS Algorithm

Our SCTS algorithm is based on identifying subspaces and clusters in local temporal neighbourhoods, and uses PCA on the sliding window. Applying PCA on the whole dataset as a single entity results in extracting the most important features for all clusters in the data. In contrast, applying PCA on a moving window generates the most important features for each cluster that contributes to the data stream. SCTS identifies the times at which each cluster is "active" (i.e., contributes to the stream) in a subspace, and also detects the changes from one cluster to another cluster (possibly in another subspace) in the stream. Since SCTS is an incremental algorithm, we present one iteration of it, then discuss its initialization.

Assume SCTS has processed n data points, and x_{n+1} has newly arrived. Also assume that after processing n data points the algorithm has identified $l'' \leq l'$ subspaces $B' = \{b'_1, b'_2, \ldots, b'_k\}$ with a corresponding set of clusters in each subspace $C'_i = \{c'_{i,1}, c'_{i,2}, \ldots, c'_{i,|C'_i|}\}$ for a subspace $b'_i \in B'$. When x_{n+1} arrives, the following questions should be answered:

1. Does x_{n+1} belong to the active subspace (i.e., the subspace that currently contributes to the stream)? Does it belong to any of its clusters?

2. If x_{n+1} belongs to the active subspace but does not belong to any of its clusters, is it an anomaly or an emerging cluster?

3. If x_{n+1} does not belong to the active subspace, is it an anomaly or an emerging subspace?

Answering these questions is non-trivial since the active clusters and subspaces change over time. In order to answer these questions for each data point, SCTS allocates an array of length wl named R and iteratively updates this array as the window slides. This array helps SCTS to remember which questions have been answered for data points in the sliding window. SCTS answers these questions in two steps.

Step 1: This step processes x_{n+1} as it arrives and answers question 1. SCTS calculates the subspace in the new window and compares it with the active subspace. The information of the active subspace is stored in a variable called space-tracker. For instance, in timespan A of Figure 5.1, the space-tracker stores the information of Subspace F1. The subspaces in subsequent windows are compared with the space-tracker for detecting changes in subspaces. If a change is detected (such

as the transition from timespan A to B in Figure 5.1), it associates x_{n+1} with the value 2 and stores it in the array R (i.e., $R_{x_{n+1}} \leftarrow 2$). The value 2 signifies that x_{n+1} does not belong to the current active subspace. Otherwise, it is mapped into the active subspace and the membership of x_{n+1} with the clusters in it is determined. If x_{n+1} belongs to any clusters in the subspace, the cluster is updated and $R_{x_{n+1}} \leftarrow 0$ showing that it has been absorbed by a cluster. Otherwise, $R_{x_{n+1}} \leftarrow 1$ which shows that the data point is in the active subspace, but does not belong to any of the clusters in it. Answering questions 2 and 3 for x_{n+1} is impossible at this stage, so we postpone them until x_{n+1} reaches the leftmost (first) cell in the window.

Step 2: This step is for x_{n-wl+1} (the data point in the leftmost cell in the window) and we answer questions 2 and 3 for it. The goal here is to identify emerging clusters and subspaces from anomalies. It is now possible with the information provided by the data points ahead of x_{n-wl+1} in the window. Since this is time-series data, only a single cluster contributes to the stream at each timespan. Therefore, based on the values stored in array R, it is checked if an inactive cluster has become active again, or a new cluster or subspace has emerged, or x_{n-wl+1} is an anomaly. Identifying anomalies and reporting them is not the main focus of this chapter, but an anomaly stays in the window for wl iterations and may change the subspace of the window in this period. To mitigate this effect, and make SCTS robust against anomalies, a new subspace is added only when a new subspace in the window remains for at least $2 \times wl$ iterations. Moreover, it is possible that during changes between active clusters in different subspaces, a mix of 0s, 1s and 2s appear in R. In this case, a new subspace or cluster is not added since there is no significant reason for defining one. More specifically, the window length wl plays a major role in effectively filtering anomalies and identifying emerging patterns. If the window length is too large, it is possible that several clusters contribute to the stream, and if it is too small it may not capture any statistically significant patterns. We propose a method for automatically calculating the minimum window length in Section 5.4.3 that enables significant evidence to be captured of defining a new cluster. These two steps answer all questions for each data point that arrives in the window, and in the following section, we explain these steps in more detail.

5.4.1 SCTS in Detail

The aim of SCTS is to identify subspaces and clusters in local temporal neighbourhoods of data points. SCTS is capable of identifying affine subspaces by taking a random data point in the subspace as an affine vector. An affine subspace is a linear subspace that is shifted from the origin. Therefore rather than adding another dimension to convert it into a linear subspace, we shift the data points by the affine vector to shift it back into a linear subspace.

SCTS identifies subspaces using PCA which provides the *Principal Components* (PCs) for a set of data points. However, rather than extracting the PCs for the whole data, PCs are extracted in each window. If efficiency is an issue, Moving Window PCA (MWPCA) [143] can be employed which is an efficient way of extracting PCs on a moving window. When a new data point arrives, it cancels the effect of the last data point in the window from the PCs and then updates them with the newly arrived data point. The PCs and the affine vector of the active subspace are stored in the space-tracker. SCTS also retains an array for storing 0-dimensional subspaces. These subspaces occur when all the data points in the window are identical, and are stored in a different array as a special case that may happen.

Algorithm 4 shows how SCTS works. When x_{n+1} arrives, it is first transformed into the linear subspace by the affine vector stored in the space-tracker. Then MWPCA is applied on the new window in line 10, and the PCs that cover $\beta\%$ of the variance in the data are chosen in line 11 (β is provided by the user as a parameter). In line 12 of Algorithm 4, the angle between the subspace in the space-tracker and the subspace in the window is calculated. The angle between subspaces is defined based on the angles between the vectors spanned by their bases. We employ the definition in manuscript [144] (section 3) which proposes a robust method for calculating angles between subspaces with various dimensions. In this definition, the angle between b_1 and b_2 is defined as $\min(\tau_1, \tau_2)$ where

$$\tau_1 = \sup_{x \in b_1} \measuredangle(x, b_2) \quad \text{and} \quad \tau_2 = \sup_{x \in b_2} \measuredangle(x, b_1)$$

In these equations, x is a vector in subspaces b_1 and b_2 (in our case, a data point in these subspaces), and *sup* is the mathematical set operator and is defined as if Q is a subset of P, then the $sup(Q,S)$ is the smallest element in Q that is greater than or equal to all elements in S. The calculation of these angles follows the derivation in reference [144] for a detailed description. Intuitively,

Algorithm 4 One iteration of SCTS

1: **Inputs**
2: W current window
3: B′ set of $l'' \leq l'$ detected subspaces
4: C′ set of identified clusters in C
5: st space-tracker
6: τ the angle threshold parameter
7: β the variance coverage parameter
8: **procedure** SCTS(W, B′, C′, st, τ, β)
▷ transform x_{n+1} into the linear subspace
9: $x'_{n+1} \leftarrow x_{n+1} - st$.AffineVector
▷ compute the new principal components (PCs) and weights (ω) using MWPCA [143]
10: (PC, ω) ← MWPCA($x'_{n-wl+1}, \ldots, x'_{n+1}$)
▷ extract the principal components that cover β% of the variance in ω in the data and store it in B_W (a basis for the current window)
11: B_W = extBetaPrcntPC(PC, ω, β)
▷ calculate the angle between the subspace in the window and the subspace of the space-tracker using [144]
12: angle = calculateAngle(B_W, B_{st})
▷ calculate the number of dimensions
13: $\tilde{d_w} \leftarrow \dim(B_W)$; $\tilde{d_{st}} \leftarrow \dim(B_{st})$
▷ Perform Step 1 operations
14: STS-Step1(x_{n+1}, B′, C′, *st*, $\tilde{d_w}$, $\tilde{d_{st}}$, angle, τ)
▷ Perform Step 2 Operations
15: STS-Step2(W, B′ , C′, st, B_W)
16: **end procedure**

the angle between two 1-dimensional subspaces is effectively calculated as the angle between two lines. Similarly, the angle between two 2-dimensional subspaces is calculated as the angle between two planes. This definition allows the calculation of the angle between two 3-dimensional (or higher) subspaces. It also allows the calculation of the angle between two subspaces with different numbers of dimensions. For instance, the angle between a 1-dimensional subspace and a 2-dimensional subspace is the angle between a line and a plane. This also generalizes to higher numbers of dimensions. Line 10 calculates the dimensions of these two subspaces. Using these calculations, SCTS answers question 1 for x_{n+1} in step 1.

Step 1 operations:

In step 1, the clusters in subspaces are updated and the changes in subspaces are identified. A change in subspace can occur in the orientation, the number of dimensions, or both. Based on the number of dimensions and the angle between the active subspace (stored in space-tracker) and the subspace in the window, four scenarios can occur: (i) the number of dimensions is the same and the angle between them is less than a small threshold τ (the active subspace and the window are the same), (ii) the number of dimensions is the same but the angle is greater than τ (the active subspace has changed in orientation), (iii) the number of dimensions is different and the angle is less than τ (the active subspace and the subspace in the window are dependent), or (iv) the number of dimensions is different and the angle is greater than τ (the active subspace has changed both in orientation and the number of dimensions). We first discuss each scenario in detail, then elaborate on calculating the memberships and the cluster updates:

Scenario (i): *The number of dimensions of the two subspaces are equal and the angle between them is less than the threshold* τ. In this case, the subspace of the current window is the same as the active subspace, and the size of the active subspace in the space-tracker is increased by one. If x_{n+1} belongs to a cluster in this subspaces, the relevant clusters in the subspace are updated and $R_{x_{n+1}} \leftarrow 0$. The value 0 determines that x_{n+1} has been absorbed by a cluster in the subspace. Otherwise, $R_{x_{n+1}} \leftarrow 1$, showing that x_{n+1} belongs to the subspace but it is not in any clusters.

Scenario (ii): *The number of dimensions is the same, but the angle between them is greater than* τ. In this case, the dimension of the two subspaces are the same but they are differently oriented. Therefore $R_{x_{n+1}} \leftarrow 2$ which indicates that x_{n+1} does not belong to the active subspace.

Scenario (iii): *The number of dimensions is different, but the angle between them is less than* τ. This case happens when observing dependent subspaces, where the angle between the two subspaces is low but one of them has a lower dimension. In this case, $R_{x_{n+1}} \leftarrow 2$ which indicates that x_{n+1} does not belong to the active subspace.

Scenario (iv): *The number of dimensions is different, and the angle between them is greater than* τ. This case happens when the current window is in a totally different subspace than the space-tracker. Therefore, $R_{x_{n+1}} \leftarrow 2$ which indicates that x_{n+1} does not belong to the active subspace.

Cluster prototypes and memberships. To update the clusters in the active subspace (C_{act})

using x_{n+1}, we use a Gaussian representation with hyperellipsoidal cluster prototypes. Each cluster $c_i \in C_{act}$ is represented by a mean and the inverse of its covariance matrix in the subspace. We determine the membership of x_{n+1} to clusters in the active subspace by the Mahalanobis distance:

$$(x_{n+1} - m)^T S^{-1} (x_{n+1} - m) \leq z^2. \tag{5.1}$$

In this formula, m is the center of c_i (a hyperellipsoid) at time n, $S_{n,i}^{-1}$ is the inverse of covariance matrix of c_i at time n, and z^2 is a constant that marks a specific level set of the cluster. When z^2 is chosen from the cumulative inverted chi-squared distribution with d degrees of freedom for a probability value P, $(\chi_d^2)_P^{-1}$ sets the boundary of the cluster such that the probability of an observation falling outside the boundary is $1-P$. In our experiments, we set the value of P to 0.99.

If the projection of x_{n+1} in the subspace falls inside a cluster in it, we set $R_{x_{n+1}} \leftarrow 0$. Assume that the projection falls inside a subset of clusters $C_{member} \subseteq C_{act}$. Based on the Mahalanobis distance of x_{n+1} to the clusters in C_{member}, a weight is attached to x_{n+1}. These weights that are denoted by $\psi_{n+1,i}$, $1 \leq i \leq |C_{member}|$, are the normalized degrees of membership of x_{n+1} to the clusters in C_{member}. Then, we update the mean and the inverse covariance by the following weighted incremental update formulae proposed in [100]:

$$m_{n+1,i} = m_{n+1,i} + \frac{\psi_{n+1,i}}{\Sigma_{j=1}^{n+1}} (x_{n+1} - m_{n,i})$$

$$S_{n+1,i}^{-1} = \chi_{n,i} [S_{n+1,i}^{-1} - \frac{S_{n,i}^{-1} (x_{n+1} - m_{n,i})(x_{n+1} - m_{n,i})^T S_{n,i}^{-1}}{\delta_{n,i} + (x_{n+1} - m_{n,i})^T S_{n,i}^{-1} (x_{n+1} - m_{n,i})}]$$

In the above equations

$$\chi_{n,i} = \frac{\kappa_{n,i} (\kappa_{n,i+1}^2 - \tau_{n+1,i})}{\kappa_{n+1,i} (\kappa_{n,i}^2 - \tau_{n,i})}$$

$$\delta_{n,i} = \frac{\kappa_{n+1,i} (\kappa_{n,i}^2 - \tau_{n,i})}{\kappa_{n,i} \psi_{n+1,i} (\kappa_{n+1,i} + \psi_{n+1,i} - 2)}$$

where $\kappa_{n,i} = \Sigma_{j=1}^{n} \psi_{j,i}$ and $\tau_{n,i} = \Sigma_{j=1}^{n} \psi_{j,i}^2$. Now that array R is updated based on the computations in Step 1, the emerging clusters or subspaces can be identified using Step 2.

Step 2 operations:

In the second step, SCTS identifies multi-cluster subspaces and detects emerging subspaces in the stream. Identifying multi-cluster subspaces involves identifying an emerging cluster in a previously identified subspace. On the other hand, detecting emerging subspaces involves comparing the new window subspace with all identified subspaces in S, and adding a new subspace if needed.

These tasks are performed by tracking the numbers associated with each data point in array R. After step 1, before discarding x_{n-wl+1}, $R_{x_{n-wl+1}}$ is checked. If $R_{x_{n-wl+1}}$ is 0, it shows that it has been a member of a cluster, so we safely discard it and proceed to the next iteration. If $R_{x_{n-wl+1}}$ is 1, it shows that x_{n-wl+1} is in the current subspace, but not a member of any clusters in it. Therefore, if all the data points in the window are associated with 1, it shows that a new cluster has emerged in the current subspace. Thus, a new cluster is defined for it. In this case, a multi-cluster subspace is identified.

On the other hand, if $R_{x_{n-wl+1}}$ is 2, it shows that x_{n-wl+1} is in another subspace, therefore we check if all the data points in the window are associated with the number 2. If that is the case, it shows that the subspace has changed. Before adding the current window subspace to B, it is compared with the subspaces stored in B. This check is performed to determine if a previously modelled subspace has become active again. If the subspace in the window is similar to one of the known subspaces (having the same dimension and an angle less than the threshold τ), space-tracker is updated with the identified subspace in B. Otherwise, a new subspace is defined for it by modelling its clusters. Then space-tracker is updated with the newly found subspace.

As the window slides over the stream, it is possible that anomalies alter the subspace as the window passes over them. An anomaly is present in the window for wl iterations and may alter the subspace in the space-tracker. For instance if the subspace in the space-tracker is 2-dimensional, an anomaly in the ambient space may create a 3-dimensional subspace in the window. In order to be robust against anomalies, the newly formed space-tracker remains a candidate subspace until the number of data points in that subspace reaches a threshold. In our experiments we set this threshold to $2 \times wl$. The reason for choosing $2 \times wl$ is that we wait for at least two window lengths to make sure the subspace in the window is not formed by an anomaly. Here, one iteration of the algorithm terminates and SCTS proceeds to the next iteration.

5.4.2 Initialization

At the beginning of the stream, we take a random data point in the first window of data points as the affine vector and transform all the data points in the window to a linear subspace. Then a batch PCA is performed on the window and the PCs that cover $\beta\%$ of the variance in the data are picked as the basis Δ for the initial window. The low-dimensional representation for all of the data points in the window is calculated by $y = \Delta^T \times (x - \rho)$ where y is the low-dimensional representation of the data point x in the window, Δ^T is the transpose of the calculated basis, and ρ is the randomly selected data point as the affine vector. The calculated subspace is added to B' and stored in the space-tracker. Finally, the first cluster is formed by calculating the mean and the inverse of the covariance matrix in the subspace.

This algorithm is implemented in Matlab[1] and publicly available[2].

5.4.3 Calculating Window Length

We set the window length automatically using the approach in Chapter 4, which calculates the minimum window length with an *Elliptical Confidence Region* (ECR). With ECR, we calculate the minimum sample size for modelling an underlying cluster up to a specific accuracy with a high confidence. This sample size is calculated such that the mean of the sample is within 50% of the data points in the underlying Gaussian distribution with 95% confidence. The reason of choosing 50% is that on one hand, we want to have a small sample size, and on the other hand we want a single cluster for each short timespan. Therefore, we choose 50% so that the ECR is only small enough (the sample size is large enough) such that at least half of the subsequent data points from the underlying cluster fall inside the cluster formed by the sample. Over time when new data points from the same cluster appear and are absorbed by the cluster, the mean and covariance of it become a more accurate representative of the underlying cluster. For more details regarding the window length see Chapter 4.

[1] https://mathworks.com/ [2] https://github.com/mchenaghlou/SCTS-Github

5.4.4 Computational Complexity

At each iteration, PCA is applied on a window of length wl with the time complexity of $O(min(wl \times d^2, wl^2 \times d))$. The basis of the current window that covers $\beta\%$ of the variances is extracted in O(d). The angle between subspaces is calculated in O($d \times \tilde{d}^2$) and updating the clusters in the subspace is calculated in O($k \times \tilde{d}^2$). In the second step, identifying emerging clusters and subspaces are calculated in O($l' \times wl \times \tilde{d}^2$). Therefore, the total time complexity of SCTS is O($n \times (min(wl \times d^2, wl^2 \times d) + d + d \times \tilde{d}^2) + l' \times wl \times d^2 + k \times \tilde{d}^2$) which can be simplified as O($n \times min(wl \times d^2, wl^2 \times d) + n \times d \times \tilde{d}^2 + (l' \times wl + k) \times \tilde{d}^2$). This time complexity shows that the algorithm is linear with respect to the size of the stream, linear with respect to the ambient space dimensionality, and squared with respect to the number of dimensions of subspaces. Note that subspaces have a notably lower number of dimensions compared to the ambient space, which makes SCTS an efficient algorithm when the dimensionality of subspaces is small.

Now we analyse the memory complexity of SCTS. A window and an array of length wl requires O($2 \times wl$) = O(wl) memory for processing data points in the stream. For a subspace with $\tilde{d}$ dimensions, we store its PCs as a $d \times \tilde{d}$ matrix, and for each cluster, we store the mean and the inverse of the covariance matrix in the subspace in O($l' \times d \times \tilde{d} + k \times (\tilde{d} + d \times \tilde{d})$). Therefore the total memory complexity of SCTS is O($wl + (l' + k) \times \tilde{d} \times d$). This shows that the memory complexity of SCTS does not depend on the number of data points, but on the number of underlying clusters and subspaces, and their dimensionality.

5.5 Evaluation

In this section, we compare SCTS with several conventional algorithms in the subspace clustering literature. They include SSC [43], LSR [91] and LRR [88] from spectral clustering-based methods, OSC [132] and TSC [85] from temporal and sequential methods, and k-flat [154] and k-subsequence [65] from iterative methods. The implementations of SSC, LSR, LRR, OSC and TSC are obtained from [132][3], and the implementations of kflat, and k-subspaces are obtained from JHU Vision Lab[4]. All the results, Matlab code, datasets, and experiments are publicly available online[5].

[3] https://github.com/sjtrny/SubKit [4] http://vision.jhu.edu/code/ [5] https://github.com/mchenaghlou/SCTS-Github/

Comparison Methods: We propose a three-phase experimental design to provide a systematic performance evaluation. First, we use small datasets to include algorithms such as LRR, SSC, OSC and TSC for comparison due to their large time and memory requirements. LRR, SSC, OSC, TSC, kflat and k-subspaces require O(n^2) memory where n is the dataset size. In the second phase, the evaluation is performed on relatively large datasets where we employ k-flat and ksubspaces, which are capable of processing larger datasets as a comparison. Synthetic datasets are generated to mimic the process in our motivating example (Figure 5.1) where in each timespan a cluster contributes to the stream. Therefore, clusters become active one after another and it is possible that a previous active cluster becomes active again.

In the third phase, we compare the performance of the algorithms on a real-life dataset KDD-CUP'99[6] that is for intrusion detection. In this dataset, underlying clusters contribute to the stream in different time-frames. For instance, for a period, a normal cluster contributes to the stream. Then, an attack cluster emerges, and then, another normal cluster contributes to the stream and so on. We show how identifying multi-cluster and dependent subspaces in this dataset enhances the results of SCTS in terms of separating attack data points from normal data points.

Evaluation Metrics: The metrics that we use for comparing the clustering results are three well-known cluster validity indices. First, we use a transparent and simple index known as *purity* to demonstrate how pure clusters are. The second measure we use has information theoretic interpretation known as *Normalized Mutual Information* NMI_{max} that reflects the clustering quality in terms of the reduction in entropy of class labels when we observe the clustering result. The third metric is a well-known set-matching measure known as *Adjusted Rand Index* (ARI), which shows the similarity of data points in clusters. The range of these indices are in [0, 1] where a higher value represents a better clustering.

Parameter Setting: SCTS requires two parameters. The first parameter (τ) is the angle threshold for detecting changes of orientation in subspaces. We set τ to 0.2 rad (about 10 degrees) to ensure the algorithm is sensitive to small changes in subspaces. If this parameter is set to a lower value close to 0, it becomes more sensitive to changes in the subspaces. This can be done when it is desired to identify the slight changes in the subspaces. In contrast, higher values for this parameter make SCTS resistant to changes in the subspaces and only major changes in the subspaces

[6] http://kdd.ics.uci.edu/databases/kddcup99/

will be identified. Second, β, which determines the percentage of the variances of the principal components that should be covered for choosing the principal components, is set to 99% to ensure the maximum variance is covered. This should not be confused with the variance threshold that is used for batch PCA. In batch PCA, the main features of the whole dataset are extracted, however, SCTS performs PCA on a small window to extract the main features of each cluster in the stream. We set this threshold to 99% to ensure that the maximum variance of the features for each cluster is extracted. Having lower values for this parameter makes SCTS choose fewer principal components as the subspace, which in turn may result in a loss of information in the data. In contrast, higher values for this parameter will add more principal components (dimensions) to the subspace, making it a potentially high-dimensional subspace. For the window length, instead of setting it to a fixed number, we automatically calculate it. The automatic calculation of the window size requires having a confidence score, which we set to 95%, and the number of dimensions in the data. Since we perform subspace clustering where the dimensions of the subspaces are unknown, we set this number to 15. This means that we assume the maximum number of dimensions of a subspace $b_i \in B$ is 15. This can be compared with other methods such as k-flat and k-subspaces that require the number of subspaces and their dimensionalities as input parameters, whereas SCTS only requires an upper bound for the dimensionalities of subspaces (and not the number of subspaces). Moreover, this number should not be confused with the dimensionality of the ambient space, which could be thousands of dimensions. Accordingly, the window length is calculated as 22 (as explained in Section 5.4.3 in all of the experiments. This window length is only sufficiently large to form an initial cluster that covers half of the underlying cluster. This initial sample cluster should cover at least half of the underlying cluster, so at least half of the subsequent data points fall inside this sample cluster to update it to become a more accurate representative of the underlying cluster. The reason that we chose the maximum number of dimensions for subspaces to be 15 is that many phenomena can be explained with a few features, and we intend for SCTS to be able to identify phenomena that require more than a few features to be explained. Moreover, having large values for this parameter does not affect the performance adversely (i.e., SCTS does not look for subspaces with 15 dimensions), it only results in a slightly larger window.

Table 5.1: Clustering results of Phase 1

	Multi-cluster subspaces			Overlapping clusters		
	NMI_{max}	ARI	Purity	NMI_{max}	ARI	Purity
LRR	0.86	0.76	0.88	0.58	0.47	0.62
SSC	0.85	0.75	0.88	0.73	0.66	0.78
LSR	0.86	0.75	0.88	0.58	0.47	0.62
OSC	0.98	**0.98**	**0.99**	0.58	0.36	0.59
TSC	**0.99**	**0.98**	**0.99**	0.57	0.39	0.59
k-flat	0.25	0.13	0.46	0.13	0.05	0.31
k-subs	0.63	0.03	0.67	0.03	0.00	0.22
SCTS	0.89	0.93	0.96	**0.85**	**0.88**	**0.96**

Phase 1: Small Synthetic Datasets

We propose two kinds of datasets for this phase to provide an accurate performance evaluation. First, we evaluate the algorithms on identifying multi-cluster subspaces. Second, we evaluate their performance in identifying overlapping clusters in dependent subspaces. These datasets comprise 1000 dimensions and their sizes are around 1800 data points. Since the subspaces generally have much fewer dimensions than the ambient space, we randomly selected the dimensions of the subspaces (in range [1, 10]) to be 4, 5 and 8, where each subspace contains two clusters of size around 300. The datasets for dependent subspaces comprise two main subspaces of dimensions 6 and 9 where each of these subspaces has dependent subspaces of lower dimensions. More specifically, the 6-dimensional subspace has two dependent subspaces of dimensions 3 and 1, and the other subspace of dimension 9 has one dependent subspace of dimension 5. All the clusters in dependent subspaces overlap.

Phase 1 results: The results of Phase 1 are presented Table 5.1. For the multi-cluster case, the temporal subspace clustering methods (TSC and OSC) have the best performance because they are designed to identify patterns that are temporally related. SCTS has the second best performance with 89%, 93% and 96% for NMImax, ARI and purity measures, respectively. Spectral clustering-based methods such as LRR, SSC and LSR have a slightly lower performance because they do not exploit the temporal relationship of patterns in the stream.

For overlapping clusters in dependent subspaces, the performance of all methods (except SCTS) notably deteriorates, which demonstrates that they cannot identify overlapping clusters in dependent subspaces. For SCTS the performance is 0.85, 0.88 and 0.96 for NMI_{max}, ARI and

Table 5.2: Clustering results of Phase 2

	Multi-cluster subspaces			Overlapping clusters		
	NMI_{max}	ARI	Purity	NMI_{max}	ARI	Purity
k-flat	0.02	0.00	0.03	0.01	0.00	0.04
k-subs	0.74	0.48	0.42	0.67	0.19	0.60
SCTS	**0.89**	**0.61**	**0.88**	**0.87**	**0.81**	**0.94**

purity measures, respectively, which is an indication of correctly identifying overlapping clusters in dependent subspaces. As a conclusion, spectral clustering-based and temporal subspace clustering methods are unable to identify overlapping clusters in dependent subspaces, but they can effectively identify multi-cluster subspaces. The main drawback of these methods is that they cannot process large datasets, therefore we exclude them for the rest of the experiments and we consider iterative methods in our experiments.

Phase 2: Large Synthetic Datasets

We now test k-flats and k-subspaces on relatively larger datasets. We already have demonstrated that these methods cannot identify multi-cluster and dependent subspaces, so our aim is to present the proposed method's performance in these environments. For the multi-cluster case, 20 subspaces with dimensions less than 10 are randomly selected where each subspace contains two clusters with 1000 data points. Therefore there is a transition from one cluster to another after every 1000 iterations. The datasets for dependent subspaces comprise 20 subspaces of randomly selected dimensions in the range [1, 11]. Each subspace has an overlapping cluster in a dependent subspace where the dimensions of the dependent subspaces have been randomly selected. The dimension of the dependent subspaces is selected to be less than the dimension of the original subspace.

Phase 2 results: The results of the algorithms are presented in Table 5.2. SCTS notably outperforms the other algorithms. The reason for this high performance is that subspaces and clusters are identified in their local temporal neighbourhoods. This shows that the iterative subspace clustering algorithms that can process larger streams cannot identify multi-cluster subspaces, and overlapping clusters in dependent subspaces effectively.

Phase 3: KDD-CUP-99

For the real-world dataset, we selected the well known KDD-CUP'99 dataset on Network Intrusion Detection. This dataset contains a series of TCP connection records from two weeks of LAN network traffic managed by MIT Lincoln Labs. It comprises 495020 data points with 42 attributes where we use all the 34 continuous attributes for clustering. In this dataset, data points are indexed in time and clusters (attack and normal) emerge and contribute to the stream, then they die out and other clusters emerge in the stream. The majority of this dataset are normal records along with 21 types of attacks, and the aim is the real-time detection of intrusions based on their trace in the time-series stream.

The question that we answer is whether there are any multi-cluster subspaces or dependent subspaces in this dataset? If so, are there any cases where some attack clusters are hidden in a subspace containing normal clusters, or, are there any overlapping (normal and attack) clusters in dependent subspaces posing threats to the security of the network? We first present the results of the algorithms in Table 5.3, then address these questions.

The results for the KDD-CUP 99 dataset are presented in Table 5.3. K-flat has the lowest NMImax, ARI and purity with values 0.15, 0.01 and 0.64, respectively. This shows that k-flat has not produced meaningful clusters for this stream. K-subspaces and SCTS have comparable NMI_{max} and ARI values with 0.45 vs 0.42 for NMI_{max} in favour of k-subspaces, and 0.55 vs 0.59 for ARI in favour of SCTS algorithm. Therefore, we can conclude that the clustering quality is similar for both algorithms on this dataset, however, SCTS has a higher purity with 96% compared to the purity of k-subspaces which is 78%. This shows that although the clustering qualities are similar, the purity of clusters for SCTS is notably higher. Being capable of identifying multi-cluster subspaces, and overlapping clusters in dependent subspaces has contributed to the high purity of SCTS algorithm. This means that in k-subspaces there are many attack and normal data points in a single cluster, whereas in SCTS, attack and normal data points are better separated.

Key Findings: Multi-Cluster Subspaces and Dependant Subspaces

In this section, we demonstrate the utility of SCTS by analysing the multi-cluster and dependent subspaces that SCTS has found on the KDD-CUP'99 dataset. SCTS identified 54 subspaces and

Table 5.3: Clustering results of KDD-CUP'99

	KDD-CUP 99		
	NMI_{max}	ARI	Purity
k-flat	0.15	0.01	0.64
k-subs	**0.45**	0.55	0.78
SCTS	0.42	**0.59**	**0.96**

71 clusters in this dataset and we show a subset of these subspaces and clusters in Table 5.4. In this table, each row represents a subspace along with its clusters. As an example, Subspace 21 represents a 3 dimensional subspace that comprises three clusters. Table 5.4 supports the fact that several clusters may form in a single subspace with different ground truth labels. For instance, Subspace 21 comprises three clusters where the first cluster is an attack named "Neptune", the second cluster is normal, and the third cluster is another attack type "Satan" All these clusters have high purity. Being capable of distinguishing several clusters with different ground truth labels by SCTS has contributed to its high purity rate of 96% in Table 5.3.

Table 5.4: Examples of subspaces, their clusters and their dependent subspaces found by SCTS on the KDD CUP 99' dataset

Subspaces	**Clusters**			**Dependent Subspaces**			
Sub 1 1-Dim	Normal 99%	Normal 97%	-	Sub 2 (2-Dim)	Sub 24 (2-Dim)	Sub 34 (2-Dim)	Sub 37 (2-Dim)
Sub 3 1-Dim	Normal 80%	Back 100%	-	Sub 33 (2-Dim)	Sub 35 (2-Dim)	Sub 38 (2-Dim)	Sub 41 (3-Dim)
Sub 4 1-Dim	Smurf 100%	Smurf 99%	Normal 100%	Sub 9 (2-Dim)	Sub 18 (2-Dim)	Sub 21 (3-Dim)	Sub 23 (3-Dim)
Sub 7 1-Dim	Normal 96%	-	-	Sub 33 (2-Dim)	Sub 38 (2-Dim)	Sub 41 (3-Dim)	Sub 53 (3-Dim)
Sub 9 2-Dim	Back 100%	-	-	Sub 4 (1-Dim)	-	-	-
Sub 18 2-Dim	Back 100%	-	-	Sub 4 (1-Dim)	-	-	-
Sub 21 3-Dim	Neptune 99%	Normal 100%	Satan 100%	Sub 27 (1-Dim)	Sub 28 (1-Dim)	Sub 30 (2-Dim)	Sub 33 (2-Dim)
Sub 33 2-Dim	Warez. 100%	-	-	Sub 3 1-Dim	Sub 7 (1-Dim)	Sub 21 (3-Dim)	-

Another key finding of the algorithm is dependent subspaces. SCTS has identified 107 dependent subspaces, a small subset of which are presented in Table 5.4. We examined whether there

are any overlapping clusters within these dependent subspaces, and identified those overlapping clusters that have different ground truth labels. SCTS found 160 overlapping clusters within 107 dependent subspaces (the number of overlapping clusters is greater than the number of dependent subspaces because many of these subspaces have multiple clusters). Out of 160 overlapping clusters, we identified 80 cases in which their ground truth labels were different. For instance, cluster 1 of subspace 4 (labelled as "Smurf" in the ground truth) overlaps with the single cluster in subspace 9 with the ground truth label "Back". Another example is cluster 1 in subspace 4 that is labelled as "Smurf" in the ground truth overlaps with cluster 4 in subspace 21 that is "Normal". The ability to identify cases such as these in this dataset has contributed to the high performance of our method in Table 5.3.

5.6 Conclusion

In this chapter, we proposed the first arbitrarily-oriented subspace clustering algorithm for streaming time-series. Our experiments show that SCTS identifies multi-cluster subspaces and overlapping clusters in dependent subspaces, which contribute to its improved performance in comparison with current state-of-the-art subspace clustering algorithms. The linear time complexity of SCTS enables it to process streams that are unbounded in length. Memory complexity of SCTS does not rely on the number of data points, but instead depends on the number of underlying subspaces and clusters and their dimensionalities. This enables SCTS to store clusters and subspaces in memory efficiently and process unbounded streams. It defines subspaces in local temporal neighbourhoods in the stream and employs the flexibility of hyperellipsoidal cluster prototypes to detect multi-cluster subspaces. We demonstrated the significance of these contributions on the KDD-CUP'99 dataset with almost half a million records where several multi-cluster and dependent subspaces are identified.

One limitation of the SCTS algorithm is that it is for time-series data, where usually a single cluster contributes to the stream. In more complex cases where several clusters may emerge simultaneously, SCTS will not be able to identify those emerging clusters. Data stream clustering algorithms identify such changes in a stream by employing user input parameters such as radius thresholds for clusters (ε), number of clusters in a stream (k), or density thresholds for identifying

dense and sparse regions in the input space. However, finding “good” values for such parameters is challenging as it requires several passes over the data or expert knowledge which may be expensive. In the next chapter, we propose a control structure that can be mounted on online clustering algorithms to address these challenges.

Chapter 6

Incremental Monitoring and Control of Online Clustering Algorithms

Online machine learning techniques process data points in real-time rather than storing a batch of collected data in memory and performing various tasks such as clustering and classifier design. A challenge for processing non-stationary data streams for these purposes is identifying and adapting to changes. An example of a change is when a new cluster emerges in the stream, or when two clusters merge. Most state-of-the-art online clustering algorithms require user-specified input parameters related to the shape, radius, or density of clusters. The parameters set thresholds that are used to identify and adapt to such changes in the data stream. Setting threshold parameters usually requires expert knowledge or requires many passes over the data, which is not practical for online data stream mining. In this chapter we propose an algorithm that is not an online clustering algorithm; rather it is a control structure that is mountable on a selected online clustering algorithm and guides it through changes in the stream. This control structure utilizes an incremental form of the well-known Xie-Beni cluster validity index for online assessment of cluster quality with a forgetting factor to signal changes in the underlying structures in the stream. For instance, when a cluster emerges in the stream the control structure recognizes this change and signals the clustering algorithm to add a cluster prototype. Our experiments on synthetic and real-world datasets show that our proposed control structure improves the performance of two online clustering algorithms.

6.1 Introduction

PATTERNS in non-stationary data stream emerge, evolve and mature over time, and the main challenge is to identify and adapt to these changes in real-time. This task is particularly difficult because every data-point is processed and then discarded, so the only history of the procedure resides in the parameters of a cluster footprint. Guha et al. [56] divide streaming cluster-

ing algorithms into two major groups: (i) split the stream inputs into windows (chunks), use batch clustering algorithms on each window, and then merge the acquired information in the windows; and (ii) use incremental learning techniques to update clusters as data points become available one by one and track their changes in the stream. We refer to the second group of algorithms as online clustering.

A common challenge that online clustering algorithms address is identifying and adapting to various changes that appear in the stream. Researchers have addressed this challenge by defining input parameters that relate to the shape, density or number of clusters believed to emerge in the stream. Some of these parameters are k, the number of clusters believed to exist in the stream; ε, the radius of clusters; $MinPts$, the minimum number of data points in a certain radius to be considered as a cluster; and density thresholds to identify dense regions in the input space. Although there are guidelines for setting these parameter values, finding a "good" value is always challenging, and the goodness of values depends on the underlying data being processed.

We briefly explain several online clustering algorithms and the way they use such input parameters for clarification. D-Stream [31] divides the input features into segments, creating hyper-rectangular grids of equal size. Each grid cell is used to store synopsis information of data points falling into it in secondary storage. D-Stream requires an input parameter for identifying dense grids (C_m) and another input parameter for identifying sparse grids (C_l). Another example is DenStream [26] that inserts a newly arrived data point into its nearest cluster with radius ε under certain conditions. skM [56] and its other variations such as StreamKM++ [1], or "Fast and accurate k-means for large datasets" [126] require k as an input parameter. Other examples include OnCAD (see Chapter 4) and EROLSC [70] that assume data in the stream is coming from a mixture of unknown distributions. EROLSC requires k to identify emerging clusters in its list of anomalies (data points that do not belong to current set of clusters). OnCAD performs a batch clustering (DBScan [44]) on its most recent window when necessary, which requires setting ε for DBScan.

Such parameter values impact the quality of clusterings and one way to find the best value is to measure the cluster quality (performance) produced by different parameter values and choose the best performer. This cluster quality measurement can also be used to compare different clustering methods. Clustering performance on static data is often evaluated with *cluster validity indices*

(CVI). An *external* CVI uses prior knowledge (e.g., ground-truth labels), whereas an *internal* CVI uses information intrinsic to partitions of the data (e.g., the cohesion of data in clusters or the separation of clusters from each other) to make the comparison between different clustering methods or parameter values. The most important point about CVIs is that batch CVIs are not useful for streaming inputs because they require all the data points to reside in memory, whereas in streaming environments the partition is not retained. A recent study by Moshtaghi et al. [102] showed how to make incremental calculations of two well-known batch internal CVIs that can be used in streaming environments. Moreover, they showed that changes in the data stream were signaled by certain patterns in the incremental CVI values. Our aim is to use this idea to impose a control structure on online clustering algorithms, which avoids the problem of setting parameters such as k (the number of prototypes), ε (the radius of a cluster) or density thresholds.

Specifically, we introduce a *control structure* that is capable of guiding a wide variety of online clustering algorithms to adapt to meaningful changes in a data stream. This control structure utilizes the information provided by incremental CVI values to measure how well the current set of cluster statistics (the cluster footprint) represent structure in the stream in real-time. This information is employed to alert the algorithm that a significant change has occurred in the input stream, and then, to possibly create a new element in the footprint, or delete an existing one. We believe this control structure can be used to improve the performance of many online clustering algorithms for adapting to changes in the stream. We choose a simple and well-known algorithm, MacQueen's sequential k-means (skM), as a representative of incremental clustering to demonstrate the effectiveness of our methodology. We also employ the online clustering algorithm OnCAD (from Chapter 4) to show the utility of this methodology by imposing a naive control structure.

6.2 Background and Related Work

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of d-dimensional feature vectors. The sets of fuzzy and crisp non-degenerate (non-zero rows) *k-partitions* of X are the sets of matrices

$$M_{fkn} = \left\{ U \in \mathbb{R}^{kn} : u_{ij} \in [0,1] \ \forall i, j; \sum_{i=1}^{k} u_{ij} = 1 \ \forall j; \sum_{j=1}^{n} u_{ij} > 0 \ \forall i \right\} \quad (\text{f} \Rightarrow \text{fuzzy}) \tag{6.1a}$$

$$M_{hkn} = \{ U \in M_{fkn} : u_{ij} \in \{0,1\} \ \forall i, j \} \quad (\text{h} \Rightarrow \text{hard}) \tag{6.1b}$$

The ij-th element of $U \in M_{fkn}$ is interpreted as the membership of x_j in the i-th fuzzy or hard cluster in X. An equivalent way to represent crisp $U \in M_{hkn}$ is $U \leftrightarrow X = \bigcup_{i=1}^{k} X_i; \emptyset = X_i \bigcap_{i \neq j} X_j$, where $\{X_i\}$ are the k crisp subsets in U. The cardinalities of the k clusters are $|X_i| = n_i, i = 1, \ldots, k; \sum_{i=1}^{k} n_i = n$. Every hard partition is fuzzy, but not conversely, $M_{hkn} \subset M_{fkn}$.

6.2.1 Batch k-Means Models and Algorithms

There are several equivalent ways to formulate *batch k-means* (kM). Let $V = \{v_1, \ldots, v_k\} \subset \mathbb{R}^{kd}$ be a set of k prototypes (cluster centers) in $\mathbb{R}^d$ and let $U \in M_{hkn}$. The *within groups sum of squared errors* (WGSS) objective function is

$$J_1(U, V; X) = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} u_{ij} \left\| x_j - v_i \right\|_A^2 \right) \tag{6.2}$$

where A induces the inner product norm $\left\| x_j - v_i \right\|_A^2 = (x_j - v_i)^T A (x_j - v_i)$. Function J_1 lies at the heart of batch kM, which is the optimization problem (aka the *minimum variance partitioning problem*)

$$\underset{U \in M_{hkn}, V \in \mathbb{R}^{kd}}{\text{minimize}} \{ J_1(U, V; X) \} \tag{6.3}$$

The subscript "1" in eq. (6.2) and eq. (6.3) signifies that hard k-means (hkM, η=1) is a special case of fuzzy k-means (fkM, $\eta \geq 1$). Since our focus is on hkM, We drop the "h" for hkM in the

sequel. The necessary conditions for solutions to eq. (6.3) are well known:

$$u_{ij} = \begin{cases} 1; \left\|x_j - v_i\right\|_A^2 \leq \left\|x_j - v_l\right\|_A^2, i \neq l & \text{for} \quad 1 \leq i \leq k \\ 0; \text{otherwise} & \qquad\quad 1 \leq j \leq n \end{cases} \tag{6.4}$$

$$v_i = \Big(\sum_{x_j \in X_i} x_j\Big) \Big/ n_i = \sum_{j=1}^{n} u_{ij} x_j \Big/ \sum_{j=1}^{n} u_{ij}; \quad \text{for} \quad 1 \leq i \leq k \tag{6.5}$$

The fuzzy generalizations of eqs. (6.2) to (6.5) are, respectively [14, 15]:

$$J_\eta(U,V;X) = \sum_{i=1}^{k} \left(\sum_{j=1}^{n} u_{ij}^\eta \left\|x_j - v_i\right\|_A^2 \right); \; U \in M_{fkn} \, , \; \eta \geq 1 \tag{6.6}$$

$$\underset{U \in M_{fkn}, V \in \mathbb{R}^{kd}}{\text{minimize}} \{J_\eta(U,V;X) : \eta \geq 1\} \tag{6.7}$$

If $\left\|x_j - v_i\right\|_A^2 > 0 \; \forall i, j$, then a pair U,V may minimize $J_\eta(U,V;X)$ for $\eta > 1$ only if:

$$u_{ij} = \left(\sum_{s=1}^{k} \left(\left\|x_j - v_i\right\|_A^2 \Big/ \left\|x_j - v_s\right\|_A^2 \right)^{\frac{1}{\eta - 1}} \right)^{-1} \quad \forall i, j \tag{6.8}$$

$$v_i = \sum_{j=1}^{n} u_{ij}^\eta x_j \Big/ \sum_{j=1}^{n} u_{ij}^\eta \;\; \forall i \tag{6.9}$$

We do not use eq. (6.9) in the proposed methodology, but we do use eqs. (6.4), (6.5) and (6.8). It is shown in [14] that as $\eta \to 1$ from above: (i) $J_\eta(U,V;X) \to J_1(U,V;X)$; (ii) eq. (6.8) converges to eq. (6.4) for all i and k; (iii) eq. (6.9) converges to eq. (6.5) ; and (iv) at η=1, $U \in M_{hkn}$.

The problem in eq. (6.3) is well-defined because there is always a global minimum for J_1, but this function cannot have local minima, because J_1 is discontinuous in U. Thus, it is a non sequitur to state that any method of solving eq. (6.3) terminates at or near a local minimum. There are many models related to the optimization problem in eq. (6.3) which are called "k-means" in various books and papers. The most heavily documented and widely used of these are the batch (global) kM model, and the sequential (local) kM model (skM). The confusion between the two models and algorithms to optimize them is exacerbated by many authors who refer to either of them simply as "k-means" without identifying which version is meant. To make matters

worse, there are many relatives of each of these basic approaches. There are also differences in terminology about whether to use (c) or (k) to denote the number of clusters [15]. The main difference between kM and skM is that the sequential version considers local structure in the data one point at a time, whereas kM tries to capture the global relationship between all the vectors in X during alternating estimation to find a pair (U, V) that might solve eq. (6.3).

The first written work (at least in English) describing a batch version of an iterative algorithm for minimizing $J_1(U,V;X)$ appeared in an unpublished technical report written by Lloyd in 1956 and republished in [90], a procedure also discussed informally by Cox [34]. However, the first discussion of $J_1(U,V;X)$ as a clustering criterion is widely credited to Steinhaus [130]. Moreover, Lloyd acknowledged that an even earlier appearance of his method appeared in a paper written by Lukaszewicz and Steinhaus [130]. There are many ways to attempt minimization of $J_1(U,V)$, but none are as popular as simple alternating iteration through the necessary conditions for U and V in eq. (6.4) and eq. (6.5). This iteration scheme is not quite alternating optimization in the usual sense, because the necessary condition for U in eq. (6.5) cannot be obtained by differentiation [15]. This basic approach is often referred to as Lloyd's algorithm. There is a vast body of papers that use Lloyd's algorithm or a relative of it that are simply called k-means (or c-means).

6.2.2 Sequential k-Means Models and Algorithms

The sequential version of k-means (skM) is attributed to MacQueen [93], who began his 1967 paper as follows: "*The main purpose of this paper is to describe a process ... which is called k-means ... that gives partitions which are reasonably efficient in terms of within-class variance.*" MacQueen credits Sebesteyen [123] with the invention of skM, saying at the end of his introduction "*Sebestyen (1962) has described a procedure called adaptive sample set construction [sec. 4.5 in [123]], which involves the use of what amounts to the k-means process. This is the earliest explicit use of the process with which the author is familiar.*" As we shall see shortly, Sebestyen actually described a slightly more general version than the one that MacQueen called k-means. Here is MacQueen's description:

"*Stated informally, the k-means procedure consists of simply starting with k groups each of which consists of a single random point, and thereafter adding each new point to the group whose mean the new point is nearest. After a point is added to a group, the mean of that group is adjusted*

in order to take account of the new point. Thus at each stage the k-means are, in fact, the means of the groups they represent (hence the term k-means). ... When the sample points are rearranged in a new random order, there is some variation in the grouping which is obtained." The pseudo-code for skM is listed in Algorithm 5.

Algorithm 5 Sequential k-means

procedure SKM(X,k)
  Initialize V_k with the first k data points $V_k = \{x_1, x_2, \ldots, x_k\}$
  U_k is the $k \times k$ identity matrix
  $n_1 = n_2 = \ldots = n_k = 1$
  for all x_n in X **do** ▷ Start n from k+1
    $d(v_j, x_n) = \arg\min_{1 \leq i \leq k} \{d(v_i, x_n)\}$
    $n_j = n_j + 1$
    $v_j = v_j + (x_n - v_j) / n_j$
    The *j-th* column of u_n is set to 1, $u_n = (0,0,\ldots,1,\ldots 0)^k$, where the 1 is at the *j-th* place
  end for
  return V_n, U_n
end procedure

MacQueen's skM ends in one pass through X with estimates for $U \in M_{hkn}, V \in \mathbb{R}^{kd}$. The two main problems with applying MacQueen's skM to batch data are that: (i) k must be prespecified; and (ii) the results are order dependent. A third disadvantage of skM that is sometimes mentioned is that the clusters are basically hyperspherical. But this is due to using Euclidean distances. Neither MacQueen nor Sebestyen specified this metric, and it is clear that both methods can be used with any metric on the input space. Sebestyen's procedure was slightly more general than skM. We paraphrase the method he gave in 1962 on p. 102 of [123]:

[Procedure S] *When the first input is introduced, it is assigned to class 1, and this input becomes the mean vector of class 1. The second input arrives. If it is within a threshold distance ε of the first mean, it is assigned to class 1, and the mean is updated. Otherwise, it becomes the mean of a new class 2. Continuing, this procedure results in a set of points which represent the locations of clusters of inputs, whose approximate sizes are ε.*

Sebestyen called his algorithm "sample set construction". Procedure S is clearly the genesis of the well known and often used *leader algorithm* for batch clustering of X which was named and aptly described by [60] and [137]. The leader algorithm is easily adapted for online use with streaming data by simply continuing Procedure S as inputs arrive. Removal of the requirement

that k must be pre-specified is paid for by having to choose the threshold ε, and it is still order dependent.

MacQueen's skM is arguably the original *competitive learning* algorithm. Some competitive learning schemes do not pre-specify k, others require it as an input. The hallmark of all competitive learning schemes is the competition $\sim$ winner take all, or follow the leader. The general form of competitive learning algorithms is focused on obtaining a good set of prototypes V that represent X compactly. But once the prototypes are found, it is a simple matter to use them with the *nearest prototype rule* (1-NPR) at eq. (6.4) to generate the companion k-partition built by Lloyd's algorithm. A pair (U,V) obtained in this way might solve eq. (6.3), but in all likelihood do not.

Using competitive learning algorithms to generate k-partitions of X was never strongly advocated by Kohonen. Nonetheless, there have been many papers devoted to clustering schemes based on this more general form of skM. The partitions built by eq. (6.4) using prototypes from competitive learning schemes are justified by asserting that a high-quality set of cluster centers (V) coupled to U via eq. (6.4) will necessarily produce a high-quality partition. Based on this, there have been many studies of ways to accelerate the competitive learning algorithm, and these methods are often called *adaptive k-means*, but most of them do not apply to the online case [36, 32, 112, 98, 99]. Work on the theory and acceleration of adaptive leader algorithms has continued up to the present: see [95] for a comprehensive survey. However, these algorithms continue to require pre-specification of the parameter k, and do not generally target clustering as their online application.

There have been many papers devoted to clustering schemes based on this more general form of skM. The partitions built by eq. (6.5) using prototypes from competitive learning schemes are justified by asserting that a high-quality set of cluster centers (V) coupled to U via eq. (6.5) will necessarily produce a high-quality partition. Based on this, there have been many studies of ways to accelerate the competitive learning algorithm, and these methods are often called *adaptive k-means*, but most of them do not apply to the online case [36, 32, 112, 98, 99]. Work on the theory and acceleration of adaptive leader algorithms has continued up to the present: see [95] for a comprehensive survey. However, these algorithms continue to require pre-specification of the parameter k, and do not generally target clustering as their online application.

6.2.3 Batch Cluster Validity Indices (bCVIs)

Let CP denote a set of *candidate partitions* of a batch dataset:

$$CP = \{U \in M_{hkn} : k_m \leq k \leq k_M\} \tag{6.10}$$

where k_m and k_M are the minimum and maximum values of the integer k. Cluster validity refers to the problem of choosing the "best" $U \in CP$. Various heuristics are associated with different *batch CVIs* (bCVIs), but most often the "best" partition in CP in the sense of a *real-valued function* $V : M_{hcn} \in \mathbb{R}$ is the one that yields the maximum (*max-optimal* $\uparrow$) or minimum (*min-optimal* $\downarrow$) value among the partitions in CP. Books discussing batch cluster validity include an early text by Jain and Dubes [42], and a quite comprehensive treatment by [131] and [10].

A well-known fact about the utility and success of bCVIs at choosing a "best partition" from those on offer is that none of them are uniformly reliable over even a few different datasets and clustering algorithms. There are many excellent surveys on using *internal* bCVIs that compare selected indices using various clustering algorithms, different datasets and different measures of success. The seven papers [41, 97, 40, 25, 10, 138, 153] itemize and compare about 150 different bCVIs over various input parameters and definitions of what constitutes a "best" partition, but none of them offer conclusive evidence for selecting a best "general purpose" bCVI. To the best of our knowledge, only two bCVIs have been adapted for use in the context of online monitoring of streaming inputs. This is the topic we turn to next.

6.3 Incremental Cluster Validity Indices (iCVIs)

We start with an illustration of the general idea. Suppose that n inputs have arrived sequentially and been clustered online, leaving a *cluster footprint* $V_n \in \mathbb{R}^{k_n d}$. The number of prototypes at time n (k_n) has been learnt during processing (note that V_n is V at time n and k_n is k at time n). Now input x_{n+1} arrives, is assigned memberships $\{u_{i,n+1} : 1 \leq i \leq k\}$ and x_{n+1} is used to update $V_n \rightarrow V_{n+1}$ by eqs. (6.4) and (6.5). The calculation of the label vector $\{u_{i,n+1} : 1 \leq i \leq k\}$ and prototypes V_{n+1} are done by an incremental clustering such as skM.

The fundamental questions about this procedure are: (i) is there sufficient evidence to warrant

creating a new cluster $(k_{n+1} \leftarrow (k_n + 1))$ for this input?; or (ii) should two existing clusters be merged $(k_{n+1} \leftarrow (k_n - 1))$?; and (iii) which prototypes should be merged? Bear in mind that we have not retained the previous data-points so we cannot reproduce the partition U_n that might be computed with eq. (6.4) and V_n. Besides, there are no parameters such as k or ε to help answer these questions. The evidence we propose to rely upon here will be in the form of an incremental value (one step update) of a cluster validity index. This will lead us to a new incremental form of skM.

Dubes et al. [41] proposed a set of four structural criteria that any bCVI should possess: [recognition of] Compactness, Isolation, Global fit to data and Intrinsic dimensionality of the data. Most researchers agree that the first two (compactness and separation) are pre-eminent properties to be possessed by "good" clusters. Separation between clusters that have cluster centers $\{v_k\}$ is often defined as the inter-cluster center distance.

$$sep(v_i, v_j) = \left\| v_i - v_j \right\|. \tag{6.11}$$

There are many ways to capture the idea of dispersion (compactness) of a cluster. We define the *fuzzy within cluster dispersion* of the i-th cluster in fuzzy partition $U \in M_{fkn}$ after n streaming inputs as

$$C_{i,n,\eta} = \sum_{j=1}^{n} u_{i,j}^{\eta} \left\| x_j - v_{i,n} \right\|^2; \eta \geq 1. \tag{6.12}$$

This quantity is a measure of the compactness of the i-th fuzzy cluster about its centroid. When $U \in M_{hkn}$ is crisp, values of u_{ij}^{η} are either 1 or 0 for all values of $\eta \geq 1$. The quantity in eq. (6.12) appears in many internal bCVIs that are used to evaluate crisp partitions of X. For example, the *Davies-Bouldin* (DB, $\downarrow$) index by [38]) and the *Xie-Beni* index (XB, $\downarrow$) by [147] both incorporate this term as part of their functions. Moreover, this term appears in a simple modification of two of the 17 *Generalized Dunn Indices* (GDIs, $\uparrow$), viz., GDI 43 and GDI 53 [16], which are incrementalized in [69].

[102] defined eq. (6.12) for only η=2. Ibrahim et al. [68] pointed out that the factor u_{ij}^2 at $\eta = 2$ in eq. (6.12) behaves like a constant, i.e., it factors out of all terms in the derivation of the update equation for eq. (6.12) when the next input arrives, so when U is fuzzy, eq. (6.12) can be

parametrized in η. [102] wrote the incremental form of eq. (6.12) (for η=2) as:

$$C_{i,n+1,\eta} = C_{i,n,\eta} + \Delta C_{i,n,\eta} \tag{6.13}$$

The computation of eq. (6.13) is the key to forming incremental versions of the four bCVIs (DB, XB, GDI 43, GDI 53). We reproduce the algorithm given in [102] but generalized for any $\eta \geq 1$ as Algorithm 6, which yields, for all i, the updated value of the function $C_{i,n+1,\eta}$ in eq. (6.13) as a function of x_{n+1}. The norm is the Euclidean norm, $A = I_p$.

Algorithm 6 Incremental fuzzy withing cluster dispersion calculation for x_{n+1} without forgetting

procedure ICOMPACT($v_{i,n}, v_{i,n+1}, u_{i,n+1}, x_{n+1}$, $G_{i,n}, M_{i,n}, C_{i,n}$)
  Initialize: $v_1 = x_1$, $k_1 = 1$
  for $i \leftarrow 1$ to k **do**
    $Q_{i,n+1} = (v_{i,n} - v_{i,n+1})^T G_{i,n}$
    $B_{i,n+1} = ||v_{i,n} - v_{i,n+1}||^2$
    $A_{i,n+1} = (u_{i,n+1})^\eta ||X_{n+1} - v_{i,n+1}||^2$
    $C_{i,n+1} = C_{i,n} + A_{i,n+1} + M_{i,n} B_{i,n+1} + 2Q_{i,n+1}$
    $G_{i,n+1} = G_{i,n} + M_{i,n}(v_{i,n} - v_{i,n+1}) + (u_{i,n+1})^\eta (X_{n+1} - v_{i,n+1})$
    $M_{i,n+1} = M_{i,n} + (u_{i,n+1})^\eta$
  end for
  return $G_{i,n+1}, M_{i,n+1}, C_{i,n+1}$
end procedure

Algorithm 6 is the heart of the incremental iCVIs developed by [102] and [69, 68]. Notice especially that the winning prototype determines a crisp membership for the incoming point, but after the prototypes are updated, memberships are recomputed with the necessary condition for minimization of the fuzzy k-means objective function displayed in eq. (6.8). See [15] for details of this function.

Moshtaghi et al. also developed incremental forms of the DB and XB iCVIs that incorporate a forgetting factor. We use the XB forms in this chapter, but we have experimented with both of these iCVIs, and either can be used with our incrementalized version of skM. The Xie-Beni index

by [147] used only the Euclidean norm, and has the form

$$XB_2(U,V;X) = \frac{\sum_{i=1}^{k}\left(\sum_{j=1}^{n} u_{ij}^2 \left\|x_j - v_i\right\|^2\right)}{n\left(\min_{i\neq j}\{\left\|v_i - v_j\right\|^2\}\right)} = \frac{\sigma_2(U,V;X)}{n \times sep(V)} = \frac{J_2(U,V;X)}{n \times sep(V)} \tag{6.14}$$

While the model norm specified was Euclidean, any inner product norm can be used in eq. (6.14). Xie and Beni interpreted their index by writing it as a ratio of the *total generalized variation* $\sigma_2(U,V;X)$ of (U,V), to $sep(V)$, the separation of the prototype vectors V as stipulated in eq. (6.6) and shown in eq. (6.9). A good (U,V) pair should produce a small value of the generalized WGSS $\sigma_2 = J_2$, indicating good cohesion (or compactness) of the k clusters in U. And well-separated v_i's will produce a high value of $sep(V)$. So, when $XB_2(U_1,V_1;X) < XB_2(U_2,V_2;X)$ for either or both of these reasons, U_1 is presumably a better partition of X than U_2. Consequently, the *minimum* of XB_2 over $CP \subset M_{fkn}$ is taken as the most desirable partition of X (i.e., XB_2 is *min-optimal* ($\downarrow$)). Xie and Beni pointed out that XB_2 is related to the fuzzy k-means objective function [15] at η = 2, but it can also be used on fuzzy partition pairs (U,V) found by any clustering algorithm that produces $U \in M_{fkn}$ and $V = \{v_1,\ldots,v_k\} \subset \mathbb{R}^{kd}$. Since $M_{hkn} \subset M_{fkn}$, XB_1 is an internal bCVI that can be used for hard k-means at η = 1.

[108] extended the Xie-Beni index to the more general form

$$XB_\eta(U,V;X) = \frac{\sum_{i=1}^{k}\left(\sum_{j=1}^{n} u_{ij}^\eta \left\|x_j - v_i\right\|^2\right)}{n\left(\min_{i\neq j}\{\left\|v_i - v_j\right\|^2\}\right)} = \frac{J_\eta(U,V;X)}{n \times sep(V)}; \eta \geq 1 \tag{6.15}$$

Comparing eq. (6.15) to eq. (6.12) shows that XB_m contains exactly the fuzzy cohesion term for each cluster defined by [68]. Let $iXB_m(n)$, $J_\eta(n)$ and $\{v_{i,n}\}$ be the current values in eq. (6.14) after n inputs, and let $iXB_m(n+1)$, $J_\eta(n+1)$ and $\{v_{i,n+1}\}$ denote updated values after x_{n+1} arrives. The numerator of eq. (6.14) is then updated using the values of k_n from Algorithm 6. Equation (6.16) shows the one step update of iXB_m:

$$iXB_\eta(n+1) = \frac{\sum_{i=1}^{k_{n+1}} C_{i,n+1,\eta}}{(n+1) \times \min_{i\neq j}\left\|v_{i,n+1} - v_{j,n+1}\right\|^2} \tag{6.16}$$

The incremental update shown at eq. (6.16) is displayed for the Euclidean norm. Whether this incremental update remains valid for other inner product norms is still an open question. Now we turn to the problem of the historical impact of past inputs on sequential updates.

Experiments reported in [102] show that new streaming inputs will become less and less impactful on the total value of the updates $\{C_{i,n+1,\eta}\}$ as $n \to \infty$. To minimize this saturation effect, we add a *forgetting factor* $(0 < \lambda < 1)$ to the incremental model by incorporating it in the fuzzy cohesion term:

$$C_{\lambda,i,n,\eta} = \sum_{j=1}^{n} \lambda^{n-j} u_{ij}^{\eta} \left\| x_j - v_{i,n} \right\|^2; \eta \geq 1 \tag{6.17}$$

In this way, the contribution to $C_{\lambda,i,n,\eta}$ of the sample that arrived (q) inputs before sample (n) is weighted by (λ^{n-q}). Since $0 < \lambda < 1$, $\lambda^{n-q} < \lambda^{n-q+1}$, so older samples lose their relevance as newer inputs arrive. The speed at which λ^{n-q} as $n \to \infty$ depends on the choice of λ, but it will be fast in any case. Moshtaghi et al. derived the following update formulae for $iXB_{\lambda,\eta}$, the incremental form of iXB_m with forgetting factor λ (with $\eta = 2$):

$$\begin{aligned}
&Q_{\lambda,i,n+1,\eta} = (v_{i,n} - v_{i,n+1}) G_{\lambda,i,n} \\
&C_{\lambda,i,n+1,\eta} = C_{\lambda,i,n,\eta} + \Delta C_{\lambda,i,n,\eta} \\
&\Delta C_{\lambda,i,n+1,\eta} = 2\lambda Q_{\lambda,i,n+1,\eta} + \lambda M_{\lambda,i,n,\eta} B_{i,n+1,\eta} + A_{i,n+1,\eta} \\
&M_{\lambda,i,n+1,\eta} = \lambda M_{\lambda,i,n,\eta} + u_{ij}^{\eta};\ \ M_{\lambda,i,1,\eta} = u_{i,1}^{\eta} \\
&G_{\lambda,i,n+1,\eta} = \lambda G_{\lambda,i,n,\eta} + \Delta G_{\lambda,i,n,\eta} \\
&\Delta G_{\lambda,i,n+1,\eta} = \lambda M_{\lambda,i,n,\eta} (v_{i,n} - v_{i,n+1}) + u_{ij}^{\eta} ((x_{n+1} - v_{i,n+1}))
\end{aligned} \tag{6.18}$$

Computation of factors A and B in Algorithm 6 are not affected by the forgetting factor. Substitution of eq. (6.18) into the appropriate lines of Algorithm 6 yield the update $C_{\lambda,i,n+1,\eta}$ for the fuzzy cohesion with forgetting, and substituting $C_{\lambda,i,n+1,\eta}$ into the numerator of eq. (6.16) yields the one-step update for the incremental Xie-Beni index with forgetting factor (λ):

$$iXB_{\lambda\eta}(n+1) = \frac{\sum_{i=1}^{k_{n+1}} C_{\lambda,i,n+1,\eta}}{(n+1) \min\limits_{i \neq j} \left\| v_{i,n+1} - v_{j,n+1} \right\|^2}; \quad \eta \geq 1; 0 < \lambda < 1 \tag{6.19}$$

In the next section, we define our incremental version of skM, which is augmented by the use

of eq. (6.19) to detect changes in the input data and to control the creation (and also deletion) of clusters that emerge (or disappear) in the streaming inputs. Our experiments use only $\eta = 2$, so we remove this variable from subsequent equations and algorithms. See [69, 68] for experiments on the effects of η on iCVIs.

6.4 Incremental Monitoring and Control of "Sequential k-means"

In this section, we propose the monitoring and control structure for skM as an example of online clustering algorithms.We name the new algorithm as Incremental Sequential k-means (iskM), however this process can be adapted to many other online clustering algorithms as well. Online clustering algorithms based on maintaining a record of cluster centers (such as skM) should be able to adapt to changes in the streaming data by adding, removing or updating the prototype footprint. The main questions that need to be answered in such an environment are: (i) is there sufficient evidence to warrant adding a new prototype?; (ii) should two existing prototypes be merged?; and (iii) which prototypes should be merged? In the following section we explain how we use the incremental Xie-Beni index values to answer these questions.

6.4.1 Incremental Sequential k-means (iskM)

The Xie-Beni index is the ratio of cohesion to separation eq. (6.19). Therefore, after a new input is processed, a jump in iXB_λ indicates either a jump in cohesion (the input is far from all prototypes) or a drop in separation (a sign that two prototypes are close). One way to use the iXB_λ values is to run skM, detect jumps in cohesion or drops in separation, and make decisions (adding or removing prototypes) based on the observed behaviour. The problem with this approach is that jumps and drops are quite frequent in the streaming environment, and the unbounded nature of those measures aggravates the problem. As a result, we employ *two* skMs that run in parallel so that any decision made by one, is validated by the other.

We first present a high-level description of iskM, which comprises three components: (i) the first component is an skM (we call *Current-skM*) that is sensitive to changes in separation; (ii) the second component is another skM (*Control-skM*) that is sensitive to changes in cohesion; and (iii) a module that compares the two index iCVI values and decides if an addition or removal of a

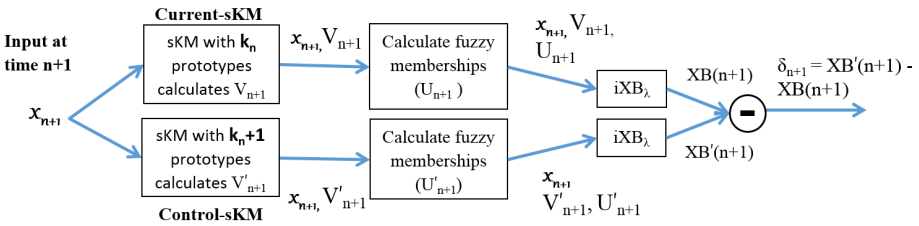


Figure 6.1: One step update process of iskM for time-correlated data

prototype is required (note that we work with the iCVI values of the two skMs and not the cohesion or separation measures directly because of the problems discussed in the previous paragraph).

While processing inputs, a similar size jump in the iXB_λ values of Current-skM and Control-skM by the new input does not provide useful information because it is impossible for a data-point to increase cohesion and decrease separation simultaneously. So this is considered to be a stable state in the stream. However, a bigger jump in the iXB_λ values of Control-skM (that is more sensitive to cohesion) is an indication that the newly arriving data point is far from the current set of cluster prototypes, and a new prototype needs to be added. On the other hand, a bigger jump in the iXB_λ values of Current-skM (that is more sensitive to separation) indicates that prototypes are too close to each other and need to be merged. Figure 6.1 shows a diagram of the proposed algorithm and Algorithm 7 represents the pseudo-code. In the rest of this section, we formally describe the process of change-point detection.

When x_{n+1} arrives, the current set of prototypes of Current-skM and Control-skM are updated, i.e., $V_n \rightarrow V_{n+1}$ and $\boldsymbol{V'_n} \rightarrow \boldsymbol{V'_{n+1}}$ in line 7 of Algorithm 7. Then fuzzy memberships are computed for Current-skM and Control-skM with eq. (6.8), resulting in the fuzzy label vectors $\{U_{n+1}\}$ and $\{U'_{n+1}\}$ for x_{n+1}. Then we can calculate iXB_λ values for both Current-skM and Control-skM (after introducing x_{n+1} to them) in lines 10 and 11 that are used by the third component.

For notational convenience, let the $XB(n+1)$ and $XB^{'}(n+1)$ be the iXB_λ values of Current-skM and Control-skM (respectively) at time (n+1). Let $\overline{XB}(n)$ and $\overline{XB^{'}}(n)$ denote the mean of the indices for Current-skM and Control-skM skM (respectively) for the first n inputs, and let $vXB(n)$ and $vXB^{'}(n)$ denote the variance of the iXB_λ values of Current-skM and Control-skM (respectively) for the first n inputs. We compare $XB(n+1)$ with $\overline{XB}(n)$ and $vXB(n)$, and $XB^{'}(n+1)$ with $\overline{XB^{'}}(n)$ and $vXB^{'}(n)$ to check if the iXB_λ values of the new input deviates much from the last pair of iCVI values. Now let $\delta_{n+1} = XB^{'}(n+1) - XB(n+1)$ at time $n+1$, and let

Algorithm 7 Incremental sequential k-means for streaming environments

procedure ISKM(X, λ, $\lambda^{'}$)
  $V_2 = \{x_1\}$ ▷ Initialize V_2
  $V_2^{'} = \{x_1, x_2\}$ ▷ Initialize $V_2^{'}$
  $n_1 = n_1' = n_2' = 1$
  initperiod = 10
  for the new x_{n+1} in X **do** ▷ Start n from 2
    V_{n+1} = skM(x_{n+1}, V_n); V'_{n+1} = skM(x_{n+1}, V'_n) ▷ Update Cluster Centers
    Calculate U_{n+1} and U'_{n+1} by eq. (6.8)
    $XB(n+1)$ = $iXB_\lambda(V_{n+1}, U_{n+1}, x_{n+1})$
    $XB^{'}(n+1)$ = $iXB_\lambda(V'_{n+1}, U'_{n+1}, x_{n+1})$
    $\delta = XB^{'}(n+1) - XB(n+1)$
    ▷ Check for adding prototype
    if $XB^{'}(n+1) \geq \overline{XB^{'}}(n) + 1.5 \times \sqrt{vXB^{'}(n)}$ **and** $\delta \geq \overline{\delta} + 1.5\sqrt{v\delta}$ **and** $n >$ initperiod **then**
      $V_{n+1} = V_{n+1} \cup x_{n+1}$; $n_{new} = 1$
      $V'_{n+1} = V'_{n+1} \cup x_{n+1}$; $n'_{new} = 1$
    end if
    ▷ Check for merging prototypes
    if $XB(n+1) \geq \overline{XB}(n) + 1.5\sqrt{vXB(n)}$ **and** $\delta \leq \overline{\delta} + 1.5\sqrt{v\delta}$ **and** $n >$ initperiod **then**
      Calculate localiXB(i) $\forall v_i \in V_{n+1}$ by eq. (6.23)
      i = max(localiXB)
      if $i > 1$ **then** ▷ Do not remove the first prototype in Current-skM
        $d(v_j, x_n) = \arg\min_m \{d(v_m \in V \setminus v_i, v_i)\}$
        $n_j = n_j + n_i$ ▷ Merge v_j and v_i
        $v_j = (n_j v_j + n_i v_i) \,/\, (n_j + n_i)$
        $n'_j = n'_j + n'_i$
        $v'_j = \left(n'_j v'_j + n'_i v'_i\right) / \left(n'_j + n'_i\right)$
        $n_i = \emptyset$; $v_i = \emptyset$ ▷ remove v_i
        $n'_i = \emptyset$; $v'_i = \emptyset$
      end if
    end if
    Update $\overline{\delta}$ and $v\delta$ by eqs. (6.20a) and (6.20b)
    Update $\overline{XB}(n+1)$ and $vXB(n+1)$ by (6.20c,6.20d)
    Update $\overline{XB^{'}}(n+1)$ and $vXB^{'}(n+1)$ by (6.20e,6.20f)
  end for
  return V_{n+1}
end procedure

$\overline{\delta}_n = \sum_{i=2}^{n} \delta_i/(n-1)$ denote the mean of the changes in the index values over the first n inputs (there is no δ_1), and let $v\delta_n$ denote the variance of the $(n-1)$ values of δ.

The last step of our method is the incremental calculation of the means and the variances for both the indices and their difference. Following [24], we introduce eqs. (6.20a) to (6.20f) for incremental calculation of $\overline{\delta}_{n+1}$, $v\delta_{n+1}$, $\overline{XB}(n+1)$, $vXB(n+1)$, $\overline{XB'}(n+1)$ and $vXB^{'}(n+1)$ from the calculated values of $XB(n+1)$ and $XB^{'}(n+1)$ with the forgetting factor λ' that is used by the third component:

$$\overline{\delta}_{n+1} = \lambda'\overline{\delta}_n + (1-\lambda')\delta_{n+1}; \tag{6.20a}$$

$$v\delta_{n+1} = \lambda' v\delta_n + \frac{1-\lambda'^2}{2}(\delta_{n+1} - \overline{\delta}_n)^2; \tag{6.20b}$$

$$\overline{XB}(n+1) = \lambda'\overline{XB}(n) + (1-\lambda')XB(n+1); \tag{6.20c}$$

$$vXB(n+1) = \lambda' vXB(n) + \frac{1-\lambda'^2}{2}(XB(n+1) - \overline{XB}(n))^2 \tag{6.20d}$$

$$\overline{XB'}(n+1) = \lambda'\overline{XB'}(n) + (1-\lambda')XB^{'}(n+1); \tag{6.20e}$$

$$vXB^{'}(n+1) = \lambda' vXB^{'}(n) + \frac{1-\lambda'^2}{2}(XB^{'}(n+1) - \overline{XB'}(n+1))^2 \tag{6.20f}$$

Note that: (i) the λ' forgetting factor is different from λ that is defined for the iXB_λ, (ii) eqs. (6.20a), (6.20c) and (6.20e) are the same formula over different values (similar to eqs. (6.20b), (6.20d) and (6.20f)), (iii) the values δ_{n+1}, $\overline{\delta}_n$, $v\delta_n$, $XB(n+1)$,$\overline{XB}(n)$, $vXB(n)$, $XB^{'}(n+1)$,$\overline{XB'}(n)$ and $vXB^{'}(n)$ determine the circumstances to add or remove prototypes. After completing the calculations for the third component, we can answer the following questions.

6.4.2 Is There Sufficient Evidence to Warrant Creating a New Cluster?

The cohesion measure in eq. (6.17) is defined by the distance of data-points from the prototypes. So, when a new cluster emerges, we expect a sudden jump in the index values of both components. The jump for Control-skM (that is sensitive to cohesion) would be larger as it has an extra prototype to account for the new cluster. More specifically, since the denominator in eq. (6.17) has the form

$$(n+1)\min_{i\neq j} \left\| v_{i,n+1} - v_{j,n+1} \right\|^2$$

it incorporates the distance of the two closest prototypes to calculate iXB_λ. Hence, an increase in the numerator reflects a greater change in the Control-skM as it has two prototypes (that are very close to each other) in a single cluster. The jump of Current-skM will be smaller since one prototype (that already represents a cluster) needs to move towards the emerging cluster. Accordingly, a significantly larger jump of iXB_λ for Control-skM is an indication of an emerging cluster. Hence, when there is a jump at $XB^{'}(n+1)$, we validate it by checking that δ_{n+1} also has a jump (the difference of $XB^{'}(n+1)$ and $XB(n+1)$ is high). If both conditions are met, we add a prototype to both components. Accordingly, when

$$XB^{'}(n+1) \geq \overline{XB^{'}}(n) + 1.5\sqrt{vXB^{'}(n)}; \quad \text{and} \tag{6.21a}$$

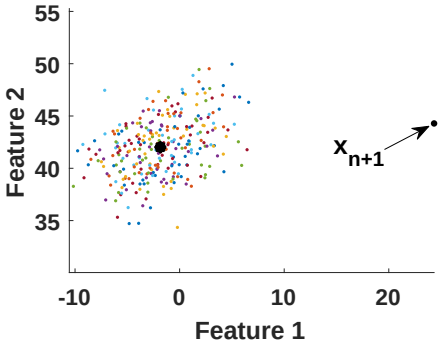
$$\delta_{n+1} \geq \overline{\delta}_n + 1.5\sqrt{v\delta_n} \tag{6.21b}$$

we define this as a significant change between Current-skM and Control-skM and add x_{n+1} as a new prototype to both algorithms. These conditions are illustrated in Figure 6.2. The new datapoint x_{n+1} arrives, both components are updated (Figure 6.2a and Figure 6.2b) and $XB(n+1)$, $XB^{'}(n+1)$ and δ_{n+1} are calculated. Here, a prototype needs to be added because: (i) $XB^{'}(n+1)$ is beyond the 3σ region (Figure 6.2d, eq. (6.21a)), and (ii) δ_{n+1} is beyond the 3σ region (Figure 6.2e, eq. (6.21b)).

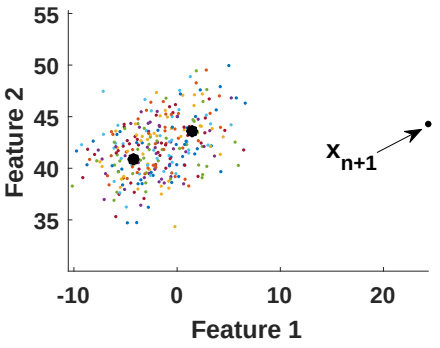
The value 1.5 used to detect deviation from the mean is the adjusted 3σ value for a one-sided statistical test. Eq. (6.21a) ensures that there has been a jump in the index values of Current-skM, and eq. (6.21b) ensures that the jump is significant. Whenever eqs. (6.21a) and (6.21b) hold, we add a new prototype for the incoming x_{n+1}.

6.4.3 Should Two Existing Clusters Be Merged?

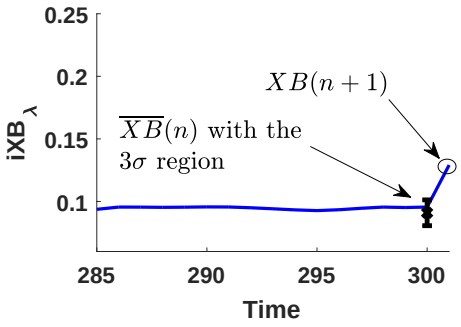
To answer this question, our aim is to identify *redundant* prototypes. A redundant prototype is: (i) a prototype that is added to a cluster that already is represented by a prototype or (ii), is a prototype that has moved so close to another prototype that it does not provide any useful information. For either of these situations, the separation measure of the Current-skM becomes smaller and we



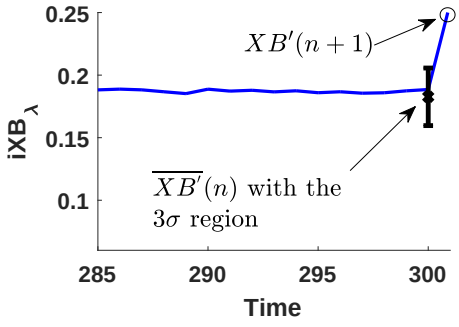
(a) Current-skM



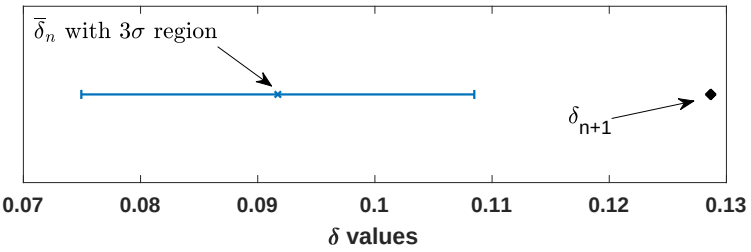
(b) Control-skM



(c) Current-skM iXB$_\lambda$ values



(d) Control-skM iXB$_\lambda$ values



(e) The mean and the variance of the δ values. δ_{n+1} is out of the 3σ region because the jump at the iXB$_\lambda$ value of the Control-skM is bigger than the jump of the iXB$_\lambda$ value of the Current-skM

Figure 6.2: The criteria to add a prototype. The new data-point x_{n+1} that is relatively far from the current cluster(s) causes a jump the iXB$_\lambda$ values of both components. Since the jump of the Control-skM value is bigger, a prototype needs to be added for x_{n+1}

expect an increase in its iXB_λ value. Since the denominator in eq. (6.17) has the form

$$(n+1)\min_{i\neq j}\left\|v_{i,n+1}-v_{j,n+1}\right\|^2$$

it incorporates the distance of the two closest prototypes to calculate iXB_λ. A decrease in the denominator reflects a greater change in the Current-skM as its prototypes are spread over the clusters. Moreover, the change in the Control-skM would be smaller (if there is any) since it already has a redundant prototype and has two prototypes that are very close to each other. Accordingly, when prototypes are too close to each other, there should be a significant change in the index values of Current-skM. Hence, after observing a jump at $XB(n+1)$, we check its significance by checking if δ_{n+1} has a sudden drop (the difference is high). If both conditions are met we need to remove a prototype by merging. Accordingly, when

$$XB(n+1) \geq \overline{XB}(n) + 1.5\sqrt{vXB(n)} \quad \text{and} \tag{6.22a}$$

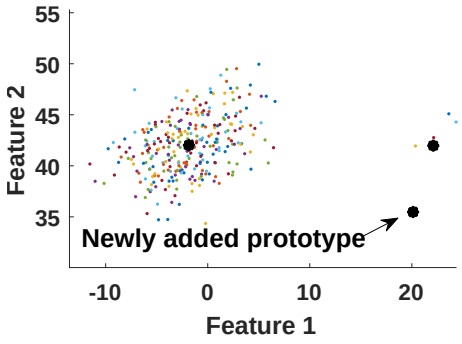
$$\delta_{n+1} \leq \overline{\delta}_n - 1.5\sqrt{v\delta_n} \tag{6.22b}$$

we define this as a significant change between the Current-skM and Control-skM and a prototype needs to be removed. These conditions are illustrated in Figure 6.3. This figure is the continuation of Figure 6.2 where at input 307 a prototype is mistakenly (Figure 6.3a and Figure 6.3b) added because eqs. (6.21a) and (6.21b) hold. The added prototype needs to be removed because: (i) $XB(n+1)$ is beyond the 3σ region (Figure 6.3c, eq. (6.22a)), and (ii) δ_{n+1} is below the 3σ region (Figure 6.3e, eq. (6.22b)).

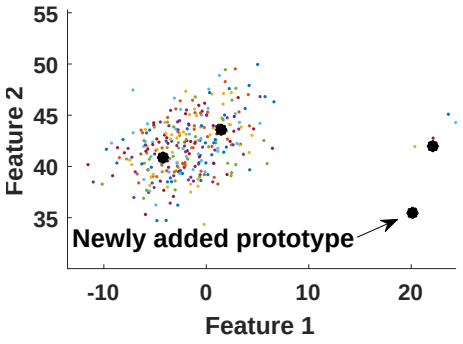
Again, eq. (6.22a) ensures that there has been a jump in the index values of Current-skM, and eq. (6.22b) ensures that the jump is significant. The value 1.5 used to detect deviation from the mean is the adjusted 3σ for a one-sided statistical test. Note that at each input that eq. (6.22) holds, we remove a prototype by merging two prototypes.

6.4.4 Choose the Prototype to be Removed

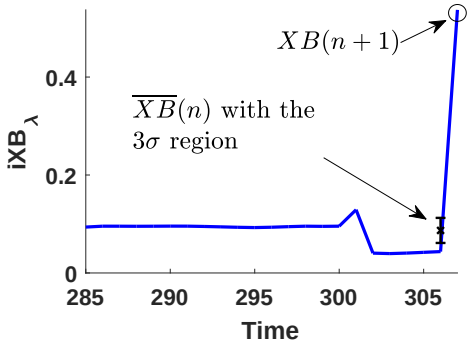
The last question we need to address is: which prototype should be removed? One method is to find the two closest prototypes and merge them. However, since the merger is done whenever the



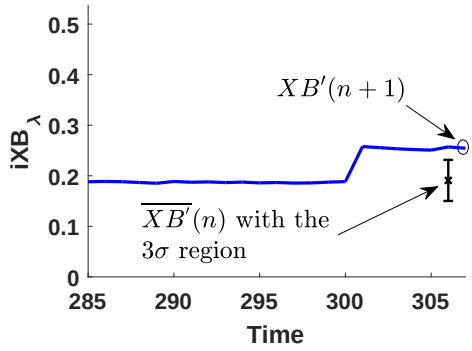
(a) Current-skM



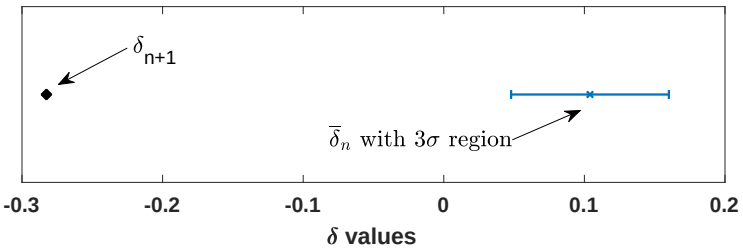
(b) Control-skM



(c) Current-skM iXB$_\lambda$ values



(d) Control-skM iXB$_\lambda$ values



(e) The mean and the variance of the δ values. δ_{n+1} is out of the 3σ region because the jump of iXB$_\lambda$ of the Current-skM is larger than the jump of iXB$_\lambda$ of the Control-skM

Figure 6.3: The criteria to merge prototypes. The newly added prototype that is relatively close to another prototype in Current-skM, makes a jump in the iXB$_\lambda$ values of Current-skM. Since the jump at the Current-skM is larger, two prototype need to be merged

conditions in eq. (6.22) hold, it may cause the removal of unwanted prototypes. Therefore, we calculate *local* iXB$_\lambda$ values for each prototype i by

$$iXB_{\lambda i}(n+1) = \frac{C_{\lambda,i,n+1}}{n_i \times \min\limits_{i\neq j} \left\| v_{i,n+1} - v_{j,n+1} \right\|^2} \tag{6.23}$$

where $C_{\lambda,i,n+1}$ is the fuzzy within-cluster dispersion of prototype i, n_i is the number of data-points of prototype i, and $\min \left\| v_{i,n+1} - v_{j,n+1} \right\|^2$ is the distance of the closest prototype to prototype i. Eq. 6.23 indicates how cohesive and separate a prototype is in its local neighbourhood. We merge the prototype that has the maximum local iXB_λ value with its closest prototype. Note that the merge is accomplished by averaging the chosen pair of prototypes.

At this point, x_{n+1} is discarded, $\overline{\delta}_{n+1}$, $v\delta_{n+1}$, $\overline{XB}(n+1)$, $vXB(n+1)$, $\overline{XB'}(n+1)$ and $vXB'(n+1)$ are calculated and iskM proceeds to next input. In summary: (i) recurrent jumps and drops in the iCVIs are addressed by employing two indices whose relative difference indicates the importance of the change; (ii) to detect emerging clusters, we make Control-skM more sensitive to cohesion by adding an extra prototype to one of its clusters; (iii) to detect redundant prototypes, we use Current-skM, which carries the cluster footprint of iskM; (iv) to validate decisions made by each skM we use the 3σ one-sided statistical test; and (v) adding and removing prototypes by Current-skM and Control-skM often ends in an equilibrium that minimizes iXB$_\lambda$ values of Current-skM. We expect slightly higher iXB_λ values of Control-skM because it has a redundant prototype.

This algorithm is implemented in Matlab[1] and publicly available[2].

6.4.5 Time Complexity of iskM

In the streaming environment the size of the stream is infinite, therefore we calculate the worst-case time complexity of iskM for a single update. For each new input, Current-skM and Control-skM are updated in $O(k)$, where k is the number of prototypes at the current time for Current-skM. Then the fuzzy memberships of the new input with the prototypes are calculated in $O(k)$, as the distance of the new input must be calculated with each prototype.

To calculate XB(n+1), the numerator of eq. (6.19) is calculated in $O(k)$ and the denominator (finding the two closest prototypes) in $O(k^2)$, resulting in the complexity of $O(k^2+k)$. Similarly,

[1] https://mathworks.com/ [2] https://github.com/mchenaghlou/iskm

the time complexity of XB′(n+1) is $O(k^2+k)$. Checking eq. (6.21) and eq. (6.22) is performed in $O(1)$. Adding a prototype is calculated in $O(1)$, but removing a prototype is performed in $O(k^2)$, since for each prototype we need to calculate eq. (6.23) (this equation involves finding the closest prototype to each prototype). Updating the means and variances are performed in $O(1)$.

Therefore, the total time complexity of iskM is $O(k^2+k)$, i.e., $O(k^2)$. For a dataset of size n, the total time-complexity of iskM will be $O(nk^2)$ (k is the number of prototypes) where the time complexity of skM is $O(nk)$. It is worth noting that although the time complexity of iskM is quadratic in terms of the number of prototypes (k), it is linear with respect to the number of data-points (n).

6.5 Evaluation

The main contribution of this chapter is a control structure that can be mounted on stream clustering algorithms and used for guiding them through the changes in a stream. Therefore, we compare the performance of two online clustering algorithms with and without this control structure. First, we compare skM, that requires the number of underlying clusters, to iskM, that employs the proposed control structure. Then, we use our online clustering algorithm known as *OnCAD* (Online Clustering and Anomaly Detection) from Chapter 4 to further demonstrate the effectiveness of the proposed control structure. After integrating our proposed control structure on the OnCAD algorithm, we refer to it as i-OnCAD in this chapter.

Recall that OnCAD is a sliding window based algorithm that employs incremental updating of hyper-ellipsoidal cluster prototypes as data points arrive. It calculates the minimum window length based on an input parameter that relates to the weights of the components in a mixture distribution that is desired to be detected. On the arrival of a new data point, OnCAD updates the hyper-ellipsoidal cluster prototypes based on its cluster membership, and flags the newly arrived data point if it does not belong to any existing clusters. When a new cluster emerges, the window is filled with tagged data points that do not belong to any existing clusters, therefore, a batch clustering algorithm (DBScan) is performed on the tagged data to identify emerging clusters. OnCAD with DBScan requires user-supplied parameters for the radius of clusters (ε), and the minimum number of data points in a cluster. Although OnCAD proposes a formula to calculate

the minimum number of data points in a cluster, it requires ε to be specified by the user. Moreover, while the footprint that represents a cluster in skM and iskM is just the cluster center, OnCAD and i-OnCAD have a more sophisticated form, viz., a prototype that is the (cluster) center of a hyperellipsoid with boundary specified by the ellipsoidal radius.

We mounted a naive incremental control structure on OnCAD called i-OnCAD. The main difference of these two algorithms is that OnCAD relies on input parameters to identify changes in the stream, whereas i-OnCAD employs a simple control structure that assists it to adapt to the changes in the stream. Our intention is to show that i-OnCAD and iskM, both of which utilize the information provided by the control structure in real-time, outperform OnCAD and skM, respectively, that rely on user input parameters. Our implementation of iskM and i-OnCAD along with the following experiments are publicly available on github at https://github.com/iskmeans/iskm.

The proposed experiments comprise two synthetic and one real-life datasets. The synthetic datasets are used to show how the proposed control structure identifies changes in the stream. We provide figures to illustrate that the control structure alerts the clustering algorithm when a change occurs in the stream. The real-life dataset is the GSA (Gas Sensor Array under dynamic gas mixtures) provided in [47]. In this dataset, several chemical sensors monitor an environment that is subjected various mixtures of Ethylene and CO, and the task is to identify when the gas mixtures by the sensor readings.

6.5.1 Experiment 1 (Clusters Emerge Sequentially)

The first synthetic dataset X1, shown in Figure 6.4, comprises seven clusters that emerge sequentially in the order: $X_1, X_2, X_3, \ldots, X_7$. Our aim is to show that the proposed control structure can effectively identify clusters that sequentially emerge in the stream. The seven clusters in X1 have 1000 data points each, so a total of 7000 points are presented to the algorithms

In this experiment we compare the results of skM and OnCAD (that require pre-specified input parameters) with iskM and i-OnCAD (with our incremental control structure), respectively. The results of skM and iskM are illustrated in Figure 6.5. Figures 6.5a and 6.5b show the cluster centres calculated by skM and iskM as black stars (in these figures, the colors show membership labels). The skM algorithm was given the number of labeled subsets (k=7), so it automatically finds 7 prototypes, but the centers it finds, shown in Figure 6.5a, are quite misleading. This figure shows

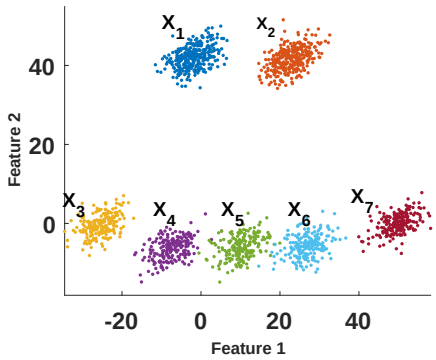
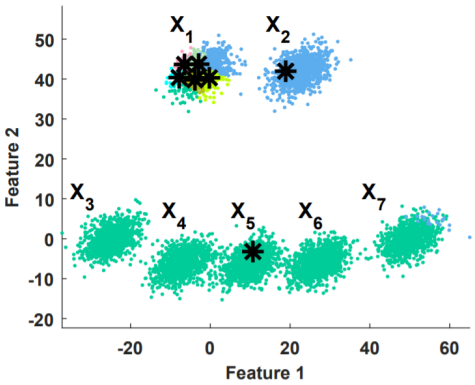


Figure 6.4: Synthetic dataset

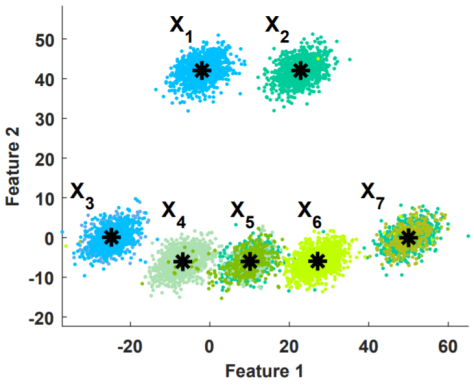
that the set of cluster prototypes produced by this algorithm does not represent the underlying clusters. Specifically, skM lands five prototypes in X_1, one in X_2, and one in X_5. The other four clusters are not represented in the skM footprint at all. On the other hand, iskM yields one cluster center in each of the seven clusters as shown in Figure 6.5b.

Figure 6.5c represents the iXB_λ values calculated for skM and iskM. The colors in the background in this figure show the span of the different clusters contributing to the stream. For the brown (thick) graph, iXB_λ only monitors the skM process, but for the blue graph iXB_λ both monitors and controls iskM in real-time. These values show how well (with respect to the Xie-Beni index with a forgetting factor) the cluster prototypes produced by the algorithms represent the underlying clusters. The poor performance of skM is reflected by its significantly higher iXB_λ values in Figure 6.5c compared to iskM. This is because as clusters emerge during the process, the data points fall relatively far from cluster prototypes for skM which results in an increase in the cohesion measure of this index. An important observation is that whenever a cluster emerges in the stream, a sudden increase of iXB_λ values is observed for the skM algorithm. However, iskM utilizes this information by adding and merging prototypes with the help of the proposed control structure.

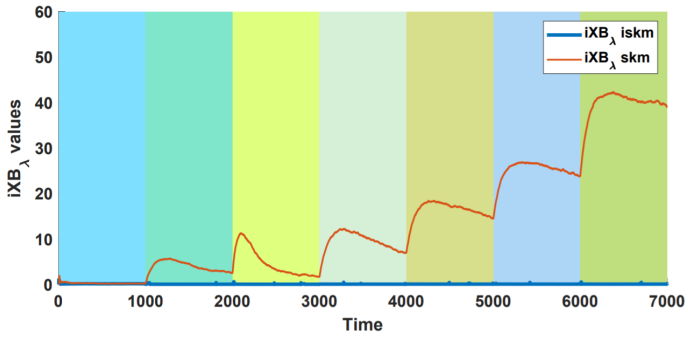
Since the iskM process of adding and merging prototypes is not clear in Figure 6.5c, we added Figure 6.5d to better illustrate this process. This figure shows how iskM utilizes the iXB_λ values of Current-skM and Control-skM to adjust to the changes in the stream (the times when a prototype is added is depicted by a $+$ sign and the times when a prototype is merged is depicted by a $\triangledown$ sign). The iskM algorithm tells us when a prototype is added and when a prototype is merged.



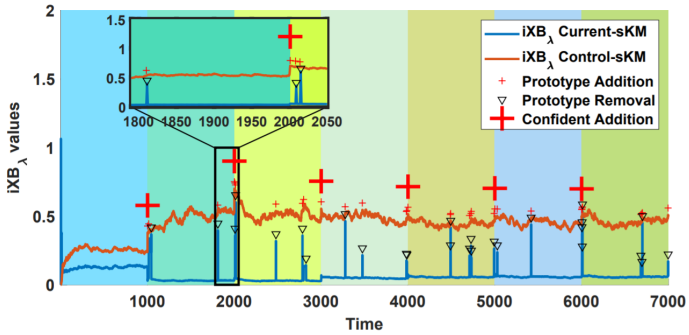
(a) The clustering results of skM



(b) The clustering results of iskM



(c) The iXB_λ values of skM and iskM. Different colors show membership labels. The poor performance of skM is reflected by its significantly higher iXB_λ values. When a new cluster emerges the iXB_λ values of skM increases considerably



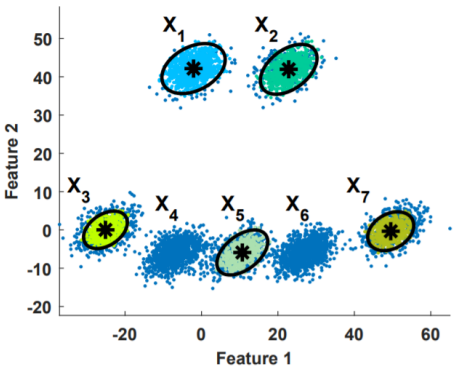
(d) The iXB_λ values of Current-skM and Control-skM in iskM. Different colors show membership labels. These two components work in parallel to adjust to the changes in the stream by adding and merging prototypes. Big + signs represent the additions that are not associated with a merger

Figure 6.5: The clustering results of experiment 1 (dataset X1, clusters X_1 to X_7 emerge in order). The set of cluster prototypes produced by skM in 6.5a shows that skM lands five prototypes in X_1, one in X_2, and one in X_5. The other four clusters are not represented in the skM footprint at all. On the other hand, the results of iskM in 6.5b yields one cluster center in each of the seven clusters. The poor performance of skM is reflected by its significantly higher iXB_λ values in 6.5c compared to iskM

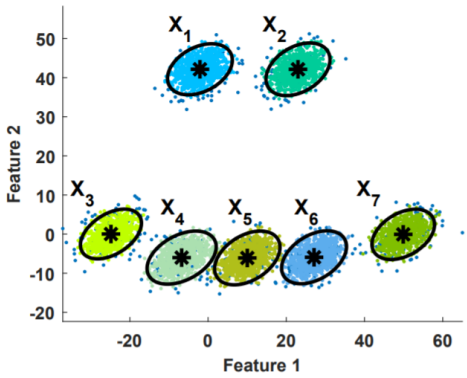
Our observation is that iskM adds a prototype for the slightest change, but then recognizes that it was wrong and removes it. By filtering out these unwanted additions, we can understand how the clusters evolved. This can be seen in Figure 6.5d. Figure 6.5d shows how Current-skM and Control-skM work together by adding and merging cluster prototypes to adjust to the changes in the underlying clusters in the stream. In this figure, the prototype additions that are not associated with a merge, are called "confident additions" and represented with a big + sign. Our results show that iskM has added 34 prototypes and merged 27 that correctly yields 7 prototypes for this dataset.

The results of OnCAD and i-OnCAD for this experiment are illustrated in Figure 6.6. OnCAD uses input parameters for the radius of clusters (ε), whereas i-OnCAD uses a naive control structure to alert it for adding and merging prototypes. To set the parameter ε for OnCAD, we ran this algorithm for various values ε = 0.5, 1 and 1.5, and calculated the Xie-Beni index (min-optimal) for each number (28.02, 0.08, 0.65) and chose the radius that minimized this index (in this case, 1). Comparing Figures 6.6a and 6.6b shows that OnCAD has missed two clusters, viz., X_4 and X_6 because the chosen parameters could not isolate them, whereas i-OnCAD has correctly identified the underlying clusters in the stream. This experiment shows that the control structure plays a major role in correctly identifying the changes in the underlying clusters in the stream.

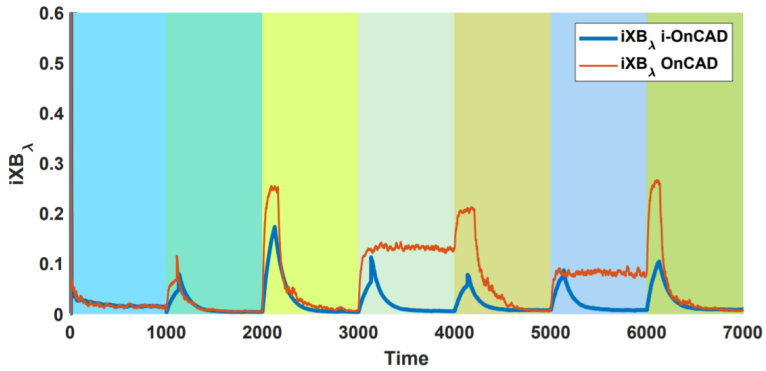
Figure 6.6c represents the iXB_λ values calculated for OnCAD and i-OnCAD. For the first 3000 data points, the iXB_λ values are very similar for both algorithms because they correctly identify emerging clusters. However, between 3000 and 4000, the iXB_λ values of OnCAD are notably higher than i-OnCAD. This is because OnCAD misses the cluster X_4. From the time 4000 to 5000, we see a gradual decrease in iXB_λ values of OnCAD. This is because it correctly identifies the cluster X_5, and since iXB_λ incorporates a forgetting factor, it gradually forgets about the cluster X_4 and these iXB_λ values represent the data points in the cluster X_5. However, between 5000 and 6000, OnCAD misses the cluster X_6. Towards the end of the stream, where X_7 is correctly identified by both algorithms, their iXB_λ values get close to 0. On the other hand, as for i-OnCAD, whenever a new cluster emerges, a sudden increase is observed with the iXB_λ values of i-OnCAD but adding a cluster prototype decreases this value.



(a) The clustering results of OnCAD



(b) The clustering results of i-OnCAD



(c) The iXB_λ values of OnCAD and i-OnCAD. Different colors show membership labels. OnCAD has missed two clusters (X_4 and X_6) which is reflected by having higher iXB_λ values in two periods: from 3000 to 4000, and from 5000 to 6000. I-OnCAD correctly idenfies the seven emerging clusters

Figure 6.6: The clustering results of experiment 1 (dataset X1, clusters X_1 to X_7 emerge in order). The set of cluster prototypes produced by OnCAD in 6.6a shows that OnCAD does not recognize two clusters X_4 and X_6. On the other hand, 6.6b shows that i-OnCAD yields one cluster center in each of the seven clusters. The poor performance of OnCAD is reflected by its higher iXB_λ values in 6.6c

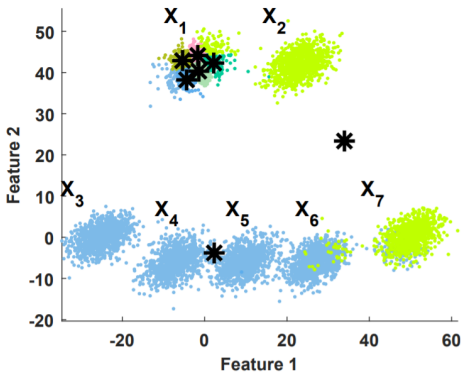
6.5.2 Experiment 2 (Clusters Emerge Concurrently)

The second synthetic dataset X2 is similar to X1 except that $X_3, \ldots, X_7$ emerge simultaneously. That is, points are drawn from the remaining 5 subsets and arrive as inputs in random order instead of indexed order as in X1. X2 is used to show that the control structure is capable of identifying more complex changes in the stream. The outputs of skM and iskM in Figures 6.7a and 6.7b show that skM has not produced a reasonable representation for the underlying clusters, whereas iskM has correctly identified them. Bear in mind that skM was given the true number of underlying clusters as its input parameter, whereas iskM yields a better performance only by monitoring the iXB_λ values calculated in real-time.

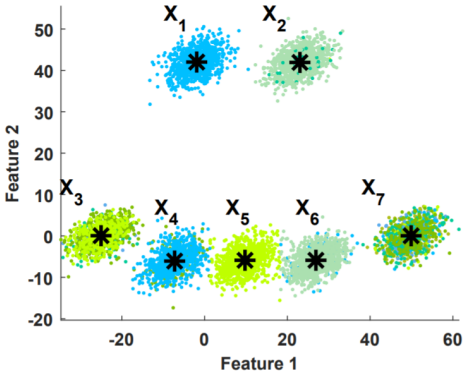
The iXB_λ values of skM and iskM are shown in Figure 6.7c. This figure shows that after time 2000, skM totally misses the concurrently emerging clusters as reflected in its significantly higher iXB_λ values. On the other hand, iskM maintains a significantly lower iXB_λ values for the whole process by adding and merging prototypes whenever signaled by the control structure.

Figure 6.7d shows that iskM has added 40 prototypes and merged 33 which yields 7, the true number of underlying clusters. Each addition and merger is performed by monitoring the iXB_λ values of Control-skM and Current-skM. As an example, we closely study this process from the time 1980 to 2050 where five clusters emerge. In this period, iskM added prototypes in times 2001, 2002, 2003, 2004, 2006 and 2011, but the prototype added at time 2004 was a redundant prototype that has resulted in a sudden increase in the iXB_λ values of Current-skM. Therefore, it is merged in time 2005 which reduces the iXB_λ values of Current-skM considerably. This can be validated by the ground truth labels that the bottom five clusters emerge at the mentioned times. Based on the ground truth, no cluster emerges for the rest of the stream, and iskM maintains the true number of prototypes by adding and merging them only by monitoring the iXB_λ values in real-time.

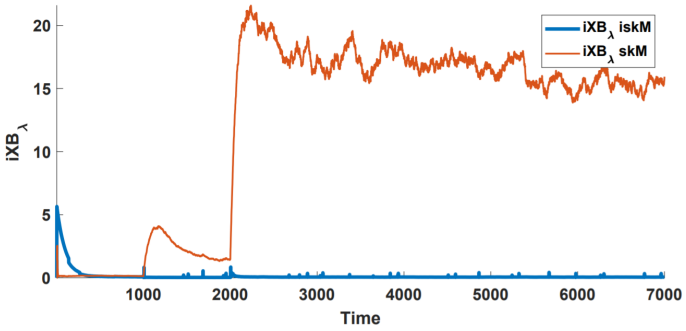
The results of OnCAD and i-OnCAD are illustrated in Figure 6.8. To set the radius (ε) of clusters for OnCAD, we ran this algorithm for various values ε = 0.5, 1 and 1.5, and calculated the Xie-Beni index (min-optimal) for each number (28.02, 1.63, 27.74) and chose the radius that minimized this index (in this case, 1). Comparing Figures 6.8a and 6.8b shows that OnCAD totally misses the bottom five clusters, mainly because the radius parameter could not identify those clusters in the stream. However, the control structure of i-OnCAD has correctly identified



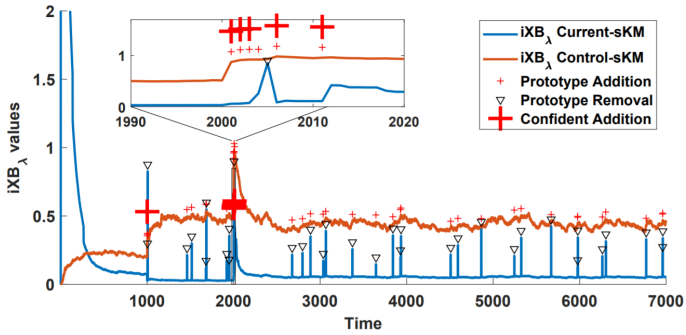
(a) The clustering results of skM



(b) The clustering results of iskM



(c) The iXB_λ values of skM and iskM. After the time 2000 where skM misses the five bottom clusters, its iXB_λ values increases for the rest of the stream



(d) The iXB_λ values of Current-skM and Control-skM in iskM. Different colors show membership labels. These two components work in parallel to adjust to the changes in the stream by adding and merging prototypes. Big + signs represent the additions that are not associated with a merger

Figure 6.7: The clustering results of experiment 2 (dataset X2, clusters X_3 to X_7 emerge concurrently). The set of cluster prototypes produced by skM in 6.7a shows that skM has not produced a reasonable representation for the underlying clusters, whereas iskM has correctly identified them. The poor performance of skM is reflected by its significantly higher iXB_λ values in 6.7d compared to iskM

the changes in underlying clusters and alerted the clustering algorithm for adding or merging prototypes.

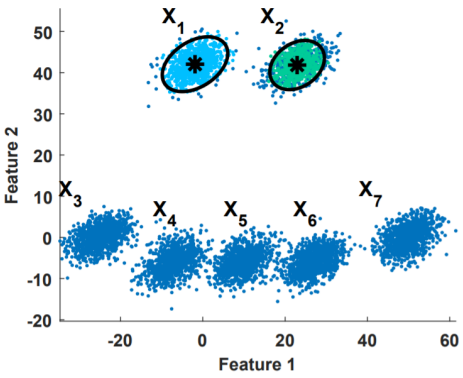
Figure 6.8c represents the iXB_λ values calculated for both algorithms in real-time. In this figure, up to the time 2000, the iXB_λ values of both OnCAD and i-OnCAD are quite similar as they both correctly identify the cluster X_2. However, after the time 2000, the iXB_λ values of OnCAD increases significantly and remains high for the rest of the stream. On the other hand, as for i-OnCAD, the iXB_λ of i-OnCAD increases between the times 2000 and 3000, but after the time 3000, it adapts to the changes in the stream by correctly maintaining the true number of underlying clusters illustrated in Figure 6.8b.

6.5.3 Experiment 3 (Visualization of GSA Dataset)

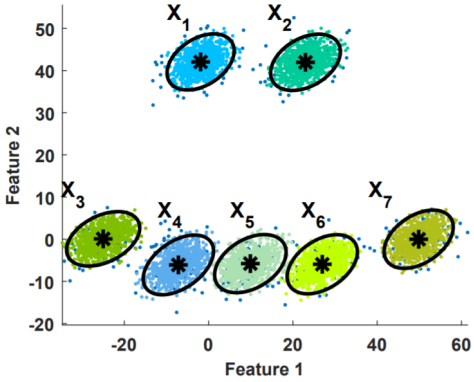
In this experiment we employ a real-world dataset, *GSA* (Gas Sensor Array under dynamic gas mixtures) provided in [47]. In this dataset, gas mixtures of varying concentration levels were introduced and measured by 16 chemical sensors. We chose the Ethylene and CO dataset that were recorded for 12 hours at a rate of 100 Hz. This dataset is mainly for drift detection, where applying various gas mixtures creates drift. After exposing a gas mixture to the environment, it takes about four seconds for the sensors to capture the change. Therefore, we chose a subset of the data after about 4 seconds of exposing a new mixture where it reaches an equilibrium, considering them as clusters.

The main goal of this experiment is to show that iskM is capable of tracking the changes of clusters in the stream. One problem of using all of the 16 dimensions is that the distribution of prototypes over the data points cannot be illustrated, therefore we tested and chose the two best features that describe the clusters to visualize the distribution of the calculated prototypes over the dataset. The next experiment is dedicated to a scenario where the number of dimensions is high. The number of data points in this dataset is 57,012 and based on the ground truth, there are 12 underlying clusters that are depicted in Figure 6.9.

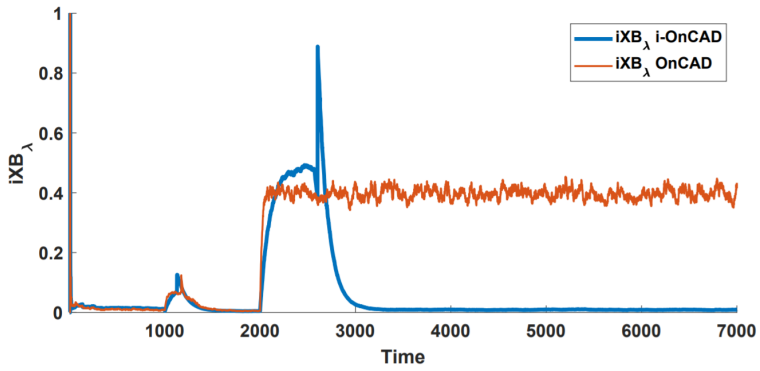
The results of skM and iskM are illustrated in Figure 6.10. Figures 6.10a and 6.10b represent the prototypes calculated by skM and iskM by black stars. Although skM was given the true number of underlying clusters, i.e., 12 in this case, it has a poor performance as evident in Figure 6.10a. This figure shows that the cluster footprint produced by this algorithm does not represent



(a) The clustering results of OnCAD



(b) The clustering results of i-OnCAD



(c) The iXB_λ values of OnCAD and i-OnCAD. Different colors show membership labels. OnCAD has missed two clusters (X_4 and X_6) which is reflected by having higher iXB_λ values in two periods: from 3000 to 4000, and from 5000 to 6000. I-OnCAD correctly idenfies the seven emerging clusters

Figure 6.8: The clustering results of experiment 2 (dataset X2, clusters X_3 to X_7 emerge concurrently). Comparing the clustering results in 6.8a and 6.8b shows that OnCAD totally misses the bottom five clusters, mainly because the radius parameter could not identify those clusters in the stream. However, the control structure of i-OnCAD has correctly identified the changes in underlying clusters and alerted the clustering algorithm for adding or merging prototypes. The poor performance of OnCAD is reflected by its higher iXB_λ values in 6.8c

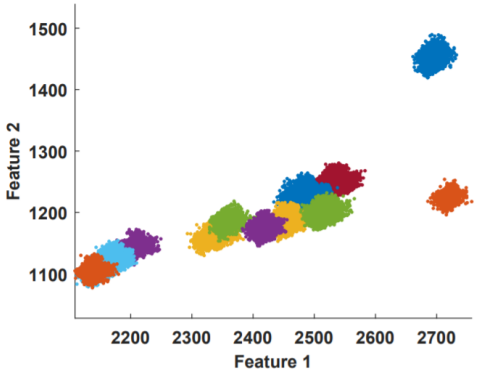
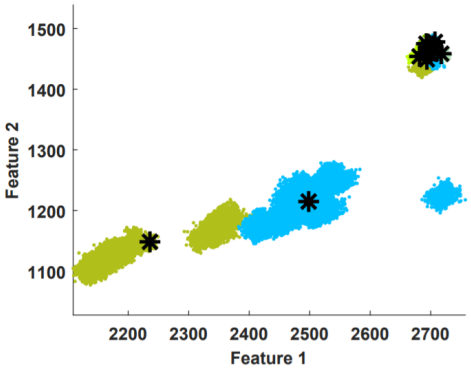


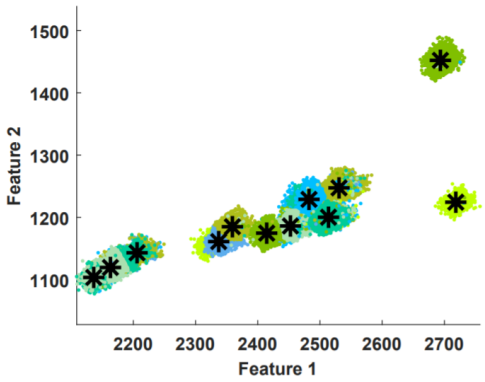
Figure 6.9: GSA dataset

the underlying clusters. Specifically, skM lands 10 prototypes in the top-right cluster, and two prototypes for the rest of the dataset. On the other hand, iskM yields one prototype in each of the 12 clusters as shown in Figure 6.10b. Figure 6.10c represents the iXB_λ values calculated for skM and iskM. In this figure, different colors in the background determine the times that the clusters contribute to the stream. The poor performance of skM is reflected by its significantly higher iXB_λ values compared to iskM.

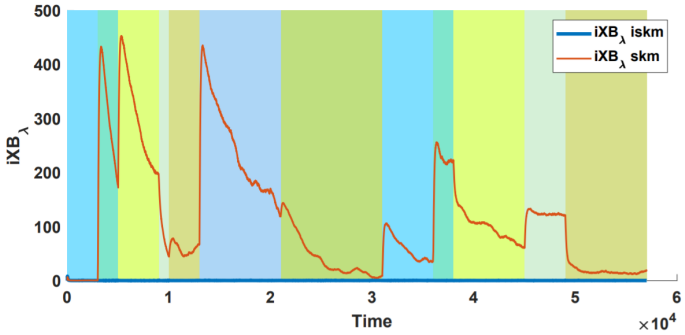
Now we explain how iskM utilizes the iXB_λ values to adjust to the changes in the stream. Figure 6.10b shows that iskM has tracked the true number of underlying clusters correctly. To present a better understanding of the iskM process, we present the iXB_λ values of the Current-skM and Control-skM in Figure 6.10d, and the times that it has added or merged prototypes. In this figure, different colors determine the times that clusters were contributing to the stream. Moreover, the times that prototypes are added are depicted by a $+$ sign and the times that prototypes are merged are depicted by a $\triangledown$ sign. Our results show that iskM has added 391 cluster centres and merged 379 that correctly yields 12 prototypes for this dataset. We explain this process from the time 8900 to 10200 in detail. At times 9004 and 9012 two prototypes were added but the latter was merged due to an increase in the iXB_λ of the Current-skM. This can be validated by the ground truth where a single cluster emerges at 9004. Again, at times 9434, 9605, 9609, 9773, 9795, 9972, 10007 and 10010, eight prototypes are added but then immediately removed because they were redundant, except the prototype added at time 10010. This can also be validated by the ground truth, since another cluster emerges at time 10005. Although it takes a few data points for iskM to recognize this emerging cluster, Figure 6.10d shows how Current-skM and Control-skM



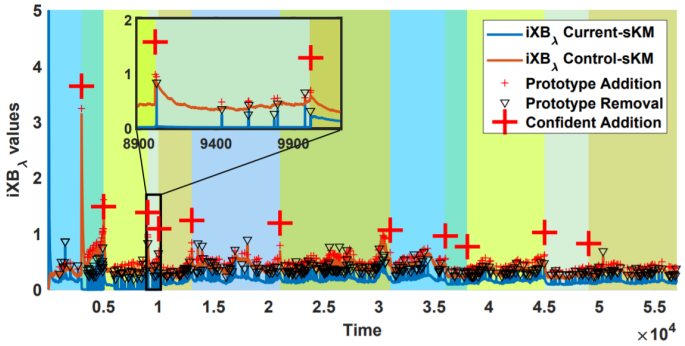
(a) The clustering results of skM



(b) The clustering results of iskM



(c) The iXB_λ values of skM and iskM. Different colors show membership labels. The poor performance of skM is reflected by having significantly higher iXB_λ values



(d) The iXB_λ values of Current-skM and Control-skM in iskM. Different colors show membership labels. These two components work in parallel to adjust to the changes in the stream by adding and merging prototypes. Big + signs represent the additions that are not associated with a merger

Figure 6.10: The clustering results of experiment 3 (real-life gas sensor array dataset). The set of cluster prototypes produced by skM in 6.10a shows that skM has not produced a reasonable representation for the underlying clusters, whereas iskM has correctly identified them. The poor performance of skM is reflected by its significantly higher iXB_λ values in 6.10d compared to iskM

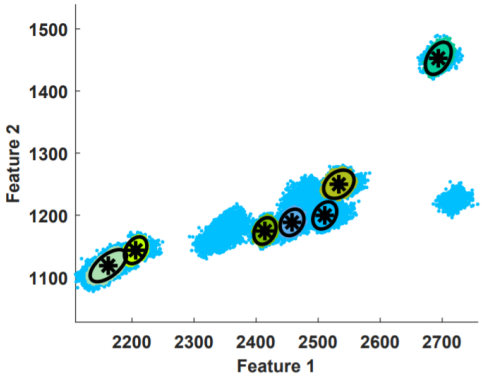
work together by adding and merging cluster prototypes to adjust to the changes in the underlying clusters in the stream.

The results of OnCAD and i-OnCAD are presented in Figure 6.11. To set the parameter ε (the radius of clusters) for OnCAD, we ran this algorithm for various values ε = 1, 2, 3, 4, 5 and 5.5, and calculated the Xie-Beni index (min-optimal) for each number (249, 249, 249, 206, 0.26, 0.27) and chose the radius that minimized this index (in this case, 5). Figure 6.11a shows that OnCAD added seven clusters but misses five clusters which is a poor representation of the underlying clusters. However, i-OnCAD has correctly identified the 12 clusters in this dataset. The performance of the algorithms is reflected in Figure 6.11c, which represents the iXB_λ values calculated for both algorithms in real-time.

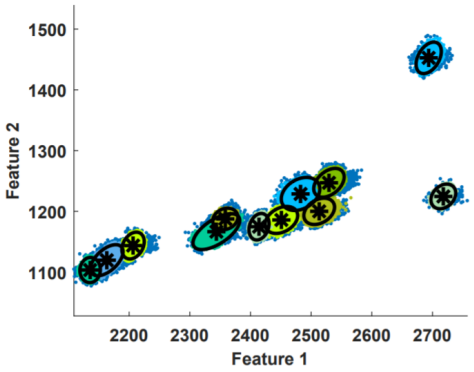
6.5.4 Experiment 4 (Real-life GSA Dataset)

In this experiment we employ the GSA dataset, the same dataset as in Experiment 3, but instead of using 2 features out of 16 available feature, we use 12 of them. Here, since it is not possible to visualize the data and the produced clusters, our aim is to show that how the iXB_λ values guide clustering. The reason we exclude 4 features is that their readings are not consistent with other sensors, and when other sensors reach an equilibrium, their readings continue varying, which prevents the formation of clusters. We also exclude a comparison of OnCAD with i-OnCAD in this experiment, mainly because OnCAD's performance deteriorates as the number of features increases.

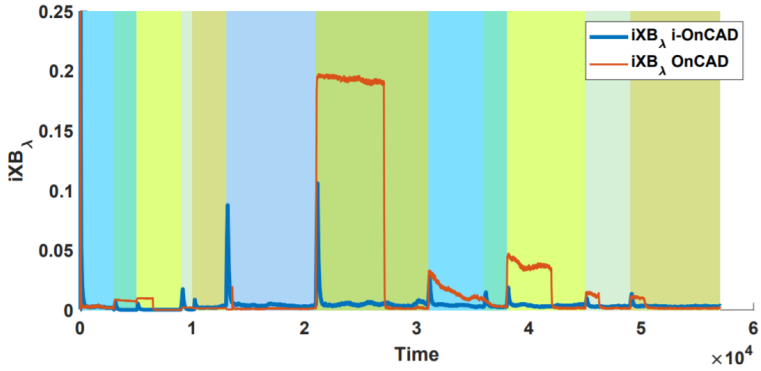
Figure 6.12a represents the iXB_λ values calculated for skM and iskM. The poor performance of skM is reflected by its significantly higher iXB_λ values compared to iskM. The maximum value of iXB_λ for skM is calculated as 70,725 which has occurred at time 13,002 right before the sixth cluster starts to contribute to the stream. Figure 6.12b represents the iXB_λ values of the two components of iskM. In this figure, the background colors show membership labels. It can be verified that when a cluster emerges, by the change of color, a confident addition is performed by iskM, which indicates that the emerging cluster has been identified. The only exception is at time 38000 where a cluster emerges, but there is no confident addition for it. After analyzing the data, we found out that this cluster is very close to another cluster, which makes it difficult to be identified. However, this figure shows that after the time 38010, several prototype additions



(a) The clustering results of OnCAD

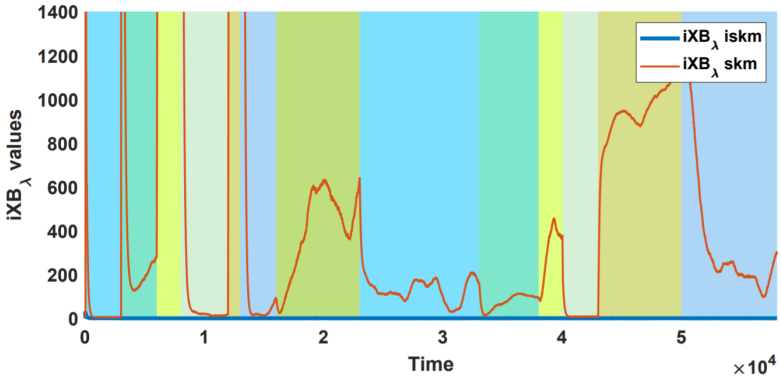


(b) The clustering results of i-OnCAD

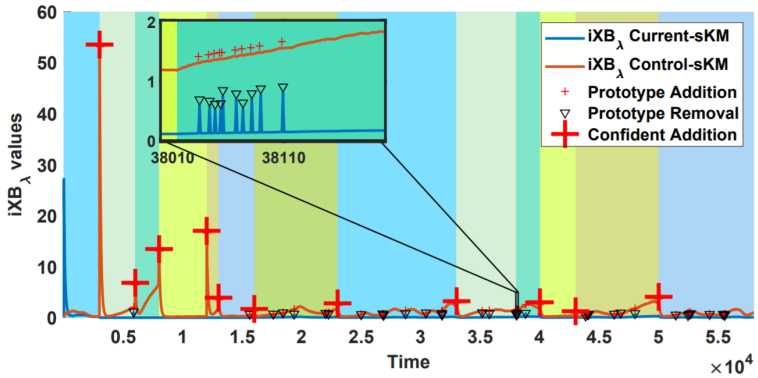


(c) The iXB_λ values of OnCAD and i-OnCAD. Different colors show different cluster contributing to the stream. The poor performance of OnCAD is reflected by having higher iXB_λ values

Figure 6.11: The clustering results of experiment 3 (real-life gas sensor dataset). Comparing the clustering results in 6.11a and 6.11b shows that OnCAD totally misses several clusters, mainly because the radius parameter could not identify those clusters in the stream. However, the control structure of i-OnCAD has correctly identified the changes in underlying clusters and alerted the clustering algorithm for adding or merging prototypes. The poor performance of OnCAD is reflected by its higher iXB_λ values in 6.11c



(a) The iXB_λ values of skM and iskM. Different colors show membership labels. The poor performance of skM is reflected by having significantly higher iXB_λ values



(b) The iXB_λ values of Current-skM and Control-skM for the iskM algorithm. Different colors show membership labels. The iskM algorithm closely monitors the iXB_λ values of both components for adding and merging prototypes. Big $+$ signs represent the additions that are not associated with a merger

Figure 6.12: The iXB_λ values of skM and iskM

Table 6.1: The external cluster validity index values for all of the experiments . NMI_{max} and ARI values of iskM and i-OnCAD are notably higher than skM and OnCAD due to the control structure that has helped them to better identify and adapt to the changes in data streams

	Cluster Validity Index	skM	iskM	OnCAD	i-OnCAD
Experiment 1	NMI_{max}	0.33	**0.82**	0.52	**0.82**
	ARI	0.17	**0.83**	0.28	**0.81**
Experiment 2	NMI_{max}	0.39	**0.71**	0.29	**0.83**
	ARI	0.25	**0.73**	0.14	**0.83**
Experiment 3	NMI_{max}	0.32	**0.67**	0.48	**0.85**
	ARI	0.21	**0.52**	0.23	**0.86**
Experiment 4	NMI_{max}	0.66	**0.96**	-	-
	ARI	0.54	**0.95**	-	-

and mergers have occurred, which demonstrates that a change is detected by iskM but it has not been significant enough for a confident addition. One solution for such scenarios is to study the data when such suspicious activities occur in iXB_λ readings. These readings can be analysed by a domain expert to interpret the behaviour of underlying clusters in the data.

In order to compare the results of the clustering algorithms on the four labeled datasets, we employ two external bCVIs, viz., the *Adjusted Rand Index* (ARI) which is a pair-counting measure in [67], and *Normalized Mutual Information* (NMI_{max}) by [84]. Both of these indices compare the ground truth labels to the labels obtained by the algorithms, and both are max-optimal (higher values indicate a better match). Table 6.1 shows the external cluster validity index values NMI_{max} and ARI on all of the experiments. The control structure has helped online clustering algorithms to better identify and adapt to the changes, which has significantly increased the performance of clustering.

6.6 Conclusion

Stream clustering algorithms that leave prototypes for their footprints need to adapt to changes in the stream by adding or removing prototypes when necessary. Current state-of-the-art algorithms require a user-defined input parameter that relates to the number of clusters in the stream, the radius of clusters, or the densities of clusters for identifying changes in the stream. In this chapter, we proposed a control structure that uses incremental Xie-Beni index values for online assessment of cluster qualities in real-time. The Xie-Beni index measures the cohesion of data points around

prototypes, and the separation of prototypes from each other to calculate a non-negative value which demonstrates how well cluster centers represent the data. We mounted this control structure on skM as a representative of online clustering algorithms. The new algorithm (iskM) comprises two skMs (Control-skM and Current-skM) that run in parallel where Control-skM is more sensitive to the cohesion of data points and Current-skM is more sensitive to the separation of prototypes. When a new cluster emerges in the stream, Control-skM identifies this change, but the decision for adding a new prototype is validated by the incremental Xie-Beni index values of Current-skM. Similarly, when two prototypes become too close to each other, Current-skM identifies this change, but the decision of merging these two prototypes is validated based on the incremental Xie-Beni index values of Control-skM. These two modules work along each other to identify and adapt to the changes in the stream. We also mounted the proposed control structure on another online clustering algorithm, OnCAD, which also improved its performance in both synthetic and real-life experiments. In the next chapter, we discuss future research directions for this technique along with the other methods that were proposed in earlier chapters.

Chapter 7

Conclusion

THE growth in IoT devices in consumer, commercial and government contexts is creating demand for machine learning in a wide range of online applications. These devices generate a potentially unbounded stream of data without the supervision of experts, hence, the majority of it is unlabelled. This means that there are no meaningful tags or labels available that describe or provide some information about the data. Therefore, the machine learning techniques that can be used for such data are mainly unsupervised methods, such as clustering. Cluster analysis aims at finding the dominant patterns in a dataset and provide meaningful insights into the unlabelled data. Another important task in unsupervised methods is detecting unexpected events or anomalies. These unexpected events may translate to problems in the system that need to be studied by an expert. For instance, in a military base an anomaly may represent an enemy intrusion, or in public transport an anomaly may represent an accident that has caused congestion.

Major challenges for clustering and anomaly detection in the IoT context are non-stationary environments, along with memory and computational resource constraints. Therefore, clustering and anomaly detection algorithms designed for such environments need to be computationally efficient. Second, as the number of dimensions in the data increases, studying the full feature space for identifying patterns becomes less effective as distance measures become less meaningful [140]. Subspace clustering algorithms aim to address this problem, but have scalability issues for large datasets and streaming environments. Moreover, current state-of-the-art subspace clustering algorithms are restricted to detecting certain classes of patterns in data streams [132, 139, 43]. Finally, patterns and clusters in such data may have various shapes, sizes, and densities. Existing algorithms for stream clustering require pre-specified parameters such as the cluster radius, or the minimum number of data points in the neighbourhood of a data point for identifying dense

regions in the data [93, 155, 3, 31]. However, finding good parameter settings requires either expert knowledge, which is expensive to acquire, or several passes over the data with different parameter values to find the best settings, which is not feasible in streaming environments. Addressing these challenges will improve the feasibility of using stream clustering and real-time anomaly detection in the IoT context.

In summary, in this thesis we have proposed efficient anomaly detection, online clustering, and subspace clustering algorithms for data stream mining. These methods can be used in non-stationary environments for identifying normal behavior and unexpected events in a monitored system. We have shown that the proposed methods are more efficient and have higher or comparable accuracy with respect to a range of state-of-the-art stream clustering and stream anomaly detection techniques.

7.1 Summary of Contributions

We first presented an efficient algorithm for identifying anomalies in a data stream in Chapter 3. In this method, we proposed a window-based data stream anomaly detection algorithm that instead of building a new profile of clusters for each window, checks if the set of defined clusters in the model can be used to explain the data in the new window. This reduces the memory complexity of the algorithm to the number of underlying clusters, rather than the number of windows or data points in the stream. We also showed that the time-complexity of the algorithm scales linearly with the number of data points, and after some time period, when all clusters are identified by the algorithm, the time required for processing each window remains constant for the rest of the stream.

In Chapter 4, we proposed a new method for real-time anomaly detection while performing online clustering. Current state-of-the-art algorithms for online clustering either do not detect anomalies or detect anomalies in a separate process in an offline manner. However, detecting anomalies in real-time is of great utility in many application domains such as intrusion detection or fault detection in industrial machinery. In this algorithm, a sliding window of data is analysed in two steps. In the first step, the membership of the newly arrived data point is calculated w.r.t the already identified clusters, where the relevant clusters are incrementally updated. In the second

step, when a new cluster emerges and the data points in the window do not belong to any clusters in the model, a batch clustering algorithm on the window is used to identify emerging clusters. However, setting the window length is a major challenge. If the window length is too small, the algorithm will not capture any statistically significant patterns in the stream, and if the window length is too large, it becomes similar to batch clustering algorithms. In this algorithm, we take into account the temporal proximity of data points as well as their spatial proximity for detecting anomalies in real-time. We use the Elliptical Confidence Region with 95% confidence to calculate the minimum window length, which is the minimum sample size to ensure that the sample mean is within a certain distance of the distribution mean, so subsequent data points update the cluster.

High-dimensional streaming time-series data clustering was addressed in Chapter 5, where we presented SCTS, to the best of our knowledge the first subspace clustering algorithm for streaming time-series. Common distance metrics such as Euclidean distance are adversely affected by the curse-of-dimensionality in high-dimensional datasets. This results in poor accuracy of clustering methods that use these metrics. However, in most cases, the data can be represented using a lower-dimensional manifold [140, 139, 43]. Therefore, finding a subset of features (dimensions) for each cluster can greatly help the accuracy of clustering. Dimensionality reduction techniques such as PCA identify the most important features for the whole dataset which is not desirable. This is because each cluster may have a unique set of features that best describe the cluster. We use PCA on a sliding window (where the window length is calculated based on the method in Chapter 4) to identify the features for each cluster that emerge and evolve over the time-series. After identifying the subspace of the cluster, we use hyper-ellipsoidal cluster prototypes to set a boundary for clusters. We showed that our proposed algorithm is capable of identifying subspaces with multiple clusters, and clusters that completely overlap but appear in different subspaces and timespans. Our experiments demonstrated that these two cases improve the quality of clusters and anomaly detection in an intrusion detection setting compared to state-of-the-art algorithms.

In non-stationary environments where clusters of data emerge, evolve and change over time, identifying these changes is crucial for stream clustering algorithms. Current algorithms use a set of input parameters as thresholds for detecting such changes in patterns. For instance, they require the radius of underlying clusters, the minimum number of points in a cluster, or density thresholds. Finding good values for such parameters requires expert knowledge, which is expensive, or

several passes over the data, which is infeasible in streaming environments. We proposed a control structure that can be mounted on other online clustering algorithms to help them through changes in streams of data. We mounted this control structure on the skM algorithm as a representative of online clustering algorithms. This control structure answers two questions: (i) when to add a cluster prototype in case of an emerging cluster? and (ii) when to merge two prototypes when two clusters become too close to each other? It comprises two skMs that run in parallel where the first skM is more sensitive to cohesion of clusters and answers the first question, and the second skM is more sensitive to separation of prototypes and answers the second question. This control structure also incorporated a forgetting factor to be a reflection of the most recent data points in the stream. To the best of our knowledge, this is the first known control structure proposed in the literature of online and stream clustering.

7.2 Future Research

We now present several future research directions motivated by this thesis, which are open challenges for data stream clustering and anomaly detection.

Behavioural anomalies:

In Chapter 3, we proposed a novel technique for identifying active clusters, i.e., clusters that contribute to the stream. These active clusters represent different states of the system being monitored. Take the temperature and luminosity of an office as an example. In this case, we expect two main states for these variables, one state for sensor readings during the day with high values, and another state for sensor readings over night with lower values. Another example is industrial machinery where the equipment includes sensors for monitoring their operation for early detection of failures and prevention of catastrophic events. This machinery also exhibits different states in their operation. For instance, assume a specific type of machinery undertakes 6 states A, B, C, D, E, and F, where the normal sequence of these states are A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D $\rightarrow$ E $\rightarrow$ F for a complete cycle. An interesting type of anomaly is studying patterns in the sequence of these active states or clusters. In our example, if a sequence such as A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ **F** $\rightarrow$ E $\rightarrow$ **D** is observed it represents an anomaly because it is expected that F be at the end of the sequence. At this level, an anomaly represents an unexpected sequence of events (active clusters), which may seem normal

when examining the data points and clusters, but becomes an anomaly when the sequence of these states are analysed.

Incremental subspace clustering for streams with multiple concurrently emerging clusters:
An assumption made in time-series data is that close data points in time are closely related in value. This means that only a single underlying cluster contributes to the stream at each point in time. The subspace clustering algorithm for streaming time-series data (SCTS) is designed based on this assumption for identifying emerging clusters by putting all unassigned data points in the window in a new cluster. However, several active clusters may contribute to the data stream, which poses several challenges for SCTS. A future direction is to extend SCTS for such environments.

New control structures with other iCVIs:
Internal cluster validity indices capture various characteristics of clusters such as cohesion or separation in different ways. Therefore, comparing their performance in various circumstances is an interesting problem. The control structure proposed in Chapter 6 is based on the well-known Xie-Beni index. A promising future direction is evaluating other CVIs such as the Davies-Bouldin index to propose a new control structure, or comparing those CVIs to determine which CVIs work better under certain circumstances.

Control structure for each cluster:
The control structure proposed in Chapter 6 is for monitoring the general process of online clustering algorithms. It monitors the cohesion of data points and separation of cluster centers to measure how well the current set of cluster centers represent the recent data points in the stream. A limitation of this control structure is that it monitors all clusters to detect changes in the stream. This becomes a challenge in a constantly changing environment where detecting the changes might be confusing for the proposed control structure. A possible extension of this work is proposing a control structure for each cluster which monitors the dispersion of data points for the cluster. This control structure guides the clusters produced by the clustering algorithm and can split a cluster into two clusters, or merge two clusters into one cluster when necessary.

Bibliography

[1] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "StreamKM++: A clustering algorithm for data streams," *Journal of Experimental Algorithmics (JEA)*, vol. 17, pp. 2–4, 2012.

[2] C. C. Aggarwal, *Data classification: algorithms and applications*. CRC press, 2014.

[3] C. C. Aggarwal, S. Y. Philip, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases*. Elsevier, 2003, pp. 81–92.

[4] 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 ata*, vol. 27, no. 2. ACM, 1998.

[5] A. Amini, T. Y. Wah, and Y. W. Teh, "DENGRIS-Stream: A density-grid based clustering algorithm for evolving data streams over sliding window," in *Proceedings of the International Conference on Data Mining and Computer Engineering*, 2012, pp. 206–210.

[6] P. Angelov, "Evolving takagi-sugeno fuzzy systems from streaming data, eTS+," in *Evolving Intelligent Systems: Methodology and Applications*. Wiley Online Library, 2010, vol. 12, p. 21.

[7] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *Proceedings of the 16th Conference on Information and Knowledge Management*. ACM, 2007, pp. 811–820.

[8] ——, "Distance-based outlier queries in data streams: the novel task and algorithms," *Data Mining and Knowledge Discovery*, vol. 20, no. 2, pp. 290–324, 2010.

[9] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the MIRAI botnet," in *Proceedings of the 26th Security Symposium Security 17*, 2017, pp. 1093–1110.

[10] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. PéRez, and I. Perona, "An extensive comparative study of cluster validity indices," *Pattern Recognition*, vol. 46, no. 1, pp. 243–256, 2013.

[11] S. Arora, P. Raghavan, and S. Rao, "Approximation schemes for euclidean k-medians and related problems," in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, vol. 98, 1998, pp. 106–113.

[12] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii, "Scalable k-means++," *Proceedings of the Very Large Data Bases Endowment*, vol. 5, no. 7, pp. 622–633, 2012.

[13] R. Benkoczi, B. Bhattacharya, M. Chrobak, L. L. Larmore, and W. Rytter, "Faster algorithms for k-medians in trees," in *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*. Springer, 2003, pp. 218–227.

[14] J. C. Bezdek, "Fuzzy mathematics in pattern classification," Ph.D. dissertation, Cornell University, 1973.

[15] ——, *A Primer on Cluster Analysis: Four Basic Methods that (Usually) Work*. Sarasota, FL., USA: First Design Publishing, 2017.

[16] J. C. Bezdek and N. R. Pal, "Some new indexes of cluster validity," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 28, no. 3, pp. 301–315, 1998.

[17] V. Bhatnagar, S. Kaur, and S. Chakravarthy, "Clustering data streams using grid-based synopsis," *Knowledge and Information Systems*, vol. 41, no. 1, pp. 127–152, 2014.

[18] A. Bielecki and M. Wójcik, "Hybrid system of ART and RBF neural networks for online clustering," *Applied Soft Computing*, vol. 58, pp. 1–10, 2017.

[19] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "MOA: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1601–1604, 2010.

[20] C. Bohm, K. Railing, H.-P. Kriegel, and P. Kroger, "Density connected clustering with local subspace preferences," in *Proceedings of the 4th IEEE International Conference on Data Mining*. IEEE, 2004, pp. 27–34.

[21] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku, "Streaming k-means on well-clusterable data," in *Proceedings of the 22nd Annual Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2011, pp. 26–40.

[22] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[23] A. Broder, L. Garcia-Pueyo, V. Josifovski, S. Vassilvitskii, and S. Venkatesan, "Scalable k-means by ranked retrieval," in *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*. ACM, 2014, pp. 233–242.

[24] M. M. Bruce, "Estimation of variance by a recursive equation," National Aeronautics and Space Administration, Washington D.C, USA, Tech. Rep. NASA-TN-D-5465, October 1969.

[25] M. Brun, C. Sima, J. Hua, J. Lowey, B. Carroll, E. Suh, and E. R. Dougherty, "Model-based evaluation of clustering validation measures," *Pattern Recognition*, vol. 40, no. 3, pp. 807–824, 2007.

[26] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the SIAM International Conference on Data Mining*, 2006.

[27] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, "Scalable distance-based outlier detection over high-volume data streams," in *IEEE 30th International Conference on Data Engineering*. IEEE, 2014, pp. 76–87.

[28] M. Capó, A. Pérez, and J. A. Lozano, "An efficient approximation to the k-means clustering for massive data," *Knowledge-Based Systems*, vol. 117, pp. 56–69, 2017.

[29] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition," *Neural Networks*, vol. 4, no. 4, pp. 493–504, 1991.

[30] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, p. 15, 2009.

[31] 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*. ACM, 2007, pp. 133–142.

[32] C. Chinrungrueng and C. H. Sequin, "Optimal adaptive k-means algorithm with dynamic adjustment of learning rate," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 157–169, 1995.

[33] J. P. Costeira and T. Kanade, "A multibody factorization method for independently moving objects," *International Journal of Computer Vision*, vol. 29, no. 3, pp. 159–179, 1998.

[34] D. R. Cox, "Note on grouping," *Journal of the American Statistical Association*, vol. 52, no. 280, pp. 543–547, 1957.

[35] X. H. Dang, V. Lee, W. K. Ng, A. Ciptadi, and K. L. Ong, "An EM-based algorithm for clustering data streams in sliding windows," in *Proceedings of the International Conference on Database Systems for Advanced Applications*. Springer, 2009, pp. 230–235.

[36] C. Darken and J. Moody, "Fast adaptive k-means clustering: some empirical results," in *Proceedings of the IEEE International Joint Conference on Neural Networks*. IEEE, 1990, pp. 233–238.

[37] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[38] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 224–227, Feb. 1979.

[39] N. Desai, K. Dhameliya, and V. Desai, "Feature extraction and classification techniques for speech recognition: A review," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 12, pp. 367–371, 2013.

[40] E. Dimitriadou, S. Dolničar, and A. Weingessel, "An examination of indexes for determining the number of clusters in binary data sets," *Psychometrika*, vol. 67, no. 1, pp. 137–159, 2002.

[41] R. Dubes and A. K. Jain, "Validity studies in clustering methodologies," *Pattern Recognition*, vol. 11, no. 4, pp. 235–254, 1979.

[42] ——, "Clustering methodologies in exploratory data analysis," in *Advances in Computers*. Elsevier, 1980, vol. 19, pp. 113–228.

[43] E. Elhamifar and R. Vidal, "Sparse subspace clustering," in *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 2790–2797.

[44] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "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*, vol. 96, 1996, pp. 226–231.

[45] J. A. Feldman and D. H. Ballard, "Connectionist models and their properties," *Cognitive Science*, vol. 6, no. 3, pp. 205–254, 1982.

[46] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[47] J. Fonollosa, S. Sheik, R. Huerta, and S. Marco, "Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring," *Sensor Actuators, B*, vol. 215, pp. 618 – 629, 2015.

[48] A. Forestiero, C. Pizzuti, and G. Spezzano, "A single pass algorithm for clustering evolving data streams based on swarm intelligence," *Data Mining and Knowledge Discovery*, vol. 26, no. 1, pp. 1–26, 2013.

[49] D. Freedman and P. Diaconis, "On the histogram as a density estimator: L2 theory," *Probability Theory and Related Fields*, vol. 57, no. 4, pp. 453–476, 1981.

[50] J. Gama, *Knowledge discovery from data streams*. Chapman and Hall/CRC, 2010.

[51] ——, "Data stream mining: the bounded rationality," *Informatica*, vol. 37, no. 1, 2013.

[52] J. Gao, J. Li, Z. Zhang, and P.-N. Tan, "An incremental data stream clustering algorithm based on dense units detection," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2005, pp. 420–425.

[53] C. W. Gear, "Multibody grouping from motion images," *International Journal of Computer Vision*, vol. 29, no. 2, pp. 133–150, 1998.

[54] A. Ghoting, M. E. Otey, and S. Parthasarathy, "Loaded: Link-based outlier and anomaly detection in evolving data sets," in *Proceedings of the Fourth IEEE International Conference on Data Mining*. IEEE, 2004, pp. 387–390.

[55] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *Journal of Artificial Intelligence Research*, vol. 46, pp. 235–262, 2013.

[56] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, 2003.

[57] S. Guha and N. Mishra, "Clustering data streams," in *Data Stream Management*. Springer, 2016, pp. 169–187.

[58] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. IEEE, 2000, pp. 359–366.

[59] W. Härdle and L. Simar, *Applied multivariate statistical analysis*. Springer, 2007, vol. 22007.

[60] J. A. Hartigan, *Clustering algorithms*. Wiley, 1975.

[61] M. Hassani, P. Spaus, M. M. Gaber, and T. Seidl, "Density-based projected clustering of data streams," in *Proceedings of the International Conference on Scalable Uncertainty Management*. Springer, 2012, pp. 311–324.

[62] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11.

[63] D. W. Henderson and E. Moura, *Experiencing geometry on plane and sphere*. Cornell University, Dept. of Mathematics, 1994.

[64] D. J. Hill, B. S. Minsker, and E. Amir, "Real-time Bayesian anomaly detection for environmental sensor data," in *Proceedings of the Congress-International Association for Hydraulic Research*, vol. 32, no. 2. Citeseer, 2007, p. 503.

[65] J. Ho, M.-H. Yang, J. Lim, K.-C. Lee, and D. Kriegman, "Clustering appearances of objects under varying illumination conditions," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*. IEEE, 2003, p. 11.

[66] N. Hubballi and V. Suryanarayanan, "False alarm minimization techniques in signature-based intrusion detection systems: A survey," *Computer Communications*, vol. 49, pp. 1–17, 2014.

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

[68] O. Ibrahim, J. M. Keller, and J. C. Bezdek, "Analysis of streaming clustering using an incremental validity index," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, 2018.

[69] O. A. Ibrahim, J. M. Keller, and J. C. Bezdek, "Evaluating evolving clusters in streaming data with modified dunn's indices," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.

[70] O. A. Ibrahim, J. Shao, J. M. Keller, and M. Popescu, "A temporal analysis system for early detection of health changes," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, 2016, pp. 186–193.

[71] C. Isaksson, M. H. Dunham, and M. Hahsler, "SOStream: Self organizing density-based clustering over data stream," in *Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2012, pp. 264–278.

[72] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[73] V. Jyothsna, V. R. Prasad, and K. M. Prasad, "A review of anomaly based intrusion detection systems," *International Journal of Computer Applications*, vol. 28, no. 7, pp. 26–35, 2011.

[74] K. i. Kanatani, "Motion segmentation by subspace separation and model selection," in *Proceedings of the Eighth IEEE International Conference on Computer Vision*, vol. 2. IEEE, 2001, pp. 586–591.

[75] S. Kazemi, S. Abghari, N. Lavesson, H. Johnson, and P. Ryman, "Open data for anomaly detection in maritime surveillance," *Expert Systems with Applications*, vol. 40, no. 14, pp. 5719–5729, 2013.

[76] E. M. Knox and R. T. Ng, "Algorithms for mining distancebased outliers in large datasets," in *Proceedings of the International Conference on Very Large Data Bases*. Citeseer, 1998, pp. 392–403.

[77] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[78] ——, *Self-organizing maps*. Springer Science & Business Media, 2012, vol. 30.

[79] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *Proceedings of the 27th International Conference on Data Engineering*. IEEE, 2011, pp. 135–146.

[80] Y. Kou, C.-T. Lu, S. Sirwongwattana, and Y.-P. Huang, "Survey of fraud detection techniques," in *Proceedings of the International Conference on Networking, Sensing and Control*, vol. 2. IEEE, 2004, pp. 749–754.

[81] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The ClusTree: indexing micro-clusters for anytime stream mining," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 249–272, 2011.

[82] R. Krishnapuram and J. M. Keller, "A possibilistic approach to clustering," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 2, pp. 98–110, 1993.

[83] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Proceedings of the Advances in Neural Information Processing Systems*, 1992, pp. 950–957.

[84] Y. Lei *et al.*, "Generalized information theoretic cluster validity indices for soft clusterings," in *IEEE Symposium on Computational Intelligence and Data Mining*, 2014, pp. 24–31.

[85] S. Li, K. Li, and Y. Fu, "Temporal subspace clustering for human motion segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4453–4461.

[86] E. Liberty, R. Sriharsha, and M. Sviridenko, "An algorithm for online k-means clustering," in *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments*. SIAM, 2016, pp. 81–89.

[87] J. Lin and H. Lin, "A density-based clustering over evolving heterogeneous data stream," in *Proceedings of the International Colloquium on Computing, Communication, Control, and Management*, vol. 4. IEEE, 2009, pp. 275–277.

[88] G. Liu, Z. Lin, and Y. Yu, "Robust subspace segmentation by low-rank representation," in *Proceedings of the 27th International Conference on Machine Learning*, 2010, pp. 663–670.

[89] L.-x. Liu, H. Huang, Y.-f. Guo, and F.-c. Chen, "rDenStream, a clustering algorithm over an evolving data stream," in *Proceedings of the International Conference on Information Engineering and Computer Science*. IEEE, 2009, pp. 1–4.

[90] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[91] C.-Y. Lu, H. Min, Z.-Q. Zhao, L. Zhu, D.-S. Huang, and S. Yan, "Robust and efficient subspace segmentation via least squares regression," in *Proceedings of the European Conference on Computer Vision*. Springer, 2012, pp. 347–360.

[92] Y. Ma, A. Y. Yang, H. Derksen, and R. Fossum, "Estimation of subspace arrangements with applications in modeling and segmenting mixed data," *SIAM Review*, vol. 50, no. 3, pp. 413–458, 2008.

[93] 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, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[94] L. Martí, N. Sanchez-Pi, J. Molina, and A. Garcia, "Anomaly detection based on sensor data in petroleum industry applications," *Sensors*, vol. 15, no. 2, pp. 2774–2797, 2015.

[95] H. B. McMahan, "A survey of algorithms and analysis for adaptive online learning," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 3117–3166, 2017.

[96] A. Meyerson, "Online facility location," in *Proceedings of the IEEE International Conference on Cluster Computing*. IEEE, 2001, pp. 426–431.

[97] G. W. Milligan and M. C. Cooper, "An examination of procedures for determining the number of clusters in a data set," *Psychometrika*, vol. 50, no. 2, pp. 159–179, 1985.

[98] H. M. Moftah, A. T. Azar, E. T. Al-Shammari, N. I. Ghali, A. E. Hassanien, and M. Shoman, "Adaptive k-means clustering algorithm for MR breast image segmentation," *Neural Computing and Applications*, vol. 24, pp. 1917–1928, 2014.

[99] P.-Y. Mok, H. Huang, Y. Kwok, and J. Au, "A robust adaptive clustering analysis method for automatic identification of clusters," *Pattern Recognition*, vol. 45, pp. 3017–3033, 2012.

[100] M. Moshtaghi, J. C. Bezdek, C. Leckie, S. Karunasekera, and M. Palaniswami, "Evolving fuzzy rules for anomaly detection in data streams," *IEEE Transactions on Fuzzy Systems*, 2014.

[101] M. Moshtaghi, J. Bezdek, and C. Leckie, "Online clustering of multivariate time-series," in *Proceedings of the 2016 SIAM International Conference on Data Mining*, Florida, USA, May 2016.

[102] M. Moshtaghi, J. C. Bezdek, S. M. Erfani, C. Leckie, and J. Bailey, "Online cluster validity indices for performance monitoring of streaming data clustering," *International Journal of Intelligent Systems*, vol. 34, no. 4, pp. 541–563, 2019.

[103] M. Moshtaghi, S. Rajasegarar, C. Leckie, and S. Karunasekera, "An efficient hyperellipsoidal clustering algorithm for resource-constrained environments," *Pattern Recognition*, vol. 44, pp. 2197–2209, 2011.

[104] H.-L. Nguyen, Y.-K. Woon, and W.-K. Ng, "A survey on data stream clustering and classification," *Knowledge and Information Systems*, vol. 45, no. 3, pp. 535–569, 2015.

[105] I. Ntoutsi, A. Zimek, T. Palpanas, P. Kröger, and H.-P. Kriegel, "Density-based projected clustering over high dimensional data streams," in *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 2012, pp. 987–998.

[106] L. O'callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-data algorithms for high-quality clustering," in *Proceedings of the 18th International Conference on Data Engineering*. IEEE, 2002, pp. 685–694.

[107] M. O'Neill *et al.*, "Insecurity by design: Today's IoT device security problem," *Engineering*, vol. 2, no. 1, pp. 48–49, 2016.

[108] N. R. Pal and J. C. Bezdek, "On cluster validity for the fuzzy c-means model," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 3, pp. 370–379, 1995.

[109] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[110] M. PhridviRaj and C. GuruRao, "Data mining–past, present and future–a typical survey on data streams," *Procedia Technology*, vol. 12, pp. 255–263, 2014.

[111] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2007, pp. 504–515.

[112] D. Puschmann, P. Barnaghi, and R. Tafazolli, "Adaptive clustering for dynamic IoT data streams," *Internet of Things Journal*, vol. 4, pp. 64–74, 2017.

[113] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[114] S. Rajasegarar *et al.*, "Elliptical anomalies in wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 6, no. 1, p. 7, 2009.

[115] S. Rajasegarar, C. Leckie, M. Palaniswami, and J. C. Bezdek, "Distributed anomaly detection in wireless sensor networks," in *Proceedings of the 10th International Conference on Communication Systems*. IEEE, 2006, pp. 1–5.

[116] J. Ren, B. Cai, and C. Hu, "Clustering over data streams based on grid density and index tree," *Journal of Convergence Information Technology*, vol. 6, no. 1, 2011.

[117] J. Ren and R. Ma, "Density-based data streams clustering over sliding windows," in *Proceedings of the Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 5. IEEE, 2009, pp. 248–252.

[118] E. Ribeiro, A. C. P. de Leon Ferreira, J. Gama *et al.*, "Minas: multiclass learning algorithm for novelty detection in data streams," *Data Mining and Knowledge Discovery*, vol. 30, no. 3, pp. 640–680, 2016.

[119] C. Ruiz, E. Menasalvas, and M. Spiliopoulou, "C-denstream: Using domain knowledge on a data stream," in *International Conference on Discovery Science*. Springer, 2009, pp. 287–301.

[120] C. Ruiz, M. Spiliopoulou, and E. Menasalvas, "C-dbscan: Density-based clustering with constraints," in *Proceedings of the International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*. Springer, 2007, pp. 216–223.

[121] M. Salehi, C. Leckie, J. C. Bezdek, T. Vaithianathan, and X. Zhang, "Fast memory efficient local outlier detection in data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 12, pp. 3246–3260, 2016.

[122] M. Salehi, C. Leckie, M. Moshtaghi, and T. Vaithianathan, "A relevance weighted ensemble model for anomaly detection in switching data streams," in *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2014, pp. 461–473.

[123] G. S. Sebestyen, *Decision-making processes in pattern recognition*. Macmillan, 1962.

[124] K. Sequeira and M. Zaki, "ADMIT: anomaly-based data mining for intrusions," in *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2002, pp. 386–395.

[125] G. Sheikholeslami, S. Chatterjee, and A. Zhang, "Wavecluster: a wavelet-based clustering approach for spatial data in very large databases," *The International Journal on Very Large Data Bases*, vol. 8, no. 3-4, pp. 289–304, 2000.

[126] M. Shindler, A. Wong, and A. W. Meyerson, "Fast and accurate k-means for large datasets," in *Advances in Neural Information Processing Systems*, 2011, pp. 2375–2383.

[127] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys*, vol. 46, no. 1, pp. 13–31, Jul. 2013.

[128] K. Sim, V. Gopalkrishnan, A. Zimek, and G. Cong, "A survey on enhanced subspace clustering," *Data Mining and Knowledge Discovery*, vol. 26, no. 2, pp. 332–397, 2013.

[129] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer security: principles and practice*. Pearson Education Upper Saddle River (NJ, 2012.

[130] H. Steinhaus, "Sur la division des corp materiels en parties," *Bulletin Polish Academy of Science and Mathematics*, vol. 1, no. 804, p. 801, 1956.

[131] S. Theodoridis and K. Koutroumbas, *Pattern Recognition and Neural Networks*. Springer, 2001.

[132] S. Tierney, J. Gao, and Y. Guo, "Subspace clustering for sequential data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1019–1026.

[133] M. E. Tipping and C. M. Bishop, "Mixtures of probabilistic principal component analyzers," *Neural Computation*, vol. 11, no. 2, pp. 443–482, 1999.

[134] R. S. Tsay, "Financial time series," *Wiley StatsRef: Statistics Reference Online*, pp. 1–23, 2014.

[135] P. Tseng, "Nearest q-flat to m points," *Journal of Optimization Theory and Applications*, vol. 105, no. 1, pp. 249–252, 2000.

[136] L. Tu and Y. Chen, "Stream data clustering based on grid density and attraction," *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 3, p. 12, 2009.

[137] J. J. Valdés and A. J. Barton, "Finding relevant attributes in high dimensional data: a distributed computing hybrid data mining strategy," in *Transactions on Rough Sets VI*. Springer, 2007, pp. 366–396.

[138] L. Vendramin, R. J. Campello, and E. R. Hruschka, "Relative clustering validity criteria: A comparative overview," *Statistical Analysis and Data Mining: the ASA Data Science Journal*, vol. 3, no. 4, pp. 209–235, 2010.

[139] R. Vidal, "Subspace clustering," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 52–68, 2011.

[140] R. Vidal, Y. Ma, and S. Sastry, "Generalized principal component analysis (GPCA)," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 12, pp. 1945–1959, 2005.

[141] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, "Density-based clustering of data streams at multiple resolutions," *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 3, p. 14, 2009.

[142] W. Wang, J. Yang, R. Muntz *et al.*, "STING: A statistical information grid approach to spatial data mining," in *Proceedings of the International Conference on Very Large Data Bases*, vol. 97, 1997, pp. 186–195.

[143] X. Wang, U. Kruger, and G. W. Irwin, "Process monitoring approach using fast moving window PCA," *Industrial and Engineering Chemistry Research*, vol. 44, no. 15, pp. 5691–5702, 2005.

[144] P. Å. Wedin, "On angles between subspaces of a finite dimensional inner product space," in *Matrix Pencils*. Springer, 1983, pp. 263–285.

[145] J. Wu, H. Liu, H. Xiong, J. Cao, and J. Chen, "K-means-based consensus clustering: A unified view," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 155–169, 2015.

[146] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin, "Security analysis on consumer and industrial iot devices," in *Proceedings of the 21st Asia and South Pacific Design Automation Conference*. IEEE, 2016, pp. 519–524.

[147] X. L. Xie and G. Beni, "A validity measure for fuzzy clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 8, pp. 841–847, Aug 1991.

[148] K. Yamanishi and J.-i. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2002, pp. 676–681.

[149] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," *Data Mining and Knowledge Discovery*, vol. 8, no. 3, pp. 275–300, 2004.

[150] C. Yang and J. Zhou, "Hclustream: A novel approach for clustering evolving heterogeneous data stream," in *Proceedings of the Sixth International Conference on Data Mining-Workshops*. IEEE, 2006, pp. 682–688.

[151] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 2009, pp. 529–540.

[152] Y. Yang, Z. Liu, J.-p. Zhang, and J. Yang, "Dynamic density-based clustering algorithm over uncertain data streams," in *Proceedings of the 9th International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE, 2012, pp. 2664–2670.

[153] J. Zhang, T. Nguyen, S. Cogill, A. Bhatti, L. Luo, S. Yang, and S. Nahavandi, "A review on cluster estimation methods and their application to neural spike data," *Journal of Neural Engineering*, vol. 15, no. 3, p. 031003, 2018.

[154] T. Zhang, A. Szlam, and G. Lerman, "Median k-flats for hybrid linear modeling with many outliers," in *Proceedings of the 12th International Conference on Computer Vision Workshops*. IEEE, 2009, pp. 234–241.

[155] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proceedings of the International Conference on Management of Data*, 1996, pp. 103–114.