DENSITY GRID BASED STREAM CLUSTERING ALGORITHM

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by
Rowanda Daoud AHMED

November 2019
İZMİR

ACKNOWLEDGMENTS

I would like to express my gratitude to all people supporting me for all the period of my thesis.

First and foremost I offer my sincerest gratitude to my family: my parents and to my brothers and sisters for their constant confidence and for supporting me spiritually throughout writing this thesis and my life in general.

I am grateful to my supervisors Assoc. Prof. Dr. Gökhan Dalkılıç, Assoc. Prof. Dr. Tolga Ayav, and Prof. Dr. Yusuf Murat Erten, for their participation in this research who supported who have supported me throughout my thesis with their endless patience, knowledge, excellent guidance which helped me get results of better quality.

I am also grateful to Prof. Dr. Mehmet Ünlütürk and Assist. Prof. Dr. Selma Tekir for being members of my thesis committee, so for their patience and support in overcoming numerous obstacles I have been facing through my research.

Nevertheless, I am also grateful to IYTE staff, especially Assist. Prof. Dr. Serap Şahin for the last-minute favors. I would like to express my gratitude to my fellow IYTE doctoral students for their feedback, cooperation and of course friendship.

It is my pleasure to express my grateful thanks to Eng. Abdallah Mekky, Peace Ambassadress. Aisha Shaqfa, Ph.D. candidates Raja Juodeh, Doaa Althalathini, Nourhan El-Dabba Abuzayed, Walaa Mdookh, Ersin Çine, Damla yaşar and Nafiye Bolat as good Palestine and Turk friends for their guidance suggestions and for supporting me spiritually throughout writing this thesis and my life in general.

Last but not the least, I would like to thank Turkey Scholarships (YTB) for accepting me in this Ph.D. scholarship.

ABSTRACT

DENSITY GRID BASED STREAM CLUSTERING ALGORITHM

Recently as applications produce overwhelming data streams, the need for strategies to analyze and cluster streaming data becomes an urgent and a crucial research area for knowledge discovery. The main objective and the key aim of data stream clustering is to gain insights into incoming data. Recognizing all probable patterns in this boundless data which arrives at varying speeds and structure and evolves over time, is very important in this analysis process. The existing data stream clustering strategies so far, all suffer from different limitations, like the inability to find the arbitrary shaped clusters and handling outliers in addition to requiring some parameter information for data processing. For fast, accurate, efficient and effective handling for all these challenges, we proposed DGStream, a new online-offline grid and density-based stream clustering algorithm. We conducted many experiments and evaluated the performance of DGStream over different simulated databases and for different parameter settings where a wide variety of concept drifts, novelty, evolving data, number and size of clusters and outlier detection are considered. Our algorithm is suitable for applications where the interest lies in the most recent information like stock market, or if the analysis of existing information is required as well as cases where both the old and the recent information are all equally important. The experiments, over the synthetic and real datasets, show that our proposed algorithm outperforms the other algorithms in efficiency.

ÖZET

YOĞUNLUK BAZLI AKIŞ KÜMELEME ALGORİTMASI

Son zamanlarda uygulamalar çok büyük veri akışları ürettiğinden, akış verilerini analiz etmek ve kümelemek için stratejilere duyulan ihtiyaç, bilgi keşfi için acil ve çok önemli bir araştırma alanı haline gelimiştir. Veri akışı kümelemesinin temel ve kilit amacı, gelen verilere ilişkin fikir edinmektir. Değişken hızlara ve yapılara ulaşan ve zamanla gelişen bu sınırsız verilerde tüm olası kalıpları tanımak, bu analiz sürecinde çok önemlidir. Şimdiye kadar mevcut veri akışı kümeleme stratejileri, veri işleme için bazı parametre bilgileri gerektirmesinin yanı sıra, isteğe bağlı olarak şekillendirilmiş kümeleri bulamama ve aykırı değerleri kullanma gibi farklı sınırlamalardan mustariptir. Tüm bu zorlukların hızlı, doğru, verimli ve etkili bir şekilde ele alınması için yeni bir çevrimiçi - çevrimdışı ızgara ve yoğunluk tabanlı akış kümeleme algoritması olan DGStream'i önerdik. DGStream'in farklı benzetilmiş veri tabanları üzerindeki performansını ve çok çeşitli kavram sapmalarının, yeniliklerin, değişen verilerin, kümelerin sayısı ve boyutunu ile aykırı verilerin saptanması dikkate alındığında farklı parametre ayarları için DGStream'in performansını değerlendirdik. Algoritmamız, borsa gibi en son bilgilere ilgi duyulan uygulamalar için veya mevcut bilgilerin analizinin gerekli olduğu durumlarda ya da hem eski hem de son bilgilerin hepsinin eşit derecede önemli olduğu durumlar için uygundur. Sentetik ve gerçek veri setleri üzerinden yapılan deneyler, önerilen algoritmamızın verimlilikteki diğer algoritmalardan daha iyi performans gösterdiğini sergilemektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In recent years , the wide applicability of streaming data due to the hardware technology advances and data deployment of data gathering devices like sensors lead to an accumulation of large amounts of transactional and web data. Therefore, stream data analysis is gaining a lot of attention in data mining research over the last decade and it can be considered as a main area of big data analysis. The era of big data and data mining is the self- evidence of creation these platforms that generate and consume massive amounts of data which are the result of the recent life developments (Nutakki and Nasraoui, 2017). Hence the need for strategies for analyzing and clustering the ever increasing online continuous stream objects which requires an up-to-date clustering view at decreasing time intervals. For that, there are some statistical techniques and conventional machine learning approaches (Guha and Mishra, 2016; Barbará, 2002; Aggarwal et al., 2003). In (Gong et al., 2017), the difference between the stream and dynamic clustering algorithms is explained. Ultimately, both return the updated clustering result in real-time. But stream clustering algorithms use strategies which to give the fresh data more weight than the outdated data after distinguishing between them. Dynamic clustering is about how to efficiently update in the underlying dataset along with maintaining data clusters (Gong et al., 2017). A new dynamic clustering algorithm is proposed by Gan and Tao (Gan and Tao, 2017), which is a DBSCAN-based algorithm that inserts and deletes points in the dataset and maintains data clusters. It is very fast in returning the updated result. In stream clustering, time stamps are inserted in each of the arriving objects as a key feature to define the freshness level of the objects to achieve cluster evolution tracking. Stream clustering decays data based on the meaninglessness levels. Most of data stream clustering algorithms are version which are designed adapting the existing solutions from static solutions, in an attempt to benefit from some strategies like the incremental capability or online-offline learning (Aggarwal et al., 2003; Bhatia et al., 2004). Stream clustering can be categorized into three main categories; prototype-based, density-based, and model-based methods (Alazeez et al., 2017). Example of the prototype-based algorithms is K-Means (MacQueen et al., 1967), which aims to partition the dataset objects into k clusters in which each object belongs to the cluster corresponds to the closest mean, and then refine these clusters iteratively. The density-based algorithms like DBSCAN (Ester et al.,

1996), groups the points which look closely alike, considering dense concentrated data points areas, and then these dense areas form the final clusters. Then the low-density areas are marked as outliers. Density-based methods have the capability of finding arbitrary shaped clusters, they are also robust to the outliers, and do not need a predefined number of clusters, but at the same time, they have some limitations like requiring memory and being time-consuming. Model-based algorithms try to find the distribution of some statistical model which is the way to find the final clusters of similar points. Expectation-Maximization (EM) (Dempster et al., 1977) is an example. A new approximate algorithm (Gong et al., 2017) is a recent further attempt to improve DBSCAN. DDCstream clustering algorithm (Li and Zhou, 2017) is an intrusion detection method based on a damped window of data stream clustering which discusses the methods of abnormal detection (Liu et al., 2016).

1.1. Stream Data Challenges

There are many data clustering methods for static data (Jain et al., 2006), and there are various methods for clustering stream data too. The initial stream data clustering paradigms suffer from several limitations like buffering for later handling or dropping some data which lead to poor-quality clustering results. These approaches deal with the stream data clustering as static clustering but in a continuous version (Guha et al., 2003). The evolving data are not taken into consideration in these paradigms and both recent and the outdated data are handled in the same way. Moving window is proposed to solve this problem (Barbará, 2002) to a certain extent. Other more recent stream clustering methods tried to solve some of these limitations and several algorithms are proposed to cluster the stream data, and we shall compare some of the density-based clustering ones such as the ones detailed in (Aggarwal et al., 2003; Cao et al., 2006; Ruiz et al., 2010), and (Kranen et al., 2011)in the related work section. Processing the endless amounts of data streams generated by social media and other resources to evaluate on the fly or identify patterns is a considerable challenge. Buffering or revisiting data while processing is not practical. Patterns in the stream data may have random order of arrival and also may be active or spread across a wider time-frame. So, temporal information is an important aspect of data streams, it gives some intimation on the current and may be future patterns (Nutakki and Nasraoui, 2017). Stream data clustering faces additional challenges beyond those the static data clustering faces. Challenges include Single pass clustering. Since the data is

continuously arriving the data analysis must be performed in an online single pass fashion. Another challenge is the Limited time. Stream analysis models must adhere to time constraints since the data stream applications naturally impose a limited time constraint and try to save computing time for clustering to be able to conform to the incoming stream speed. It is impossible to stop the stream to perform analysis to perform offline clustering, or to postpone displaying the results. In stream data analysis, in order to keep up with the stream speed, the algorithm must compute the average time between any two arriving objects, so that the data clustering cannot be more than that average time. Abiding with the time constraint ensures maintaining a current clustering model. Also, the Limited memory is an important data stream analysis challenge. As the data is unbounded and it has an endless stream, keeping every coming data object in the memory is unpractical. Naive approaches maintain all the data set points in memory, while stream clustering models take this point into consideration and respect the memory constraints. Varying time allowances is another data stream clustering challenge. It is a rarely the case that the stream data is in a constant flow state. The usual case is to be in a bursty state. In these changing processing time cases, for the stream clustering algorithm to keep up with different arrival speeds, it has to resort to the worst case which is the minimal time allowance in the stream. Another important challenge is Evolving data which is so common in today's applications. We note how much the behavior of the model totally differs over time. This leads to many interesting business application cases (Aggarwal et al., 2003). For example, consumption patterns along the year ordinary days absolutely differ from those that are seen during holidays. So, due to the possibility of changing the model underlying the data stream across the time, evolving data stream has to be a point of consideration. Concept drift, novelty, number and size of clusters and outliers should be detected as well. Like these phenomena should be clearly captured and identified by the stream clustering algorithms which introduced to gain useful knowledge from these streams in real-time. Cluster Transitions, regarding an up to date view on data distribution and the clustering result, (Aggarwal et al., 2004; Jain et al., 2006; Udommanetanakit et al., 2007) focusing on keeping an up to date record by employing an exponential decay function to decrease the influence of older data. There are several strategies in (Spiliopoulou et al., 2006) categorizing the cluster transitions in data stream into external and internal transitions and express how to address these transitions like outlier detection, time horizon, concept drift visualization, arbitrary shape clusters detection and novelty. As examples of these strategies; (Van Leeuwen and Siebes, 2008) is an approach which discusses how to detect the changes in the underlying stream by means of the minimal description length. CluStream

(Aggarwal et al., 2003) proposes pyramidal time frame approach which enables the user to view arbitrary time horizons clustering's. Tracking clusters over sliding windows in evolving data streams is discussed in (Zhou et al., 2008). Outlier detection, time horizon, arbitrary shape clusters detection, novelty, concept drift, and concept drift received widespread attention recently. There are many methods and solutions focusing on it recently; some of these solutions are in (Gama et al., 2014). During the last decade a lot of the solutions handle concept drift in supervised learning employing ensemble classifiers (Farid et al., 2013) or decision trees (Yang and Fong, 2013), while investigating concept drift solutions for unsupervised methods have started recently.

There are a huge number of clustering algorithms both for static and stream data. Earlier stream data clustering algorithms are designed as continuous versions of the static ones. We shall, therefore, describe briefly the algorithms developed for clustering static data in Section 2.1. We shall then continue with the others developed for clustering the stream data focusing on the density-based ones more than others in Section 2.2. That is because density-based clustering, clustering which depends on density-connected points or employing density function, has many merits such as discovering the clusters of arbitrary shapes, handling noise, performing calculations without relying on too many parameters, and they can do it in just one scan.

1.2. Performance Metrics and Basic Definitions

We will be exposed to some of the terms over and over again in this thesis. So, we shall first explain some theoretical notions by defining the concepts like the SPtree, Density Grids and the Characteristic Vector in Section 1.2.1. And after explaining the methodology of our proposed algorithm in details, to prove how well it performs we compared it with other related stream clustering algorithms. Regarding some performance metrics in the assessment process, so let's move over the used metrics in some brief in Section 1.2.2.

1.2.1. Basic Definitions

Definition 1.1 *SPtree is a clustered multidimensional index structure called as the segmentpage clustering (SP-clustering) for efficient sequential access. In our proposed algorithm, we used it to improve the query performance by continuous sorting the relevant*

points in contiguous related grids. Using SPTree in our density-based algorithm is important because dependency on density relies on the neighborhood relationships in growing clusters through the continuity of arriving data points, the connectedness of micro-clusters, and the convergence between them. Topological Spaces allows for the definition of concepts such as continuity, connectedness, and convergence, though accelerating and improving the clustering process afterwards.

Definition 1.2 *Characteristic Vector (CV) is a tuple (tg, tm, D, label, status), where tg is the last updating time for the grid g, tm is the last removing time, if ever, for g from grid list as a sporadic grid, D is the last grid density, label is the grid class label, and status is either sporadic or normal, which is used to test the grid kind to remove it afterward(Chen and Tu, 2007).*

Definition 1.3 *Micro-cluster for a set of d dimensional points $X_{i1}...X_{in}$ with time stamps $T_{i1}...T_{in}$ is defined as the $2*d+3$ tuple ($CF2^x$, $CF1^x$, $CF2^t$, $CF1^t$, n), wherein $CF2^x$ and $CF1^x$ each correspond to a vector of d entries. The definition of each of these entries is as follows:*

- *$CF2^x$ contains d values, these are the sum of the squares of the data values for each dimension. The p^{th} entry of $CF2^x$ is equal to $\sum_{j=0}^{n} {x^p}_{ij}^2$*
- *$CF1^x$ contains d values, these are the sum of the data values for each dimension. The p^{th} entry of $CF2^x$ is equal to $\sum_{j=0}^{n} {x^p}_{ij}$*
- *$CF2^t$ contains the sum of the squares of the time stamps $T_{i1}...T_{in}$.*
- *$CF1^t$ contains the sum of the time stamps $T_{i1}...T_{in}$.*
- *n donates the number of data points.*

(Aggarwal et al., 2003).

Definition 1.4 *Cluster feature CF is a triple summarizing, which is maintained about a cluster. It is a triple vector, which includes the number of the data points, the linear sum of data points, and the squared sum of them (Zhang et al., 1996).*

Definition 1.5 *Core-micro-cluster (c-micro-cluster) defines as CMC (w, c, r) for a group of close points pi1 ... pin with timestamps $T_{i1}...T_{in}$ (Ren and Ma, 2009).*

Definition 1.6 *Potential c-micro-cluster (p-micro-cluster) defines as PMC: p-micro-cluster at the time t for a group of close points $p_{i1}...p_{in}$ with timestamps $T_{i1}...T_{in}$ defines as (CF1, CF2, w) (Ren and Ma, 2009).*

Definition 1.7 *Outlier micro-cluster (o-micro-cluster) defines as OMC: The definition of o-micro-cluster is similar to p-micro-cluster. It is defined as (CF1, CF2, w, t_0), $t_0 = T_{i1}$ is the o-micro-cluster creation time. It is used to define the o-micro-cluster life extent. When the micro-cluster weight is less than the outlier weight threshold, $\beta\mu$, in this case we can consider it as an outlier (Ren and Ma, 2009).*

Definition 1.8 *Temporal Cluster Feature (TCF) is a temporal extension of CF with the timestamp of the most recent record for keeping cluster properties in the sliding window mode. It is defined as a $(CF2^x, CF1^x, t, n)$, which is similar to CF, and t is added as the timestamp of most recent record (Ren and Ma, 2009).*

Definition 1.9 *Exponential Histogram of Cluster Feature(EHCF) data structure is proposed to construct cluster features based on sliding window model. In EHCF only the most resent N records are considered at any time (Ren and Ma, 2009).*

Definition 1.10 *Density grids: the grid contains many data records. Each record x inside the grid has its own density coefficient, and this density coefficient decreases as the data record ages. To illustrate this concept by mathematical equations, we suppose that the data record x arrives at time t_x, so it's timestamp $T(x) = t_x$, and its density coefficient at this time is $D(x,t) = \lambda^{t-T(x)} = \lambda^{t-tc}$, where λ is the decay factor constant, $t \in (0, 1)$. Integrating from this point, so we can define the grid density at some time t as the whole summation of the density coefficients of all the records belonging to that grid. Let R(g,t) be the set of all data records belong to grid g at time t, so the density of g is $D(g,t) = \sum_{x \in R(g,t)} D(x,t)$.*

Definition 1.11 *Neighbouring Grids (ND): Consider two density grids $g1 = (j_1{}^1, j_1{}^2, ..., j_1{}^d)$ and $g2 = (j_2{}^1, j_2{}^2, ..., j_2{}^d)$, $g1$ and $g2$ are neighboring grids in the k^{th} dimension, denoted as $g1 \approx g2$, if there exists k, $1 \leq k \leq d$, such that:*

$$j_i{}^1 = j_i{}^2, i = 1, ..., k-1, k+1, ..., d; \quad (1.1)$$

$$|j_k{}^1 - j_k{}^2| = 1. \quad (1.2)$$

(Chen and Tu, 2007).

Definition 1.12 *Grid Group (GG) is a set of density grids is a grid group if for any two grids in the set there exist a sequence of grids indirectly connecting these two grids to each other by a chain of neighboring grids (Chen and Tu, 2007).*

Definition 1.13 *Inside and Outside Grids (IOG): Consider a grid g, belongs to grid group G, has neighboring grids in every dimension, then g is an inside grid in G. Otherwise g is an outside grid in G (Chen and Tu, 2007).*

Definition 1.14 *Grid Cluster (GC) is a grid group G is a grid cluster if every inside grid of G is a dense grid and every outside grid is either a dense grid or a transitional grid (Chen and Tu, 2007).*

Definition 1.15 *ClusTree is a balanced multi-dimensional indexing structure with fanout parameters m, M and leaf node capacity parameters l, L, with the following properties:*

- *An inner node nodes contains entries between m and M.*
- *Leaf nodes contain between l and L entries.*
- *The root has at least one entry.*
- *An entry in an inner node of a ClusTree stores:*
  - *A cluster feature of the objects it summarizes*
  - *A cluster feature of the objects in the buffer. (May be empty.)*
  - *A pointer to its child node.*
- *An entry in a leaf of a ClusTree stores a cluster feature of the object(s) it represents.*
- *A path from the root to any leaf node has always the same length (balanced).*

(Kranen et al., 2011).

Proposition 1.1 *Let C_1 and C_2 be two sets of points. Then the cluster feature vector CFT ($C_1 \bigcup C_2$) is given by CFT (C_1) + CFT (C_2) (Aggarwal et al., 2003).*

Proposition 1.2 *Let C_1 and C_2 be two sets of points such that $C_2 \subseteq C_1$. Then, the cluster feature vector CFT ($C_1 - C_2$) is given by CFT (C_1) - CFT (C_2) (Aggarwal et al., 2003).*

1.2.2. Performance Metrics

Clustering is unsupervised learning; it is interested in dividing the data into similar groups in the absence of class labels in contrast to supervised learning where you have the data, the class labels, and the algorithm. Supervised learning just learns a function from

the input. The absence of class labels in clustering makes the evaluation and the quality assessment more difficult and complicated than supervised classification. So, in clustering, to learn about the data helps to model its distribution and underlying structure. In this sub-section, we summarized the performance metrics we used to evaluate the results of our experiments in Section 4.

Performance metrics determine how good the obtained clustering reflects the actual data. We can classify the Cluster evaluation methods into two kinds: extrinsic methods and intrinsic methods.

Extrinsic methods (supervised) when the ground truths are available, so we can assign like score to the clustering. In Extrinsic methods, the ground truth compared against the clustering results. Here are some definitions of the used metrics in this study.

Precision-Recall Purity, Precision metrics are examples of extrinsic methods. They measure of how much the prediction is successful, especially in the case of very imbalanced classes. In information retrieval, *Precision* measures the output relevancy, i.e. the fraction of retrieved relevant documents, but *Recall* deals with the returned results and measures the fraction of the relevant documents that are successfully retrieved.

Recall refers to the sensitivity of the system, Equation (1.3).

$$Recall = True\ positive/(True\ positive + False\ negative) \tag{1.3}$$

And Precision refers to how precise of your recall, Equation (1.4).

$$Precision = True\ positive/(True\ positive + False\ positive) \tag{1.4}$$

To calculate the precision and recall, you need to sum up how many times you were right or wrong among your predictions. There are four ways of being right or wrong see Figure (1.1): True Negative: the case is negative and predicted negative too. True Positive: the case is positive and predicted positive too. False Negative: the case is positive but predicted negative. False Positive: the case is negative but predicted positive.

F1-score: (or F1-measure) can be defined as the mean or weighted average of Recall and Precision to evaluate an algorithm. It measure a test accuracy, i.e. it F1-score provides a single measurement for a system. If a machine learning algorithm is good at recall, it doesn't mean that algorithm is good at precision. That's why we also need F1 score which is the (harmonic) mean or weighted average of recall and precision to

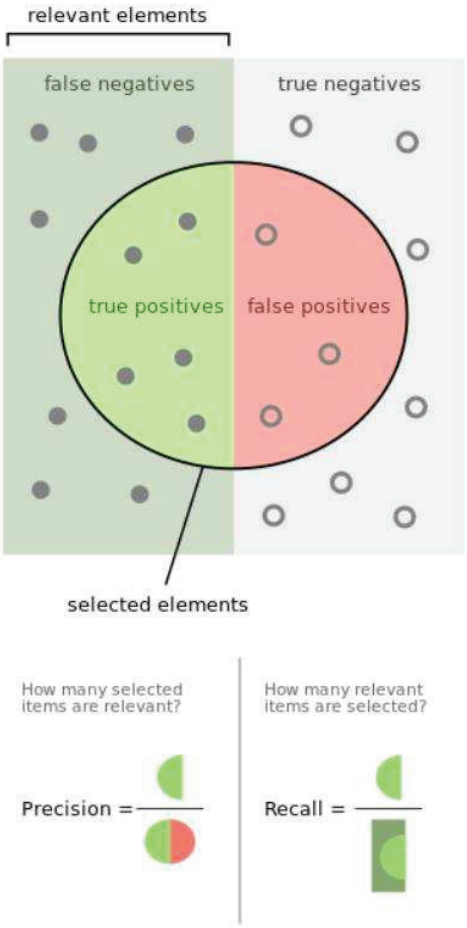


Figure 1.1. Precision-Recall (Manning et al., 2010).

evaluate an algorithm. F1 score reaches its best score at 1 and worst score at 0. The F1-score formula is:

$$F1 = 2 * \frac{p * r}{p + r} \tag{1.5}$$

Purity: measures the extent to which clusters contains a single class (Manning et al., 2010). To calculate the purity for each cluster, find out the most common class in it and count the number of its data points. After that, compute the average of overall clusters. A perfect purity score of 1 can be reached by mapping each data record to its own class. Formally, given some set of clusters C, data points N, purity can be defined as:

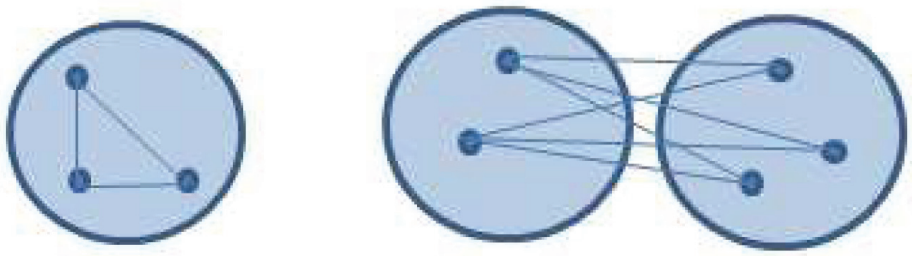
Figure 1.2. Purity example calculation.

$$Purity = \frac{1}{N}\sum_{n=1}^{C} max(number\ of\ majority\ class\ in\ cluster\ n) \tag{1.6}$$

The higher the purity is the better. A purity score of 1.0 is possible by putting each data point in its own class. For example, the purity for the following clusters in Figure (1.2), Purity =(1/20)*(3+4+6)=0.65.

SSQ: The sum of squared distances over all data points to their corresponding cluster centers. The smaller the SSQ is the better. Homogeneity: Each cluster contains only members of a single class. The Homogeneity bounds are from the lower, 0.0 to the upper bound which is 1.0, the closer Homogeneity to 1.0 is better. The higher the Homogeneity is the better. *Completeness* we suppose that all members of some class are assigned to the same cluster. The Completeness bounds are 0.0 as the lowest and 1.0 is the highest. The higher the Completeness is the better.

Intrinsic methods (unsupervised): when the ground truths are unavailable. In Intrinsic methods, how compact the clusters are, how well the clusters are separated are evaluated, i.e. Intrinsic methods are all about measuring the clustering goodness. Silhouette coefficient is an example of intrinsic methods. *Silhouette Coefficient* is a metric which is used to measure the performance of the clustering methods, it compares how the point will fit to some cluster with the assigned one by computing both values. It is proposed by Pravilovic (Pravilovic et al., 2014) to use Silhouette Coefficient in choosing the number of clusters right value, and by Rousseeuw (Rousseeuw, 1987) to measure the clustering algorithms quality. It is used in various works including the MOA framework (Bifet et al., 2010; Kranen et al., 2010). Silhouette Coefficient is a combination of



(a) Cohesion (b) Separation

Figure 1.3. Cohesion and Separation

Cohesion and Separation measures.

Cohesion: is measured by the within cluster sum of squares. It is also called Sum of Squared Error (SSE) and used as a commonly performance measure.

$$WSS = \sum_{i} \sum_{xinc_i} (x - m_i)^2 \tag{1.7}$$

Separation: is measured by the between cluster sum of squares.

$$BSS = \sum_{i} |C_i|(x - m_i)^2 \tag{1.8}$$

Where $|C_i|$ is the size of cluster i, and m is the centroid of the whole data set. The summation of cohesion and separation is constant,

$$BSS + WSS = constant \tag{1.9}$$

We can calculate the Silhouette Coefficient for both individual points and clusters as well. For individual point, the Silhouette Coefficient is defined as:

$$S = 1 - \frac{a}{b} \tag{1.10}$$

where *a* donates the averaged summed distances from some point in cluster to all points in the same cluster, while b donates the averaged summed distances of the same point to all points in the other clusters. Silhouette Coefficient bounds are between 0 and 1. The higher Silhouette Coefficient is the better.

The remainder of this thesis is organized as follows: in Chapter 2, we shall present a quick overview of the algorithms for clustering static data, then we will go in more detail over some related work in the field of stream clustering. Chapter 3 describes our new proposed stream clustering method, DGStream, in detail. In Chapter 4, we will present the results of many experiments carried out on one synthetic and many real-world datasets and show their results along with many performance metrics of our proposed algorithm, and also present the comparison of the results with several outstanding stream clustering algorithms in terms of precision, recall, F1-score measure, clustering purity, and time complexity. Finally, Section 5 will conclude our thesis.

CHAPTER 2

STREAM CLUSTERING ALGORITHMS

There are a huge number of clustering algorithms both for static and stream data. Earlier stream data clustering algorithms are designed as continuous versions of the static ones. We shall, therefore, describe briefly the algorithms developed for clustering static data. We shall then continue with the others developed for clustering the stream data focusing on the density-based ones more than others. That is because density-based clustering, clustering which depends on density-connected points or employing density function, has many merits such as discovering the clusters of arbitrary shapes, handling noise, performing calculations without relying on too many parameters, and they can do it in just one scan. We will describe some of them in the following subsections.

2.1. Static Clustering Algorithms

Several density-based clustering algorithms are developed to cluster the static data like DBSCAN (Cao et al., 2006), DENCLUE (Hinneburg et al., 1998), OPTICS (Ankerst et al., 1999), and CLIQUE (Agrawal et al., 1998). We shall describe them in some detail in th folowing sections below.

2.1.1. DBSCAN: Density-Based Spatial Clustering of Application with Noise

DBSCAN is a density-based algorithm that focuses on clustering large and noisy spatial datasets. It performs a neighborhood density analysis according to two parameters, MinPts, and Eps, so a point which has in its Eps radius at least MinPts points, is classified as the core one. A point which does not qualify as the core point but exists in the neighborhood of one core point can be classified as the border point. Any point that is neither a core point nor a border point is a noise point. The core and border points are assigned to one cluster but the noise points are not. DBSCAN can discover not only the spherical clusters but also the clusters of interwoven arbitrary shapes due to the clusters

growing according to a density-based connectivity analysis (Ester et al., 1996). DBSCAN has its limitations; DBSCAN classifies the dataset into two types of regions depending on a predefined threshold, the high density regions that are used in forming the final result of clustering as the cluster sets, and the low density regions that will be considered as noise. However, in some cases, many points identified as noises by DBSCAN may end up being meaningful data but with a low density that is under the threshold set. So, DBSCAN doesn't work well on datasets with varying densities and the high-dimensional ones.

2.1.2. DENCLUE: DENsity-based CLUstEring

DENCLUE (Hinneburg et al., 1998) is based on a solid mathematical foundation; it is like DBSCAN as it is also based on neighborhoods analysis. It figures out how one data point in the dataset can affect its neighborhood. The summation of influences of all data points is the overall density of the data space. It computes the local maxima of the density function and identifies it as density attractors that are used to assign data points to the clusters. Objects belong to related or the same cluster depending on whether they are associated with related or the same density attractor. It has been designed for clustering the multimedia in high-dimensional spatial datasets and having large amounts of noise. DENCLUE is significantly faster than DBSCAN but it depends on a large number of parameters.

2.1.2.1. OPTICS: Ordering Points to Identify Clustering Structure

OPTICS (Ankerst et al., 1999) is a phase in the clustering process; it can identify the clustering structure. It orders the points and the reachability distances in a better way to be used by other density-based algorithms afterwards.

2.1.3. CLIQUE

CLIQUE is both a grid and a density-based clustering algorithm. It is designed for clustering high dimensional spatial datasets. It partitions the dimensions into grids, the dense grids that contain at least a threshold number of data points, and the non-dense grids. Then it tries to find the embedded clusters in subspaces of the dataset (Ruiz et al.,

2010). However, the above-mentioned algorithms do not work when the data is a stream. They are applicable only to spatial datasets. As mentioned before, earlier data stream clustering algorithms have been developed to be the continuous versions of the static clustering algorithms. These approaches deal with the recent data and the outdated data in the same way. They do not consider the evolving data. Many techniques like moving window are proposed to partially solve this problem (Barbará, 2002; Babcock et al., 2003; Gama, 2010). Recently, many algorithms have been developed to cluster data streams. We intend to review in the following section some of them focusing on the density-based clustering approaches.

2.2. Stream Clustering Algorithms

The clustering algorithms that depend on user-defined parameters make the clustering difficult due to the changes which may lead clustering output result to change over time like clusters disappearing, emerging, splitting, or merging. Intuitively there is no way to previously fix the model in stream clustering. So, the old unsophisticated data stream algorithms suffer from many problems. One naive suggested solution refers to buffer stream for later handling, but this is impossible because the stream is endless. Some approaches lost valuable information due to dropping some data in order to keep up with the stream speed. Other approaches have suggested solving the stream speed adaptation, which try to scale only for the fastest stream speed, which resulted in a poor-quality clustering. Clearly these solutions mentioned above do not make the best use of the information that the stream contains and of the time available. The current stream clustering algorithms each try to solve some of the above mentioned problems. Recently, there have been different views and approaches to the data mining. For instance, (Aggarwal, 2009; Gaber et al., 2007) are proposed for classification, (Cheng et al., 2008; Dang et al., 2008; Li et al., 2008) are proposed for frequent item set mining. There are different clustering paradigms studying the data stream clustering. Earlier data stream clustering paradigms like (Guha et al., 2003, 2000; O'callaghan et al., 2002) treats data stream clustering as a continuous version of the static clustering, they are single phase divide and conquer schemes based. These data stream algorithms partition data streams into segments and based on a k-means algorithm to discover clusters in data streams. Such approaches don't consider the evolving data and they treat the recent and the outdated data the same way. To solve the evolving data problem, moving window techniques are proposed (Babcock et al., 2003). StreamKM++ (Ackermann et al., 2012) data stream clustering algorithm is a

k-means++ (Arthur and Vassilvitskii, 2007) dependent. It employs coreset constructions to computes a small weighted sample of the data stream. It uses a coreset tree for speed up. A little bit more recent approach, CluStream data stream algorithm, proposed by Aggarwal et al (Aggarwal et al., 2003). It uses a two-phase strategy. And recently many clustering algorithms are developed to cluster the stream data using the two-component technique: online-offline. Generally, in the online phase, the algorithm captures necessary summary statistics of the incoming data records. The output from the online phase is the micro-clusters that will be used in the offline phase to derive the macro-clusters via re-clustering. In the offline phase, the stream algorithm employs one of the algorithms usually used for static data clustering approaches like K-means or DBSCAN. Figure (2.1) shows the idea of using the online phase to convert the stream data points to micro-clusters and the offline phase to convert the micro-clusters to macro-clusters(Silva et al., 2013; Amini et al., 2014; Ahmed et al., 2018).

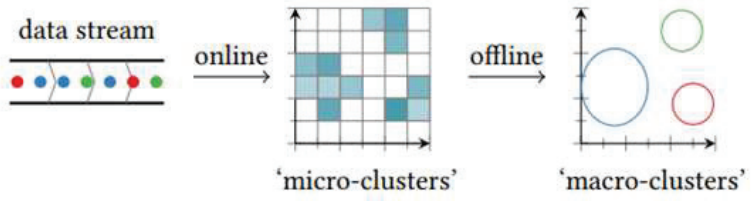


Figure 2.1. Online-offline stream clustering process(Carnein et al., 2017).

(Wang et al., 2004) clustering data stream algorithm is two-tier structure online-offline framework which is based on CluStream but in an improved offline phase. It uses an incomplete partitioning strategy. Adaptive multiple evolving data streams clustering (Dai et al., 2006), distributed data steams clustering in peer-to-peer environments (Bandyopadhyay et al., 2006), and Online-clustering of parallel data streams (Beringer and Hüllermeier, 2006), are all extensions of this work beside several applications of data stream mining (Nasraoui et al., 2006; Sun et al., 2005; Oh et al., 2005). CobWeb (Kanageswari and Pethalakshmi, 2017) is one of the first incremental systems for hierarchical conceptual clustering data. It uses a classification tree in organizing observations such that each node in the tree represents some class or concept. The class is a probabilistic

concept and it summarizes the objects classified under the node by attribute-value distributions. The classification tree is used in predicting the class of a new object or the missing attributes. CobWeb, in building the classification tree, it employs four basic operations. The operations are: merging two nodes, splitting a node, inserting a new node and passing an object down the hierarchy.

In Convex stream clustering (O'callaghan et al., 2002) data stream is processed in separate chunks and by using k-means or k-median it clusters data into k clusters in each chunk, and then it uses these clusters to generate the final clustering result. In this approach, sometimes it is a need for merging chunks results to free some space for new chunks when the case in space exceeding.

A variant approach (Spinosa et al., 2007) uses a stored list of objects besides k-means clustering in which these objects in the list do not fit the current clustering w.r.t. a "global boundary". A clustering is restarted once the list becomes large.

BIRCH (Zhang et al., 1996), maintained a hierarchical index for faster access in the very large databases. (Assent et al., 2008) represents a mapping to a frequent item set for multidimensional streams.

Kernels based approach (Jain et al., 2006) is proposed to discover arbitrary shape clusters in stream data. Alternative graphs based is proposed in (Lühr and Lazarescu, 2009), and fractal dimensions grid based is proposed in (Barbará and Chen, 2000), and (Cao et al., 2006; Chen and Tu, 2007) proposed density-based approach. None from the above approaches is capable of adapting the clustering model size to keep up with the online stream speed nor capable of delivering such a result or has the possibility of interrupting the process at any given point in time.

Density-based clustering method is a natural and attractive basic data streams clustering strategy, because it can find arbitrary and interwoven shaped clusters and can detect and handle noises. Density-based clustering methods, due to the enormous data size, don't maintain the density information for every data record. In addition to that they only need to test the raw coming data only once, in which we can call them as a one-scan method. Furthermore, they do not ask for any prior knowledge of parameters like the number of clusters k, in contrast to k-means algorithm.

DenStream is the first density-based stream algorithm we will discuss in this thesis(Cao et al., 2006). It presents core-micro-cluster and outlier micro-cluster structures to maintain, summarize clusters and distinguish the potential clusters and outliers. We will explain about DenStream briefly in Section 2.3.1.1.

D-Stream is another density-based stream algorithm which is more appropriate

and applicable for application with little domain knowledge. The stream data clustering framework, D-Stream (Chen and Tu, 2007), is a density-based clustering method that studies the relationship between all the following factors: time horizon, decay factor, and data density, so that it can generate clusters with high quality. We will explain about DStream briefly in Section 2.3.2.

The anytime idea is a very active research field in data mining and there are variant algorithms in that field, such as anytime learning (Street and Kim, 2001; Wang et al., 2003), anytime classification (DeCoste, 2002; Kranen et al., 2010; Kranen and Seidl, 2009; Seidl et al., 2009; Ueno et al., 2006; Yang et al., 2007) and top k processing (Arai et al., 2007). A wavelet-based anytime clustering algorithm has been proposed in (Vlachos et al., 2003). It is for k-means clustering of time series and not directly applied to stream data. ClusTree (Kranen et al., 2011) is a free parameter algorithm, it adapts itself to the stream speed automatically in addition to its capability of detecting novelty, outliers, and concept drift in the stream. To maintain stream summaries, it uses a self-adaptive and compact index structure. ClusTree is considered to be the first anytime stream data clustering algorithm. We will explain about ClusTree briefly in Section 2.3.3.

2.2.1. CluStream

CluStream is an online-offline Algorithm which discusses different points of view regarding data stream clustering. CluStream uses a two-phase strategy as mentioned before. In the online phase CluStream analyzes the coming data stream and stores its summary statistics using micro-clusters. These micro-clusters are temporal extensions of CF vectors and stored at snapshots in time following a pyramidal pattern in which to allow recalling the summary statistics from different time horizons. While in the offline phase it uses these statistics along with other parameters to generate final clusters. CluStream algorithm adopts the idea of streaming over many time windows, that's because streaming over many time windows gains more understanding about what is going on in the clustering process and the clusters' behaviors. So that the CluStream algorithm considered data stream clustering in one pass is not a good idea from an application point of view. Furthermore, it is too difficult in large data streams to perform such dynamic clustering over all time horizons at once. CluStream algorithm handles different data streaming ideas, each guided by requirements of real applications.

2.2.1.1. CluStream Stream Clustering Framework

CluStream has its special framework that we will discuss in this section. Being an online-offline algorithm makes it important to clarify some points. Summary statistics in the online component are temporal information prepared for the offline clustering component. This online information incrementally updated to cope with offline clustering. Secondly, online-offline clustering algorithms, in general, should determine the time interval between the times they store the online summary statistics. There is an effective trade-off between the time intervals between the process of storing the online summary statistics in one hand and clustering for this time interval which we called the time horizon. Thirdly, addressing how CluStream can use the online information for clustering, such that it gains hints for the user to set the time horizon and how to deal with the data evolving over time. The online statistics stored in a form of micro-clusters which is in a tuple feature vector (Zhang et al., 1996). Data stream clustering algorithms use feature vector due to its natural properties of addition, subtraction, and multiplications. The algorithm stores these micro-clusters in periods of time following a pyramidal pattern which provides good and reasonable trade-off between the ability to get the summary statistics stored in the micro-clusters from various time horizons and the storage requirements. These summary statistics depend on some user parameters like the granularity of clustering, time horizon or number of clusters. Definition (1.3) facilitates understanding the above-mentioned concepts.

The additive property over stream objects leads up to choose summary statistics in data stream algorithms (Arthur and Vassilvitskii, 2007). The algorithm frequently maintains the statistical summaries about the dominant micro-clusters in specific time snapshots, these snapshots have time-gap between every two snapshots. The algorithm while taking these statistics makes sure of gaining such a large number of micro-clusters and ensure such a high granularity with online updating because this increases the quality of the clusters by changing the evolved data stream. Over a particular time horizon, CluStream uses these current micro-clusters to process data and results with a higher level of clusters up to the final clustering result. For more explanation to the above concepts let us, for instance, assume that some user wants the clusters in a period of length h down from the current time t_c. Here we can benefit from the cluster feature subtractive property to find the micro-clusters stored in the interval between the current time (t_c) and the (t_{c-h}). These clusters are the historical higher-level clusters stored in the h-lengthy time horizon. To ensure approximating clusters at any time interval, it is necessary to store micro-clusters

at specific time points, i.e. snapshots every gap time interval, that's because it is impossible to store them at every time instance. In the pyramidal time frame concept which is based on the objects recency, the snapshots are categorized in different orders and stored at different levels of granularity. The orders vary from 1 to $\log T$ in a form that enables us from finding the granularity level. The passed clock time since the very beginning of the stream is referred by T. The CluStream algorithm preserved the snapshots such that at any point of time the algorithm stores only the last $\alpha + 1$ snapshots of i^{th} order, $\alpha \geq 1$ is an integer and the snapshots of the i^{th} order occurs at time horizons of α_i, so this allows for storage redundancy, that if $\alpha = 3$, so the state of micro-clusters at clock time 27 is related to orders of 0, 1, 2 and 3. That's mean each i^{th} order snapshot is taken in a time such that the stream clock time value since the beginning is divisible by α_i.

The time horizon can be approximated for large L values as close as the user wants. Also, regarding preferences, the pyramidal time window can be determined, so that it can be specified according to specific time points like the month's midst, the week's ends and so on. And to improve the granularity, small intervals can be chosen for taking the snapshots. In addition to the importance of applying strategies to eliminate the redundancy in various snapshots times.

2.2.1.2. CluStream Online Phase

CluStream algorithm prepares the online information, which is the summary statistics, which will be used by the offline phase afterward. In this phase, the online phase, there is no need to know specific user parameters because there is no dependency on any of them. Unlike in the offline phase, sometimes it is important to determine some user parameters. Evolution analysis and Horizon-specific macro-clustering are some of the offline components which use the online information. We can explain some of the details by assuming that the algorithm, at any point of time, preserves n micro-clusters. The n parameter is specified according to the main memory restriction, such that n is larger than the number of clusters but at the same time it is less than the number of the data stream points which still flow for a long time. So, the algorithm denotes the micro-clusters by $M_{c1}...M_{cn}$, each has an identified unique id once it has been created. Merging micro-clusters results with a new micro-cluster with *id* consists from the list of merged micro-clusters iIDs. At any time, the algorithm shows a result which is the micro-clusters current snapshot. And this result changes over time whenever the clock time becomes divisible by α_i and there is new object arriving from the flow data stream, where i is an

integer. Additionally, the algorithm removes any r-order micro-cluster that stored in more than α^{1+r} units past time.

CluStream uses k-means clustering algorithm in the offline phase to create the clusters as the initial n micro-clusters. To do that it uses a large number of points in which it depends on the algorithm complexity. Now and after the creation of these initial micro-clusters, CluStream starts the online process with every arriving data point in order to update the stored micro-clusters created previously. The new coming data point either it get absorbed in the nearest one or it may create its new own micro-cluster. When new point arrives, the algorithm computes the distance between the new point and the nearest micro-cluster center, and if the choice is to merge the arriving data point, the algorithm merges it with the nearest micro-cluster. Otherwise, if the new point doesn't been founded to be within the range of the closest micro-cluster regarding its radius parameter, the new data point may be categorized as an outlier or as a seed of a new micro-cluster with new unique id to distinguish it in the ahead coming steps. If creating a new micro-cluster with a new id is the decision, it leads to reduce the total number of micro-clusters by one to prepare such a space due to the memory limits constraints. We can do that by one of the two solutions. The first solution is merging two close micro-clusters. And the second solution is deleting the oldest micro-cluster after considering it as an outlier. If the solution is the second one, we must ensure deleting it safely. It is important to show that choosing the micro-cluster with the fewest number of data points to delete is not always the best choice.

2.2.1.3. CluStream Offline Phase

Micro-clusters are efficiently maintained to be as intermediate statistical representations using the stored micro-clusters' summary statistics instead of the large volume data stream. This process is an offline process that enables the user from flexibility exploring stream clusters over different horizons, so there is no one-pass requirements constraint. But at the same time, the user provides two parameters, the first one is the time-horizon h, and this is a benefit in determining the history amount needed to create higher-level clusters. The second parameter is the higher-level clusters number k, and this to determine if we can find extra detailed clusters, or if we can mind more rough clusters. Note that at each stage of the algorithm, the current set of micro-clusters depends on all the stream processing entire history since the very beginning of it. To find the clusters over a specific time horizon, we need to find the corresponding micro-clusters making use from

the additive and subtractive properties which extended from (Zhang et al., 1996) as in the property 1.1 and property 1.2. Property 1.2 helps a lot in the approximation of the micro-clusters over a pre-specified time horizon by using two snapshots at pre-defined intervals. It is important to know that every time a new micro-cluster is created, the algorithm creates a unique id for it. And whenever two micro-clusters are merged, the micro-clustering algorithm creates a list of the entire original *ids* in that micro-cluster in which we can call it as an id-list.

2.3. Clusters Evolution Analysis

In the evolving data stream, a lot of significant changes may be recorded. So, evolving data, as well as evolution analysis, are so important for a lot of business applications (Aggarwal et al., 2003). There are some important users' parameters if one wishes to be aware of these changes over specific time intervals such as a year or a week or a decade changes. For that the algorithm have input parameters $t1$ and $t2$, $t1$ is some time in the past, $t2$ is the current time, $t2 > t1$. Another parameter is the time horizon of length d in which the micro-clusters will be compared over. When we compare between the micro-clusters existing in the period $[t1 - d, t1]$ with the micro-clusters existing in the period $[t2-d, t2]$, most of the cases intuitively gives reasonable results which can help in building notes and concluding results which lead to the future decisions. For that, there are some important questions we must ask about. We should know the micro-clusters exist at time $t2$ and weren't at time $t1$, that's mean these newly created micro-clusters were created in the period between the two times $t1$ and $t2$. If there is an indictable portion of missing micro-clusters, so we can conclude that there are frequent changes in the data stream. At the same time, if there are some micro-clusters were at the time $t1$ and still at the time $t2$ and how many portions they present from the current existing micro-clusters at all, to indicate the stream stability. Algorithm store micro-clusters along with their corresponding id and id lists, this will benefit so much in following up micro-clusters information and changes details. Up to this point, we can classify the micro-clusters into three categories. The first set of micro-clusters are these micro-clusters in the interval $[t2 - d, t2]$ which not of them were in the interval $[t1 - d, t1]$. So these micro-clusters were created in the time interval between $t1$ and $t2$ and we can donate them by $M_{added}(t1,t2)$. The second set of micro-clusters are these micro-clusters interval $[t1 - d, t1]$ and not of them are in the interval $[t2 - d, t2]$. So these micro-clusters were omitted in the time interval between $t1$ and $t2$ and we can donate them by $M_{deleted}(t1,t2)$. The third set of micro-clusters are

these micro-clusters interval [$t2 - d$, $t2$] and some or all of them were in the time interval [$t1 - d$, $t1$]. So these micro-clusters were created before the time $t1$ and still to the time $t2$ and we can donate them by $M_{retained}(t1, t2)$. If this set of micro-clusters form a big portion from the size of micro-clusters, this indicates how much the stream is stable over time. So, CluStream algorithm is applied over each set of micro-clusters as own, so that it updates them and creates newly updated sets of micro-clusters depending on the evolving arrival data points.

2.3.1. Density Micro-Clustering Streams Algorithms

In a clustering data stream, it is impossible to record all the data. Therefore, it is important to discover algorithms with a single pass clustering, unknown parameters like number and size of clusters and with limited time and memory and evolving the changed data. The idea of Micro-clusters arised from this need. So, Micro-cluster is a technique in stream clustering that saves important information about the data objects in data streams, which compresses the data effectively. In this section, we illustrate the density-based clustering algorithms on data streams using micro-clusters, introducing their characteristics along with merits and limitations analysis. We will highlight those density-based algorithms that adopt and extends the micro-clusters concept (Zhang et al., 1996; Aggarwal et al., 2003). These are DenStream, C-Denstream, rDenStream, and SDStream. We will start illustrating DenStream algorithm with more details because the others based on it.

The micro-clusters concept is two-phase clustering and it is a widely known framework for data stream clustering, which split the clustering process into an online component and offline component. In the online component, the algorithm tries to capture and store the important statistics from the data stream. While in the offline component it generates the final clustering results, depending on the stored summary statistics stored in the online phase. Density concept benefits in discovering the arbitrary shaped clusters by distinguishing those dense areas from the scattered sparse areas. DBSCAN is an important algorithm used in the offline component in the density-based data stream algorithms. Due to memory limits and the unbounded flow of data stream. It is impossible to store each coming object. So, micro-cluster is a well-known strategy special for this purpose. Micro-clusters are the optimal representation for the summary statistics stored in the online component and to be used afterward for the offline component clustering. The framework for clustering evolving data streams in (Zhang et al., 1996) is the first to present the micro-cluster concept, and it was such as a feature vector representation. See

Definition (1.4) and Definition (1.5). Figure (2.2) shows the framework of micro-clusters in density-based clustering.

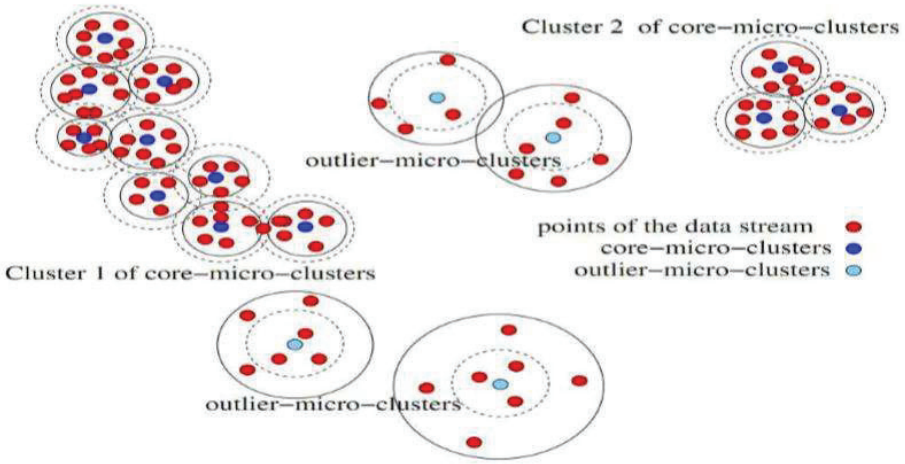


Figure 2.2. Micro-Clusters framework in density-based clustering (Ren and Ma, 2009).

2.3.1.1. DenStream

DenStream (Aggarwal et al., 2004) is a micro-cluster density-based data streams clustering algorithm. DenStream is mainly based on CluStream (Zhang et al., 1996) framework. It has two online-offline components. While density-based algorithms use the number of points in some radius as a parameter in determining the high dense areas from the low ones in the clustering process, micro-cluster density adopts a different concept. It uses the fading function (Ren and Ma, 2009) which sums up the timestamps of all points in some area. In other words, it based on neighborhood weighting areas. DenStream keeps the frequent changes and differences between clusters so that the role of these clusters changes over time along with outliers. To keep this evolved data stream, DenStream uses p-micro-cluster and o-micro-cluster (Aggarwal et al., 2004) as special structures, which are different mainly in their constrained weights. Definition (1.5), (Definition 1.6) and Definition (1.7) define respectively what are core micro-cluster, potential core micro-cluster and the outlier micro-cluster (Ng and Dash, 2010). Outlier buffer is used in the process of p-micro-clusters and o-micro-clusters for separation. DenStream

is a DBSCAN-based online-offline stream clustering algorithm. By finishing the offline component, the final clustering result can be achieved. In the online component, o-micro-clusters in outlier-buffer are all kept in separate memory spaces. When new data records arrived, they merged with the current existing p-micro-clusters and o-micro-cluster. But how is that? If the new radius of the nearest p-micro-cluster after adding the new data point to it is not more than the threshold, we add this data point to it. Otherwise, the new point is added to the nearest o-micro-clusters. And we check for any change of this o-micro-clusters weight, we can upgrade it to become p-micro-cluster if the new weight becomes higher than the specified threshold. So, we can create a new p-micro cluster and the old o-micro cluster will be deleted from the outlier buffer. Otherwise, the new data point becomes like a seed for a newly created o-micro-cluster. These o-micro-clusters in the outlier buffer may upgrade to become p-micro-cluster or they can downgrade to become an outlier. In the offline component, DenStream generates the final clusters up to the current point. Any density-based clustering algorithm, like DBSCAN, can be applied to the p-micro-clusters created in the online phase. Based on the weights of micro-clusters, the DenStream algorithm checks their weights every gap time interval to make new decisions if need. The weight must be higher than the threshold to be able to make a decision because it is all about to distinguish between the potential and real outliers micro-clusters which will change to potential micro-clusters or removed from the outlier buffer depending on its weight. That's means that the o-micro-clusters are a real outlier. At the same time, if the o-micro-cluster weight becomes more than the specified threshold it will upgrade to become p-micro-cluster. DenStream is an effective algorithm compared with CluStream, and the results from many conducted experiments proof that. DenStream execution time grows linearly with the stream proceeds.

2.3.1.2. SDStream

The SDStream (Zhou et al., 2008) is an online-offline stream clustering algorithm that depends mostly on DenStream, the previously illustrated Algorithm. It is also a density-based algorithm, so it can discover clusters with arbitrary shapes. SDStream adopts sliding windows (Ester et al., 1996) technique in clustering data points, so it regards only the data records in sliding window length and neglects the others. SDStream algorithm like DenStream, assign a weight to its micro-clusters depending on the number of points in each. So that it can distinguish between three types of micro-clusters, these are c-micro-cluster, o-micro-cluster and p-micro-cluster. These micro-clusters take the

form of Exponential Histogram of Cluster Feature (EHCF). Definition (1.9) explore the EHCF concepts (Ng and Dash, 2010), which depend on TCF in Definition (1.8) (Ng and Dash, 2010).

2.3.1.3. rDenStream

rDenStream (Ruiz et al., 2009) algorithm depends on DenStream. While DenStream algorithm has two phases, rDenStream algorithm has three phases. rDenStream's first two phases are perfectly like DenStream's two phases, while the third phase is something special for rDenStream algorithm. This third phase is all about to improve the quality of clustering by giving a second chance to the omitted data records, so that this algorithm is suitable to those applications with a massive quantity of outliers. It leads to an increase in the clustering's robustness and quality as well. rDenStream keeps these omitted records in memory to relearn from the omitted data points. So, rDenStream algorithm is considered as memory usage and time complexity compared with DenStream, but its results are better than DenStream.

2.3.1.4. C-DenStream

C-DenStream is a density-based algorithm (Ruiz et al., 2009), it uses domain or background knowledge on the stream in a proposed semi-supervised method as an extension to the base algorithm DenStream to guide the clustering process. This knowledge information is in the form of constraints. Constraints are either specific which refer to those instances that must be in the same cluster (Must-Link constraints) or popular which refer to those instances that must be in different clusters (Cannotlink constraints) (Ruiz et al., 2007). The new with C-DenStream algorithm that it applies this idea of constraints on streaming data taking cluster evolution and changes over time into consideration. So, the differences between C-DenStream algorithm and DenStream algorithm are; firstly, while DenStream uses the DBSCAN algorithm in the offline phase, C-DenStream uses C-DBSCAN (Ruiz et al., 2007) for constraints. Secondly, C-DenStream uses the constraint concept between micro-clusters when they are created, removed or maintained (Aggarwal et al., 2003).

2.3.1.5. Density Micro-Clustering Discussions

In this section, we will show in brief the density-based micro-clustering four-stream algorithms mentioned above, DenStream, SDStream, rDenstream, and C-DenStream with their cons and pros. So, the user can choose a suitable algorithm according to his own preferences and what he is interested in like the speed, the low memory usage, higher accuracy and so on. The **DenStream** algorithm puts a weight for each data point. It is a fast algorithm due to its ability to delete those outdated data points and the outliers before merging. And because it doesn't merge data points into a micro-cluster and that facilitates exploring the outliers and saves time. **SDStream** only uses the most recent data streams, while other algorithms consider the whole data stream points. It stores the micro-clusters in the EHCF data structure form. So, SDStream saves memory and it can process only with the most recent data points over the sliding window length, which means better addressing the clusters' changes and evolutions. This algorithm is suitable when the applications interested more in the recent data streams. **rDenStream** algorithm based mainly on DenStream; however, rDenStrean emphasizes handling outliers, such that it chose not to remove the data in the outlier buffer and relearn from it which result in a higher accuracy compared to DenStream algorithm. But on the other hand, it is slower than DenStream and memory usage as well. **C-DenStream** is a real applicable algorithm. It adopts the constraint concept on micro-clusters. In addition, it uses background knowledge to guide the clustering process. In this algorithm, the clusters can't have formed unless they conform to application semantics like geographical natural borders and objects.

2.3.2. DStream

DStream is a density-based framework for clustering stream data. Algorithms which based on k-means, like CluStream, require some user-specified parameters, like the number of clusters and time window, these algorithms can only find a cluster of spherical shapes and cannot handle outliers and not suitable to find clusters of arbitrary shapes. DStream addressed these issues. DStream is an online-offline algorithm, it has online and offline components. The online component continuously reads input data records and maps it into a density grid while the offline component computes the grid density, detects and removes sporadic grids from the grid-list and adjusts the clustering. DStream algorithm is a real-time clustering algorithm, it exploits the complicated relationships

between the decay factor, data density, and cluster structure, in addition, it captures the dynamic changes by adopting a density decaying technique to the data stream in order to generate clusters of arbitrary shapes effectively and efficiently. Another technique is used for detecting sporadic grids, so to remove them and get a result with improved space and time. So, DStream is a high-speed, efficiency and quality data stream clustering. To forms clusters, it partitions the data space into many density grids since it is impossible to retain all the raw data according to memory limits, and what all it needs is to work with these grids. In the DStream algorithm, every grid has a characteristic vector and then a model with discrete time steps is adopted. In Figure (2.3), the overall algorithm steps are outlined. Line 4 shows the continuously reading data stream. In every step, the while loop in lines 5-8, the algorithm reads a new record, maps this input data record into some density grid and updates its characteristic vector. This is the online component of the DStream algorithm. In lines 9-11, the algorithm performs the initial clustering which is after one gap of time. In lines 12-15, every gap time, the algorithm detects and removes sporadic grids from the grid-list and adjusts the clustering. And this is the offline component of the DStream algorithm.

```
1.  procedure D-Stream
2.      t_c = 0;
3.      initialize an empty hash table grid_list;
4.      while data stream is active do
5.          read record x = (x_1, x_2, ··· , x_d);
6.          determine the density grid g that contains x;
7.          if(g not in grid_list) insert g to grid_list;
8.          update the characteristic vector of g;
9.          if t_c == gap then
10.             call initial_clustering(grid_list);
11.         end if
12.         if t_c mod gap == 0 then
13.             detect and remove sporadic grids from grid_list;
14.             call adjust_clustering(grid_list);
15.         end if
16.         t_c = t_c + 1;
17.     end while
18. end_procedure
```

Figure 2.3. The overall process of DStream (Chen and Tu, 2007).

The grid density is always changing. It is possible to update the density of a grid only when a new data record is mapped to that grid instead of updating the density values

of all grids and data records at each time step. And the time of receiving the last data record should be recorded so that the density of the grid can be updated according to the last update time when a new data record mapped to the grid. See Definitions (1.10, 1.2, 1.11, 1.12, 1.13, 1.14)

2.3.2.1. Grid Inspection and Bgap Determination

To capture the dynamic characteristics of data streams, the DStream algorithm progressively decreases the density for each grid and data records. This is so important stage since the grids become sparse after dense and vice versa, also dense and sparse can downgrade or upgrade to transitional respectively. A sparse grid can downgrade to the transitional one if it receives some new data records. That's why the algorithm must inspect the density for each grid and adjust the clustering process every time gap (line 14 in code Figure (2.3)). Note that if we choose this time interval to be too long, data streams' dynamical changes will not be recognized well. On the other hand, if we choose this time interval to be too small, then it will result in too much computation in the offline component, which makes the work heavy and leads to a speed mismatch between the input data and the offline computations. DStream adopts the idea of setting the time interval to be the smaller of those two times intervals: The first time interval is the minimum needed time for the dense grid to downgrade to sparse.

$$\delta_0 = \left\lfloor \log_{lamda}\left(\frac{C_l}{C_m}\right)\right\rfloor \tag{2.1}$$

While the second-time interval is the minimum time for the sparse grid to upgrade to become dense grid.

$$\delta_1 = \left\lfloor \log_{lamda}\left(\frac{N - C_m}{N - C_l}\right)\right\rfloor \tag{2.2}$$

So, DStream looks at both and then chooses the small between them to ensure that the inspection process successfully detects and recognizes all dynamic characteristics of the

data stream in all grids.

$$Gap = min\delta_0, \delta_1 \tag{2.3}$$

$$Gap = min\left\lfloor \log_{lamda}\left(\frac{C_l}{C_m}\right)\right\rfloor, \left\lfloor \log_{lamda}\left(\frac{N-C_m}{N-C_l}\right)\right\rfloor \tag{2.4}$$

$$Gap = \left\lfloor \log_{lamda}\left(max\frac{C_l}{C_m}, \frac{N-C_m}{N-C_l}\right)\right\rfloor \tag{2.5}$$

2.3.2.2. Sporadic Grids Removing

The very high number of grids is a critical big challenge the DStream algorithm faces especially when the data is high-dimension andmost space grids are either empty or do not receive data stream records over and over. So, in processing and storing, only those non empty grids can be regarded and put into consideration, and the others are neglected. But the problem appears when there are outlier data in the grid space that are present in non empty grids which must be taken into consideration when processing, storing and clustering. The number of grids that contain very few numbers of data records, sporadic grids, increases extremely fast as the data stream flow at high speed, which causes an overall system slowness. So, what kind of grids the DStream mention about in code line 13 in Figure (2.3)? There are two kinds of sporadic grids whose density becomes less than some specified threshold. The first one is the sporadic grids who receive small amounts of data. This kind of grids can be deleted from the data space along with their characteristic vectors and reset their density to zero. It is experimentally proven that deleting this kind of grids doesn't affect the clustering quality. After that, if many data records from the streamflow are mapped to the previously deleted grid, this grid is added back to the grid list with a zero density. While the other kind of sporadic grids is the grids with many data records but their densities became less than the threshold by the effect of decay factor. These grids have a hope to upgrade and to become dense or transitional grids, so it is wrong to delete this kind of sporadic grids.

2.3.2.3. DStream Clustering Algorithm

Figure (2.4) and Figure (2.5) are describing initial clustering and adjust clustering process respectively. The algorithm continues reading from the flow stream of data records and computes the density of all grids in the grid list. The initial clustering process used in lines 10 in Figure (2.3) used only in the first-time gaps to generate the initial cluster. After the first gap then Adjust clustering process used in lines 14 in Figure (2.3) is executed repeatedly every time gap. Initial clustering and Adjust clustering processes are shown in Figure (2.4) and Figure (2.5) respectively. Depending on all the grids' current densities, the DStream algorithm continues removing sporadic grids that are candidates for removal and depending on that, it adjusts clustering every time gap considering only the grids in the grid list to maintain an efficient, fast and high-quality DStream algorithm.

```
1.  procedure initial_clustering (grid_list)
2.      update the density of all grids in grid_list;
3.      assign each dense grid to a distinct cluster;
4.      label all other grids as NO_CLASS;
5.      repeat
6.          foreach cluster c
7.              foreach outside grid g of c
8.                  foreach neighboring grid h of g
9.                      if (h belongs to cluster c')
10.                         if (|c| > |c'|) label all grids in c' as in c;
11.                         else label all grids in c as in c';
12.                     else if (h is transitional) label h as in c;
13.     until no change in the cluster labels can be made
14. end_procedure
```

Figure 2.4. The procedure for initial clustering.

2.3.3. ClusTree

ClusTree is a parameter-free and any-time data stream algorithm, it finds a solution for a lot of challenges like limited memory and time, maintaining results at any point

```
1.  procedure adjust_clustering (grid_list)
2.      update the density of all grids in grid_list;
3.      foreach grid g whose attribute (dense/sparse/transitional)
                is changed since last call to adjust_clustering()
4.          if (g is a sparse grid)
5.              delete g from its cluster c, label g as NO_CLASS;
6.              if (c becomes unconnected) split c into two clusters;
7.          else if (g is a dense grid)
8.              among all neighboring grids of g, find out the grid
                    h whose cluster c_h has the largest size;
9.              if (h is a dense grid)
10.                 if (g is labelled as NO_CLASS) label g as in c_h;
11.                 else if (g is in cluster c and |c| > |c_h|)
12.                     label all grids in c_h as in c;
13.                 else if (g is in cluster c and |c| ≤ |c_h|)
14.                     label all grids in c as in c_h;
15.             else if (h is a transitional grid)
16.                 if ((g is NO_CLASS) and (h is an outside
                        grid if g is added to c_h)) label g as in c_h;
17.                 else if (g is in cluster c and |c| ≥ |c_h|)
18.                     move h from cluster c_h to c;
19.         else if (g is a transitional grid)
20.             among neighboring clusters of g, find the largest one
                c' satisfying that g is an outside grid if added to it;
21.             label g as in c';
22.     end for
23. end_procedure
```

Figure 2.5. The Procedure for dynamic adjusting clusters (Chen and Tu, 2007).

in time and adapting the clustering process according to the steam speed. Besides, ClusTree uses novel descent strategies for handling the slow streams, and it uses aggregation mechanisms for handling the fast streams. Also, ClusTree puts the data stream point's ages into consideration for giving the more recent data points more importance by using such an exponential decay function. Whenever new data records come, ClusTree algorithm updates the results according to the newly arrived data records through performing single pass over them as well as it changes all the stream inter-arrival times. ClusTree algorithm is an experimentally approved that it is an efficient and effective in self-adaptive data stream clustering as well as it is scalable for giving any-time clustering results.

2.3.3.1. Self-Adaptive Anytime Stream Clustering

ClusTree is an index structure based algorithm, it can store and maintain a compact view any time the user seeks it. It is the first stream clustering algorithm for the anytime merit. ClusTree is a self-adaptive algorithm which has the ability to adapt to the coming data points stream speed automatically. And due to that, ClusTree preserves the model as whole and doesn't omit any data point, so that it inserts the coming stream object into the index structure and may merge it into the previous aggregated objects too. The structure of this section is organized as follows: In section 2.3.3.2, we define the data structure of the ClusTree along with the anytime insertion process. Section 2.3.3.3 describes older objects aging in the anytime algorithm. Sections 2.3.3.4 and 2.3.4 describe how to deal with fast streams and slow streams respectively. Section 2.3.5 summarize how ClusTree can handle different cluster shapes and transitions. The final summarization, solutions, and benefits of the ClusTree streaming algorithm are in Section 2.4.

2.3.3.2. Micro-Clusters and Anytime Insert

The data distribution in ClusTree algorithm based mainly on the idea of micro-clusters. Micro-cluster is a famous technique when the case is streaming a massive data (Aggarwal et al., 2003, 2004; Zhang et al., 1996). So that, micro-cluster which is in a tuple feature vector is the suitable representation for the data as all. Periodically, ClusTree algorithm computes the mean and variance of its micro-clusters, so that there is no need to access all stream data records or even store them due to the memory limit. The cluster feature tuple, CF = (n, LS, SS) is the form of representation instead of the store all objects, n is the number of objects, LS is the linear sum of these objects and SS is their squared sum. As stream data points flow, the cluster feature is updated incrementally, it can be considered as the true representation of the micro-clusters. So that, stream data points can be mapped to the true or the most similar micro-cluster easily. ClusTree algorithm builds a hierarchy of micro-clusters at different levels of granularity (Kranen et al., 2011). So that, it adopts hierarchical indexing structures from R-tree family (Beckmann et al., 1990; Guttman, 1984; Seidl et al., 2009) which render preserving cluster features as well as provide such an efficiently locating a right place to insert any coming stream object into a micro-cluster. For the algorithm to determine leaf entry that contains the most similar micro-cluster to the new coming object, it descends the tree down until reach

that leaf. If the similarity is enough, the algorithm updates the micro-cluster tuple values. Otherwise, it creates a new micro-cluster and forms its cluster feature (CF) with its values. But here is an important point to mention about, sometimes there is no enough time to descend the tree down to reach the leaf whenever and in anytime there is a new data point comes from the stream to insert it to the tree. So, ClusTree proposes techniques to handle such a case before reaching the tree leaf. The first technique is about keeping a global queue. But the global queue strategy has cons and pros. It is so simple and straightforward technique; however, it requires a big buffer and this needs time to deal with like time to empty that buffer, for example, resulting in a time-consuming algorithm. The second strategy maintains a global aggregate. This technique also has cons and pros. Maintaining a global aggregate solves the memory usage problem, but at the same time, it loses too much information when aggregating arbitrary objects. So, there is a third strategy proposed by ClusTree, which is interrupting the insertion process. This strategy is the local aggregate, which solves the problem of losing information, so that it preserves the clustering necessary information, as well as ensures that the algorithm can insert any newly coming data points at once. So, when there is a new object arrives, it stored in a local aggregate temporarily and this needs a space slightly larger than the space required in the global aggregate but with more accuracy. Local aggregates are a perfect solution compared with the two other discussed solutions that are the local aggregates that can be naturally integrated into the tree structure, and moreover, in the local aggregate, the time is used for regular inserts and to take a buffered local aggregate as a hitchhiker. Node entries in ClusTree hierarchy characterizes the subtrees cluster feature along with its properties, i.e. it contains the number n of aggregated objects in its subtree, their linear sum LS, and their squared sum SS, a pointer to its subtree, as well as a temporary buffer entry for local aggregates (CFs) as temporary insertions. These temporary buffers only can be existing in the inner nodes; the leaf entries don't have buffers, due to that the insertion operation happens at leaves. Figure (2.6.a) illustrates the structure of an inner and leaf entries.

R-tree, R*-tree, etc. (Beckmann et al., 1990; Guttman, 1984; Seidl et al., 2009) are examples of a multidimensional index tree structures in which the ClusTree index structures can be created and updated and only stores CFs. ClusTree in the insertion process, descents down to the closest mean subtree depending on distance calculations. The same with splitting, it outs two groups from one after minimizing the intra-group distances depending on the pairwise distances between all entries inside the node that going to be split. ClusTree has a buffer in each entry, which provides the ability to anytime

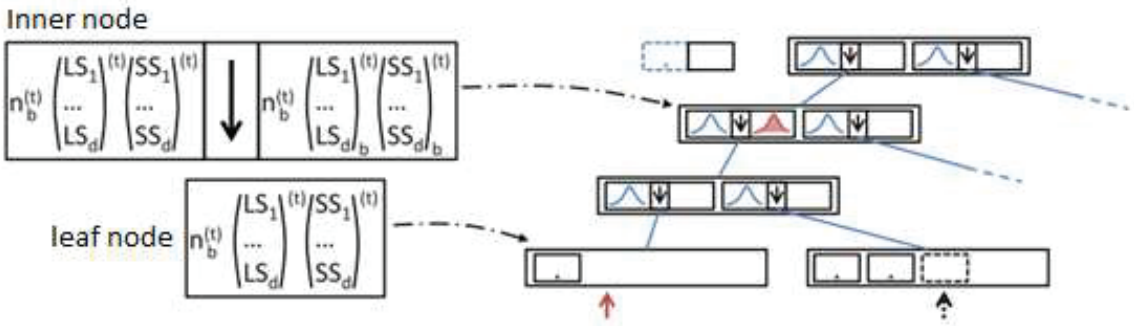


Figure 2.6. a) Inner and leaf nodes structures. b) Insertion process (Kranen et al., 2011).

stream algorithm. A buffer is a temporary place for storing either objects or aggregates that during insertion, due to the fast stream, can't reach the leaf level. The CF is stored in a buffer of a suitable entry when the insertion is interrupted. By suitable entry, we mean the entry that forms the root of the subtree in which we can descend next. Any time after that whenever this subtree is reached, the data object stores in its buffer are handled along with the hitchhiker and the insertion process continues. Figure (2.6.b) illustrates an insertion process example. Assume that we want to insert the blue dashed object next to the tree root node, and the right place for this object is the leaf node entry pointed by the dashed black arrow. In this process let us assume that the buffer of the leftmost entry on the second level is full. This full buffer belongs to the leaf node which is pointed by the solid red arrow. The insertion process descends to the second level and then it picks up the left entry buffer in its buffer CF for hitchhikers. It is shown in Figure (2.6.b) as a solid box at the right of the insertion object. Afterward, to the third level, the insertion object along with the hitchhiker descends the tree. The insertion object descends to the right entry alone to become a leaf or part from a leaf entry. But the hitchhiker stored in the left entry buffer for further future handling that's because the insertion object and the hitchhiker don't belong to the same subtrees. Taking the hitchhiker along with the insertion object and the buffer temporary storage is the point that the algorithm is an any-time stream clustering algorithm. Any time the algorithm got interrupted, it makes the best use future descents down the tree. Whenever the insertion process descents down to the target leaf node, the closest two entries are merged if there is no still time for splitting. Each leaf node entry contains an ordered unique id. It benefits in the tracking of concept drift, changes in clusters, novelty, outlier detection, etc.

2.3.3.3. Maintaining an Up-To-Date Clustering

ClusTree algorithm, to give the most recent objects more importance, uses an exponential time-dependent famous decay function $w(t) = \beta - \lambda\Delta t$, with the λ is the decay rate to control the objects weighs in such a way makes the algorithm can control the forgotten or maintaining objects more by playing with the decay rate value. As the value of the decay factor increases, the old objects forgotten in a faster way and so on. ClusTree sets $\beta = 2$. For this basis, $1/\lambda$ is the half-life of objects. So that it can keep such an up to date clustering view. Making elements of a micro-cluster feature depending on the current time t incorporates decay factor and ensures that the inner entries of the ClusTree still summarize their subtrees accurately (Kranen et al., 2011):

$$n^{(t)} = \sum_{i=1}^{n} w(t - ts_i) \tag{2.6}$$

$$LS^{(t)} = \sum_{i=1}^{n} w(t - ts_i) * x_i \tag{2.7}$$

$$SS^{(t)} = \sum_{i=1}^{n} w(t - ts_i) * {x_i}^2 \tag{2.8}$$

n is the number of unweight contributing objects, and ts_i denotes the timestamp of adding the data x_i to the CF. Cluster features are suitable for clustering stream data due to the additive and multiplicity properties (Aggarwal et al., 2004): if in some period of time from the current time (t) to ($t + \Delta t$), no object is added to a $CF(t)$, then the new value of the cluster feature is the old one at the time (t) multiplied with the decay function of the time interval (Δt) : $CF(t + \Delta t) = w(\Delta t) * CF(t)$. This property's proof found in (Aggarwal et al., 2004). Every object has its arrival timestamp and every entry has its last update timestamp too. These parameters are needed in computing the decay function. While the insertion process and descend the tree down to the leaf, all entries in the node are updated to the arrival timestamp, tx. So that all entries in the same node have the same timestamp, tx always. The new timestamp's results are:

$$es * CF = w(t_x - es * ts) * es * CF \quad (2.9)$$

$$es * buffer = w(t_x - es * ts) * es * buffer \quad (2.10)$$

$$es * ts = t_x \quad (2.11)$$

tx is the object x timestamp, $es.ts$ denotes the last update timestamp of the entry es. Every inner node in the tree structure summarizes its subtree. So, contacting the last update timestamp field in every node, besides using this formula in Equation (2.12) in weighting, ensures capturing decay with time correctly.

$$e_s * CF^{(t+\Delta t)} = \left(w(\Delta t) * \sum_{k=0}^{v_s} e_{soi} * CF^{(t)} \right) + e_s * buffer^{(t+\Delta t)} \quad (2.12)$$

As in Definition (1.15), the weighted cluster features are merely replaced with the non-weighted ones, and this means there is no need for space to store the weighted ones. In order to avoid splitting, the algorithm weighs with time, so that it just checks over the least significant entry, because it is the least entry that can contributes or even affects the clustering, and see whether it can discard it or not. If so, the algorithm discards it and subtracts its summary statistics values from its entire path up to the root. By that, it prepares a place for another future insertion entry.

2.3.3.4. Speed-up Through Aggregation (Very Fast Streams Case)

ClusTree algorithm speed-up through aggregation before insertion when it faces fast data streams. The algorithm firstly needs to determine which objects through the coming objects should be together in the same leaf to aggregate together. That will facilitate pretty much when descending the tree down to insert a new object. ClusTree creates different aggregates due to that most of the time the incoming objects are dissimilar and so ClusTree creates different aggregates each for the most similar incoming objects. So, till now the objects in the same aggregate are mostly similar. When the stream is so fast, the algorithm stores the incoming objects in the closest aggregate, so that to be not more

than some specified distance threshold with respect to the distance to the mean.

2.3.4. Making Better Use of Time Through Alternative Descent Strategies (Slow Streams Case)

ClusTree algorithm suggests alternative techniques for non-interrupting inserting objects into micro-clusters as ways how to traverse the tree structure and explores the down paths. Using these alternative descent strategies, the concept of anytime clustering is completed and we can use any remaining time to improve the insertion process. In slow streams, the insertion process just follows a single path. It starts by picking the leaf node with the smallest distance between this leaf child and the insertion object. And after reaching the leaf, we still think about how to exploit the remaining time. One most important strategy is the depth-first strategy. Almost in slow streams, there is no such interruption while inserting some incoming objects. So, the process in the depth-first strategy continues to traverse down the tree in some path regarding the similarity between the object to be inserted and the reachable leaf. And considering the time we have the insertion continues down as far as we can until reaching the leaf in Figure (2.7).

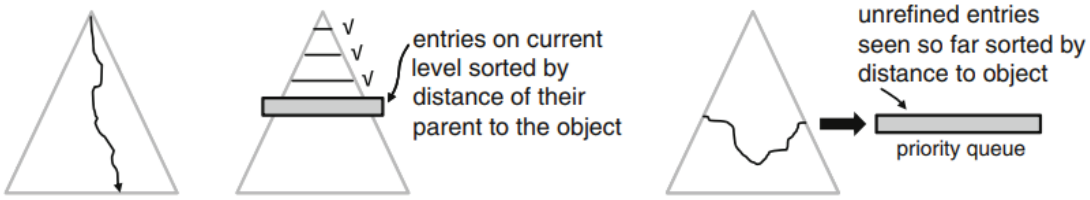


Figure 2.7. Descent strategies. a) Depth first. b) Priority breadth first. c) Best first (Kranen et al., 2011).

This result is with a good micro-clusters resolution. And if there is still time, we can do a lot of things, like for instance, we can use it anytime another incoming object to insert. Furthermore, we can adapt the model size to the stream speed by splitting the leaf. In slow streams, the tree continues to grow due to mostly reaching the tree leaves, unlike in the fast stream case, we use aggregation strategy to avoid inserting on the higher levels. But we know that the stream clustering algorithms have to handle the

limited memory challenge, so the tree growth due to the time available in slow streams is not ideal and there is a limitation on this grows. So, the depth-first strategy is good for algorithm ideality when reaching the most down level, leaves. Actually, reaching the leaf level needs just a few steps of computations and that's why ClusTree is considered a fast anytime stream clustering algorithm. ClusTree suggests other strategies to deal with the slow streams like Best first traversal, Priority breadth-first traversal, and Iterative depth-first descent, Figure (2.7). The most difference between these three descent strategies that the depth-first descent strategy stops once the maximal model size is obtained and a leaf is reached, but the Priority breadth-first and best first make use of the additional remaining time to check if there are any alternative insertion options. So, the best first traversal and priority breadth-first traversal has the same drawback which is depending on how soon the algorithm is interrupted and on how often it has to go back and continue from upper-level nodes. The algorithm might buffer the object at the upper levels and it still there. But in depth-first strategy, the algorithm is mostly able to reach leaf level nodes. According to the previous analysis, ClusTree suggests an alternative descent strategy which is like a compromise between the three previous strategies. It is the Iterative depth-first descent strategy. In this strategy, the algorithm tries to reach leaf level, and if still there any more time, the algorithm uses it to validate the taken decisions. Due to that ClusTree algorithm adopts the last strategy, Iterative depth-first descent, so we will focus on it ahead.

The idea of iterative depth-first descent strategy is starting with the original version strategy which is the depth-first descent strategy. As shown in Figure (2.8), upon reaching leaf level, and as the time permits, the algorithm evaluates the alternatives taken decisions at the nodes along the depth-first path iteratively. Start the tree descending down, Figure (2.8.a). Assume the insertion process is not interrupted. So, it returns back to the first level and from the previously chosen entry starts descending the tree down. Up to this point as in Figure (2.8.b), we have more two alternatives leaf node candidates in which we can choose the best from the three options to insert the newly coming object, Figure (2.8.c). And again as time permits, we can repeat the process as a recursive by considering the chosen root child, the chosen node at the second level, to behave as the new current root on the current best path so far, Figure (2.8.d). The process continues like that, Figure (2.8e-f), until no more unchecked siblings on the path remains or any interruption happened. On interruption case, the algorithm buffer/insert and update as all other strategies do.

Iterative depth-first descent does at most $\log_2 n$ comparisons. As an example, for 50,000 micro-clusters and setting the fanout by 3, it needs only like 100 comparisons and

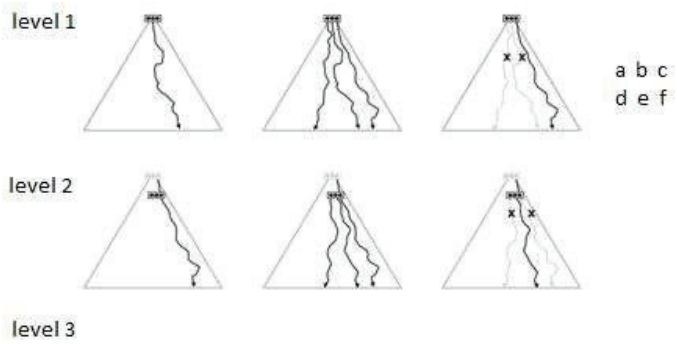


Figure 2.8. Iterative depth first descent (Kranen et al., 2011).

it is such an incredible contrast to 50,000 comparisons in (Aggarwal et al., 2003; Cao et al., 2006). So, ClusTree algorithm uses further optimization of inserts accounts for very long-time spans and aggregation for very short time spans per object (Kranen et al., 2011).

2.3.5. Cluster Shapes and Cluster Transitions

Up to this point, as the algorithm stores in the tree leaves a set of cluster features. So, by that, we can say that we finished the online component of the ClusTree algorithm. And now it is the time to apply any static algorithm to these cluster features to get the final clustering result. If we want to explore such an arbitrary shapes clusters, so (Cao et al., 2006) is for density-based clustering and so (O'callaghan et al., 2002) for k-center clustering and we have various kind of algorithms to apply in the offline mode. The most interesting thing with the ClusTree compared to others (Aggarwal et al., 2003, 2004; O'callaghan et al., 2002; Cao et al., 2006) that in its online component try to preserve such a very big number of cluster features, so that it prepares a good input for the offline clustering component. There are several strategies suggested by details in (Spiliopoulou et al., 2006), can process cluster transitions like outlier detection, concept drift detection, and novelty. Furthermore, there are many of these techniques we mentioned in the introduction in Chapter 1. ClusTree can apply any from these approaches to its output clusters. So, ClusTree algorithm can be used to detect arbitrary shaped clusters, outlier detection, concept drift detection, novelty and time horizon.

2.4. ClusTree Algorithm Conclusion

Here is a summary for all ClusTree data stream clustering algorithm techniques and solutions supported by a flowchart, in Figure (2.9), for all algorithm steps using depth-first descent strategy in slow streams. So, the strategies and techniques are as follows:

- *Tree hierarchical data structure* is to let the insert process to be logarithmic and so decrease the further computations.
- *Buffering* is to adapt the tree size, in addition to enabling anytime processing advantage.
- *Decay factor function* is such an exponential function to age older objects and so it gives the recent objects more importance.
- *Aggregation* is to speed-up clustering when the stream is fast.
- *Descent strategies (the depth-first):* let the algorithm better use the available time when the stream is slow beside insertion optimizing.
- *Cluster features* facilitate dealing with cluster transitions like outlier detection, concept drift detection, and novelty.

Here is a summary for all ClusTree data stream clustering algorithm techniques and solutions supported by a flowchart, in Figure (2.9), for all algorithm steps using depth-first descent strategy in slow streams. So, the strategies and techniques are as follows:

- *Tree hierarchical data structure* is to let the insert process to be logarithmic and so decrease the further computations.
- *Buffering* is to adapt the tree size, in addition to enabling anytime processing advantage.
- *Decay factor function* is such an exponential function to age older objects and so it gives the recent objects more importance.
- *Aggregation* is to speed-up clustering when the stream is fast.
- *Descent strategies (the depth-first):* let the algorithm better use the available time when the stream is slow beside insertion optimizing.
- *Cluster features* facilitate dealing with cluster transitions like outlier detection, concept drift detection, and novelty.

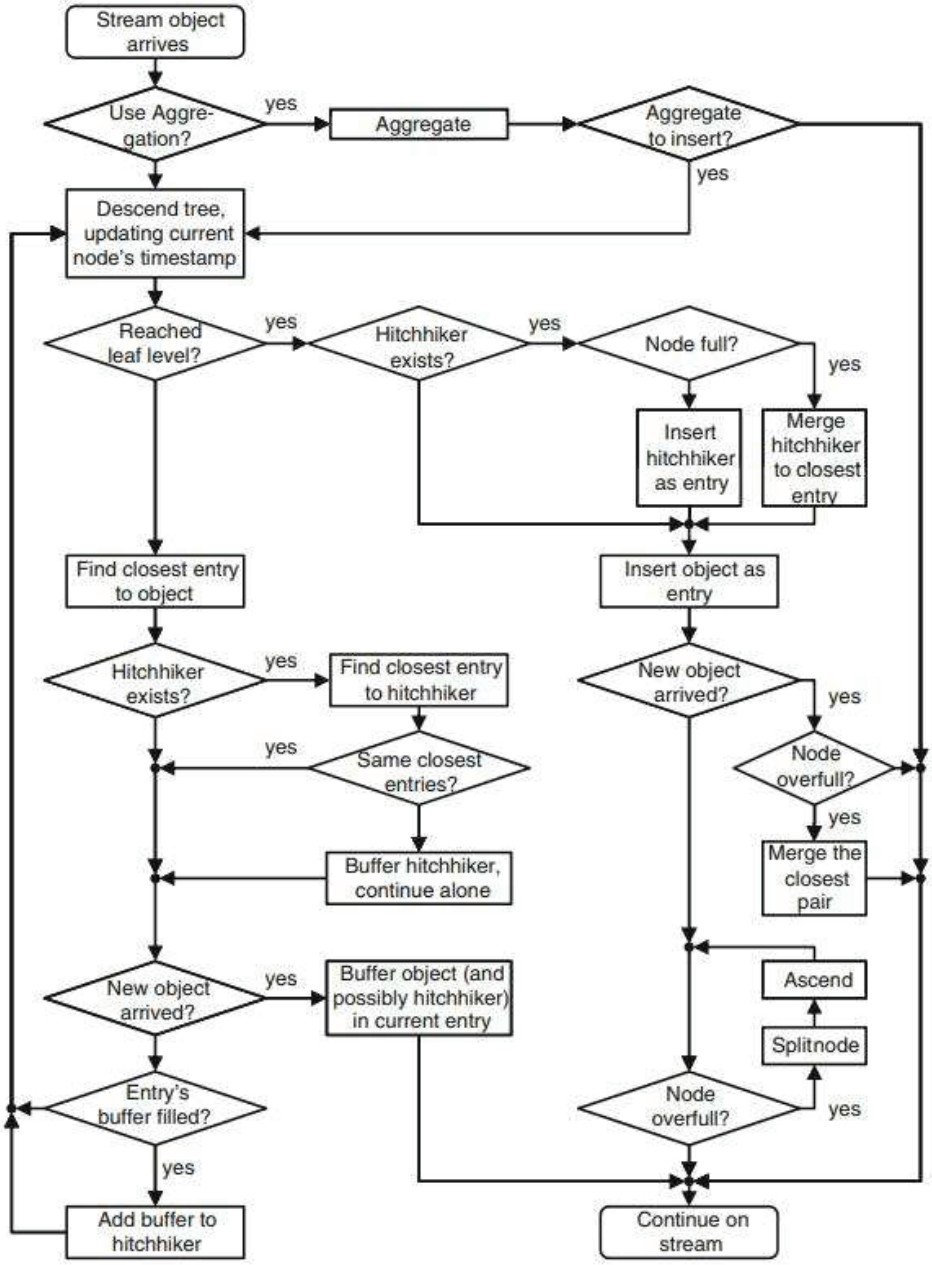


Figure 2.9. Flow chart of the ClusTree algorithm (Kranen et al., 2011).

CHAPTER 3

OUR PROPOSED ALGORITHM: DGSTREAM

DGStream algorithm assumes special architecture to cluster such unlimited data records. Like most stream algorithms, DGStream also assumes a model with a discrete-time step model, where every incoming record is labeled by an integer timestamp 0, 1, 2… n. The timestamp indicates the record arrival time. As the online-offline approach has been integrated successfully with many stream clustering algorithms (Cao et al., 2006; Chen and Tu, 2007; Kranen et al., 2011), DGStream has an online-offline processing framework as well. In the online phase, it uses feature vectors represented by a micro-cluster for each grid to dynamically maintain the necessary information about the uninterrupted arriving data records. While in the offline phase, DGStream employs a DBSCAN algorithm to benefit from its speed and to improve the running time. And it depends on grids to reduce the time complexity and accelerates the speed once more (Mekky, 2016; Alhanjouri and Ahmed, 2012). DGStream also employs a decay function mechanism to accurately reflect the stream evolution process. In addition, it uses a mechanism to delete the sparse grids to maintain processing only with a limited number of dense grids, which saves both time and memory of the system. DGStream also employs a mechanism to get rid of the noise and to handle outliers. We will see all the steps that DGStream follows in the following subsections.

3.1. Dataset Input and Standardization

Standardization of the features of the dataset is a general requirement for many data mining algorithms. It aims to rescale the distribution of data values; i.e. make the data in the dataset dimensionless, though it helps in defining data in some standard indices. So, in our algorithm, it is necessary to standardize the datasets because we are going to calculate the similarity, dissimilarity and a number of associated performance metrics of the resulted clusters after the clustering process. Z-score and minimum-maximum (or normalization, or min-max scaling) are popular examples for standardizations. In implementing DGStream, we used min-max standardization that maps the minimum value to 0, and the maximum value to 1. This type of scaling gets the standard deviations smaller,

which can reduce the effect of the outliers. Since the stream data distribution is almost non-stationary, i.e. it changes over time, which is also known as concept drift; our algorithm detects and considers these changes through the damped window model. To deal with this phenomenon, DGStream assigns the most recent incoming data points to higher weights than the weights of the older points. These weights exponentially decrease as the time goes via an employed decaying function. The density-based algorithms in (Cao et al., 2006; Chen and Tu, 2007; Isaksson et al., 2012) also adapted this model. Regarding non-stationary stream, DGStream decides to delete or create some grid according to the overall sum of all weights of data points in that grid. So that if one grid keeps receiving new data points the weight of the grid will be high because of the high weight of the new data. In case the grid does not receive any new data and its data points age over the time to become below some threshold, DGStream decides to delete this grid. In this way, DGStream's grids can be adapted to support the nonstationary data stream.

3.2. Divide the Multi-Dimensional Data Stream into Grids

DGStream divides the multi-dimensional space of the input data into density grids; we used this technique because it is impractical to maintain all the raw data. These small grids each has its density which is associated with its data records counted in it (Alhanjouri and Ahmed, 2012). And after that, the clustering process keeps these density grids and deals with each grid as a local unit to output the final cluster set. Figure (3.1) shows how the density grids can be used between the online and offline phases to cluster the data streams.

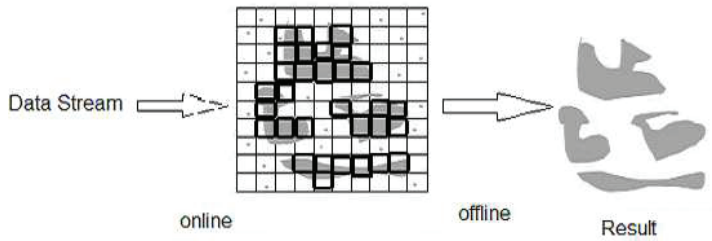


Figure 3.1. Explanation art of using the density grids in stream clustering.

3.3. Choosing Representative Points from the Density Grids

In the clustering step, instead of taking all the data points to process together, it is better to choose a set from them to represent the whole data stream we want to process. As in CURE clustering algorithm (Guha et al., 2001), it adapts the idea of choosing points from each cluster which are well scattered and can represent the cluster. After this process, it shrinks them towards the mean of the cluster by some fraction to mitigate the outliers' effects. Using representative points in clustering helps in identifying both spherical and non-spherical clusters and speed up the clustering process. Therefore, DGStream uses the same principle of choosing well-scattered representative points to represent all the read time horizon bunch of objects such that the chosen representative points attempt to capture the physical shape and the geometry of the dataset. Choosing representative points instead of all the data points they represent in DGStream, provides many benefits. It saves execution time because this leads us to deal only with these representative points instead of all the data they represent. For example, in the case of computing the distance between two clusters, the only needed distance to compute is the distance between the closest pair of representative points from each cluster. It also saves the system memory because we need only to store the representative points as input to the clustering algorithm. In this regard, there are other techniques to do this like the constructions in (De Silva and Carlsson, 2004). For instance, the lazy-witness construction robustly computes topological invariants of geometric objects. It samples the dataset and uses only a comparatively small subset point cloud that can accurately capture the dataset shape. It firstly selects landmark points from the dataset randomly. On the other hand, for achieving more spaced points, it may select this subset by performing a sequential maximum-minimum selection such that selecting the point that maximizes the minimum distance to all the selected points chosen so far.

As we said; it is important for the chosen representative data points to capture the data stream from which they are chosen from, i.e. the original stream and the chosen representative set of points must have the same shape. It is clear from Figure (3.2) that the black points that are used in the pre-clustering process are representing the input stream. Moreover, every time we read a number of examples from the incoming stream, according to the time horizon parameter, we choose well-scattered data points from the read data to represent it and continue repeating this process as the stream flow over time. The non-chosen data points from the stream will be labeled to the resulted clusters, as we will see later in this study. This step benefits in giving our algorithm a good time improvement

(Mekky, 2016; Alhanjouri and Ahmed, 2012), as depicted in the experimental results section.

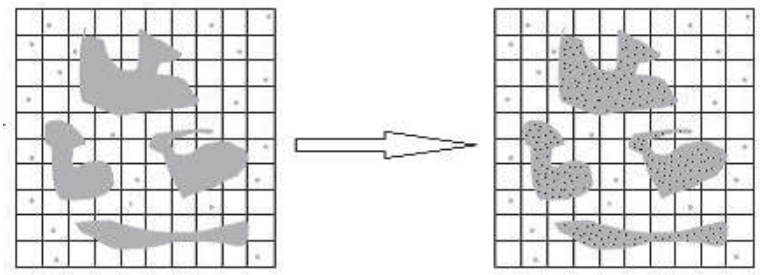


Figure 3.2. Black points are the representative points in each cell.

3.4. DGStream Clustering Process

Figure (3.3) outlines the overall DGStream algorithm. First, the algorithm reads a large number of normalized data points, and then it chooses a number of points to represent them. The large number must be enough to form such initial cluster set, it chosen about 4000 data points from the synthetic dataset in the first experinment, and 5000 points in other real-world datasets. After that, DGStream can build the SP tree of density grids and computes the initial cluster set by clustering the tree leaves. Then, it updates the characteristic vectors of the clustered grids from their initial values. Depending on the values stored in the characteristic vectors, DGStream classifies the grids to dense and sparse ones. The other points which are not chosen are labeled to the output clusters using some strategy for labeling the points to the best clusters they can belong. After that, whenever some data record arrives, the online phase of DGStream reads and maps it to the most suitable density grid in the SP tree, and accordingly updates the grid's characteristic vector values. While in the offline phase, at every pre-specified gap time, DGStream computes the densities of the grids and checks out if there is any sparse grid upgrade to become dense or if there is any grid that must be marked as a sparse grid to be deleted afterwards. The cluster set is checked and corrected dynamically by calling MainClustering method indicated in Figure (3.4).

To hold the dynamic characteristics of data streams, DGStream algorithm progressively decreases the density for each dense grid over time if it does not receive any data

records. This is an important stage since the dense grids may become sparse and vice versa. A sparse grid can be upgraded to become dense if new stream objects are mapped to it. That is why the algorithm must inspect the density for each grid and depending on that, it calls the MainClustering procedure at every gap time to adjust the final cluster set result (line 4 in the code in Figure (3.3)). The grid density is always changing. DGStream updates the grid's density only in the case the grid receives new input data records instead of updating all data records' weights and therefore the SP tree grids' characteristic vectors as well at each time step. The time of receiving the last data record, which is the time of updating the density of the grid which received that record, should be recorded to be the last update time of that grid which is considered when a new data record is mapped to the grid (Chen and Tu, 2007). Following this step saves $\theta(N)$ to $\theta(1)$ in running time, which means that it improves time efficiency since N, the number of grids, is large. Additionally, this leads to memory saving since there is no need to resave all the densities and all the corresponding timestamps of all records of the updated and not updated grids. What we need to save for each grid is the characteristic vector.

```
1. procedure DGStream
2.    initialize an empty SPTree of grids;
3.    read huge number of records and map them to the SPTree.
4.    do MainClustering(SPTree)
5.    while data stream is active do
6.          read h number of records xi = (xi1, xi2, · · ·, xid); 1<i<h
7.          determine the density grid gi that contains xi; for all xi
8.          if (gi not in SPTree) insert gi to the SPTree
9.          update the characteristic vector of gi
10.         delete sparse grids from SPTree
11.         do MainClustering(SPTree)
12. end while
13. end procedure
```

Figure 3.3. DGStream algorithm pseudocode.

3.5. Removing Sparse Grids

The very high number of grids is a critical big challenge DGStream algorithm faces especially when the data has high-dimensions. And since most of the grids are either empty i.e. contains very few number of data records or do not receive stream data records

```
1. procedure MainClustering(SPTree)
2.     choose well scattered representative points that capture the
       shape of the original dataset grids;
3.     put the remaining grids in a labeling grid list for post clustering
4.     employ DBSCAN on the chosen SPTree grids to create clusters
5.     map the records in the labeling grid list to the outputted clusters
6.     do post processing
7. end procedure
```

Figure 3.4. MainClustering method pseudocode in DGStream algorithm.

for long periods, the number of these sparse grids increases extremely fast as the data stream flows in a high speed, which causes an overall system slowness. So, the solution is to detect the grids whose density become less than some specified threshold due to small data input and remove them afterwards. Only the dense grids taken into consideration in processing and storing. The other sparse grids are neglected and removed afterwards. After that, if one removed grid receives a number of records, it will be added back to the SPTree grids but with a zero density in a hope to be upgraded to a dense one.

3.6. Labeling All Points to the Resulted Cluster Set

As it has been mentioned before, the clustering process occurs only on the well-chosen data points from the incoming stream after each time horizon. So, now is the time to do the labeling step which works with the rest of the not-chosen stream data points in placing each to the existing point to the most suitable or similar macro-cluster in the resulted macro-clusters so far. Each data point is assigned to the macro-cluster that contains the closest representative point to this one. After doing the labeling step, all the stream data points will be allocated to macro-clusters. Additionally, there is a post-processing step that specialized in merging and deleting such macro-clusters. That is macro-clusters with the same density and close enough to each other if found, they will be merged in one macro-cluster in the post-processing step. In addition, when there is any macro-cluster whose weight is lower than some specified threshold value, it will be deleted from the cluster set and considered as an outlier.

3.7. Handling Outliers

Generally, the datasets have outliers as a result of the problems that may be faced while entering data or errors in the measurement process. The distances between the outlier points and the nearest micro-clusters are high and more than the specified threshold in the DBSCAN offline algorithm. DGStream detects outliers and while the algorithm continue reading data points from the stream if some outliers near each other form such a dense grid with weight more than the specified threshold value it is upgraded to become a new micro-cluster. Otherwise, if its weight becomes less and less until become less than some threshold due to the decay factor aging, DGStream safely deletes it without degrading the algorithm quality. Handling outliers is a very important step to finish the clustering process in the data stream clustering to save both time and space of the system.

3.8. DGStream Clustering Stability

It is attractive to use stability-based principles when we want to choose our models. Interestingly, it does not require a specific model to be applied to, but it can be applied to any clustering algorithm. One could intuitively assume that clustering stability is very much related to simple solutions that have the most stable parameters, but this is not necessarily true. Many studies show that the more complicated solutions can also be stable by choosing their parameters well, that it is needed to look at the theoretical results when deciding the stability-based model selection.

So, we can claim that algorithm A is stable if it almost surely outputs the same clustering result on a sample whose size approaches to infinity every time we run it. That is $\lim_{m\to\infty} Pr(A(W_m) = C_k)$; m is the sample size, W is the relative frequency, C_k is the k output clustering result. Then, we can measure the instability from $instability(A) := 1 - \lim_{m\to\infty} Pr(A(W_m) = C_k)$, which yields zero if algorithm A is stable. Instability of an algorithm is also obtained by computing the expected distance between two clustering's results on two different datasets of the same size (Von Luxburg et al., 2010) that is $instability(A) := E(distance(C_k(x_n), C_k(\grave{x}_n)))$.

When it comes to our proposed algorithm, stability of DGStream lies in its robustness against independent resampling, random fluctuations in the data, and the replacements of the subsamples. We achieve this by choosing the best combinations with right values for the parameters and so we can get good clustering results with the best stability

and avoid wrong ones such as wrong split for at least one true cluster or wrong merge for at least two clusters. In more detail, in DGStream to evaluate the clustering stability, we need to run it several times on slightly different datasets. To achieve this, we need to generate a number of troubled versions of the dataset. These dataset versions are generated by subsampling or adding noise and outliers. In subsampling, we need to work with samples of different sizes. We drew such random noise-inlaid subsamples. In order not to lose the structure we want to discover by clustering with our algorithm, we must not change the samples too often. On the other hand, we might observe no significant stability results if the change in the dataset is not sufficient. So, it is a trade-off which we must cautiously deal with in all cases. Then, as usual, doing the dimensionality reduction to work with a low-dimension dataset is important. In this regard, DGStream doesn't commit any oversensitive reactions to noise and outliers, which is considered as the most prominent factor in stumbling these bad results of splitting or merging clusters.

Let's compare our algorithm with the stable approach proposed by Carlsson and Memoli (Carlsson and MÃŠmoli, 2010) regarding clustering stability. Carlsson and Memoli's approach constructs a hierarchical relationship among data to do the clustering process. DGStream is a clustering algorithm based on density and grids which detects and handles the dense clusters in the dataset in a different way from Carlsson and Memoli's approach.

Carlsson and Memoli's approach obtains an existence and uniqueness theorem instead of a non-existence result obtained by Kleinberg (2002) previously which tells that it is impossible for any standard clustering algorithm to simultaneously satisfy scale invariance, richness, and consistency. In Carlsson and Memoli's approach, the stability and convergence are established for a single linkage hierarchical clustering (SLHC) and that relaxes Kleinberg's impossibility result. Carlsson and Memoli's approach allows getting a hierarchical output from clustering methods, and then one can obtain uniqueness and existence. Carlsson and Memoli convergence results also refine the Hartigan's previous observation (Hartigan, 1985) regarding the underlying density. It does single linkage (SL) clustering of an independent, identically distributed (i.i.d.) samples from that density. The convergence results adopt general settings and it neither assumes such a smooth manifold underlying space nor assumes that the underlying probability measure must be with a density related to any reference measure. It does not matter how the points are distributed inside the space grids in the dataset. So, the SLHC is insensitive to variations in the density (Hartigan, 1981).

DGStream does care about how the data is distributed inside the space grids, the

order of arrival of the records, time they arrived and enter the clustering process is important in DGStream because DGStream employs a decay factor which ages the points over time. The point may exist and may belong to some cluster in some time, while it does not exist later or may belong to another cluster. It depends mainly on the timestamp of the point and its assigned weight and what happens to its weight by the decay factor as the time goes. So, DGStream is not an order invariant method; clustering a set of pints randomly in a different order can produce a different cluster set. In topology as well; the location of the point, to which grid it belongs is also of interest in DGStream. Therefore, the order of the records in the space grids also matters and that is what DGStream depends on while capturing the shape of the dataset, and deciding to merge these points together to form a cluster, and separating those points from those to form two or more clusters depending on the distribution of data points in the dataset. Moreover, at any time, DGStream can output a real-time result of the obtained cluster set up to that instant.

DGStream deals with weighted data points and hence weighted grids and these weights controlled by decay factor which ages the points and so the grids over time. This approach makes DGStream strong against random fluctuations in the data. DGStream takes notice of which data is outdated and deletes it. It is also aware of which grid at which time must be upgraded to become dense or downgraded to become a candidate to be deleted later. It is aware of clusters when they must be merged with another clusters or when one becomes necessary to be divided into two clusters. The number of clusters in DGStream is a parameter in a constant change with time in line with the shape of the data which is naturally in a constant change. Grids change due to their ages, which expose deleting some, while emerging others to address the evolving data over time better. DGStream is based on intuitive considerations to achieve good stability, and that is why it can be used in a wide range of practical real-life applications.

CHAPTER 4

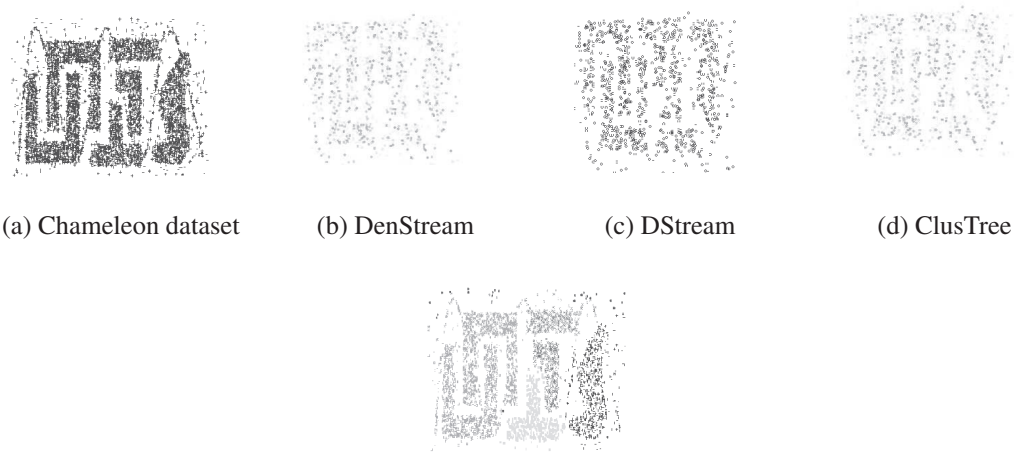
EXPERIMENTAL RESULTS

DGStream algorithm is an algorithm which combines quality and efficiency. We evaluated the quality and the efficiency of our proposed algorithm DGStream and compared it with DenStream (Cao et al., 2006), DStream (Chen and Tu, 2007), and ClusTree (Kranen et al., 2011). We mainly conducted our experiments and demonstrated their results on five datasets. One is a synthetic dataset, which is Chameleon dataset. And the others, KDDCup'99 (Hettich and Bay, 1999), Covertype (Blackard et al., 1998), Adult (Kohavi and Becker, 1996), and NSE Stocks (NSE, 2017) are real-world datasets. For both synthetic and real-world datasets, we focus on the numeric variables. So, for all datasets, we first standardize the features by minimum-maximum normalization. This means, the minimum value in one feature is mapped to 0 and the maximum value in it is mapped to 1. Note that this considerably improves the clustering result. The algorithms were implemented in Java programming language and the experiments were conducted on an Intel Core(TM) i7-4510U CPU @ 2.60GHz, 6.00GB RAM machine.

4.1. Chameleon Synthetic Dataset Results

The Chameleon dataset is an important and famous synthetic dataset in data mining and machine learning field, and contains 8000 elements. In this study, we used the first and second numerical attributes of the dataset. The chameleon dataset is a complicated dataset with nested arbitrary shaped clusters, multi-dense clusters with a lot of noise (Mekky, 2016; Alhanjouri and Ahmed, 2012). Our experiments show good results in both the clustering quality and efficiency. For the same dataset, Chameleon, our proposed algorithm DGStream, gives a better result in quality by solving the overlapping problem between clusters and reduces the noise. Also, it catches the outlier points much better, and so, a more accurate shape of clusters appears. Applying DGStream on the synthetic dataset gives very good results that demonstrate how much our proposed algorithm solved the problems that CluStream and all other k-means based algorithms are suffering from. In our experiment with Chameleon dataset evaluation, we set the horizon length value h to 1000. Every time we read h samples from the stream, we update the current result of

the cluster set with these new h samples and continue repeating this process. This process improves the quality of clustering over time (Cao et al., 2006; Hahsler and Bolaños, 2016). Figure (4.1.a) shows the original Chameleon dataset containing all the 8000 data records. Figure (4.1.b, c, d, e) show the results for clustering the dataset with DenStream, DStream, ClusTree, and our proposed algorithm DGStream respectively. These results show that our algorithm handles the outliers better with high accuracy and within lower time compared to the others. rDenStream (xiong Liu et al., 2009) is an enhanced version from DenStream algorithm which handles the outliers as well but with high time complexity. So, our algorithm gave better quality results in determining the real clusters in the given dataset with a more appropriate output.



(a) Chameleon dataset (b) DenStream (c) DStream (d) ClusTree

(e) DGStream

Figure 4.1. Clustering 8000 points from Chameleon Synthetic dataset results.

Table (4.1) shows the Chameleon synthetic dataset clustering performance metrics results of our proposed algorithm DGStream, along with other streaming algorithms. Performance metrics are time, purity, precision, recall, and F1-score. It is clear that DGStream algorithm and all other compared algorithms can perfectly determine the true classes. The purity values of all algorithms are approximately or almost exactly 1. That does not contradict the empirical study in (Carnein et al., 2017) for comparing the most important stream clustering algorithms which operate on t8.8k dataset, a similar synthetic dataset, to calculate the purity of the algorithms. We notice that the clustering output depends on the insertion order. Regarding recall, DGStream works well in retrieving almost all the relevant records to each cluster without lifting except a little. That is why

DGStream's recall is better and almost outperforms all other algorithms. However, neither the precision nor the recall alone can measure the success of the prediction, especially in the case of very imbalanced classes like our dataset examples. Therefore, F1-score is the best to be calculated for the algorithm evaluation according to it is the harmonic mean or weighted average of recall and precision. It is clear that the score for DGStream is a little bit better than ClusTree's F1-score measure, but in DenStream case, it is much better, that is because the DenStream algorithm cannot retrieve all the required records, which resulted in its bad recall measure. Finally in the other important measure, which is the time; DGStream is remarkably faster than all other algorithms as shown in Table (4.1).

Table 4.1. Performance matrices for clustering 8000 data records from Chameleon synthetic dataset by using DenStream, DStream, ClusTree, and DGStream stream clustering algorithms.

	DenStream	DStream	ClusTree	DGStream
Time (ms)	8569	9389	6734	3238
F1-score	0.2964	1	0.92	0.9575
Purity	1	1	1	0.918
Precision	1	1	1	0.921
Recall	0.174	1	0.85	0.997

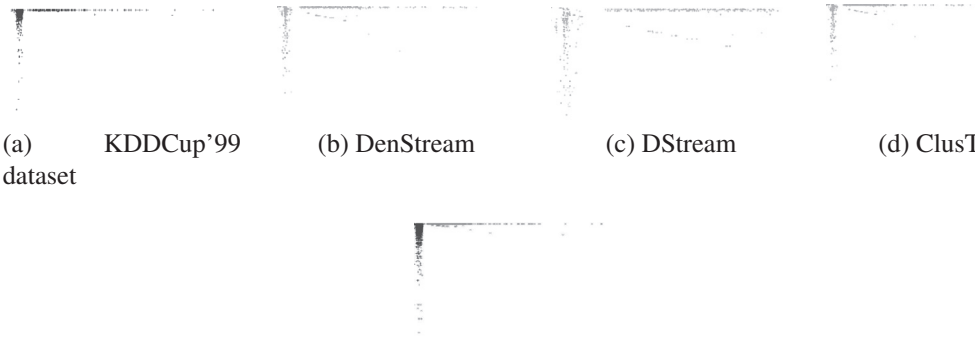
4.2. Real-World Datasets Results

We tested DGStream on three real-world datasets; KDDCup'99, Covertype, and Adult. Each dataset poses different challenges and different cluster shapes. The details are described in the following sub-subsections. Applying DGStream on the real-world datasets gives very good results, which indicates that our proposed algorithm improves both quality and efficiency compared to all existing density-based stream clustering algorithms so far.

4.2.1. KDDCup'99 Real-World Dataset Results

Among the most popular real-world datasets used for clustering data streams that we utilize is the KDDCup'99 dataset. This dataset contains 4,898,431 network traffic data records. Its attributes describe information about the connection such as the duration of the connection or the protocol type. And it's class label predicts if the connection was normal or attack, and there are 22 different attack types (Hettich and Bay, 1999).

We use the first and second numerical features of the data set. Which are "src-bytes", bytes sent in one connection, and "dst-bytes", the bytes received in one connection. Then we standardize the dataset according to the number of points we operate on. Firstly, we consider clustering the first 8000 observations from this dataset with a time horizon of 1000. Figure (4.2.a) shows the first 8000 data records from the original KDDCup'99 real-world dataset. Figure (4.2. b, c, d, e) show the results for clustering the same number of data records from the dataset with DenStream, DStream, ClusTree, and our algorithm DGStream respectively. Here, too, we observed the same outcomes as in the previous experiment with the synthetic Chameleon dataset, that our algorithm is more successful in handling the outliers and with less time complexity than all other stream algorithms.



(a) KDDCup'99 dataset (b) DenStream (c) DStream (d) ClusTree

(e) DGStream

Figure 4.2. Clustering 8000 points from KDDCup'99 real-world stream data results.

For this dataset, the clustering with DGStream gives good performance metric results as shown in Table (4.2) and Table (4.3). All stream algorithms along with DGStream give very good purity results. For the F1-score, the same, all algorithms perform very well or near perfect results and that is due to the good measures for both precision and recall for all algorithms. About the time performance, our proposed algorithm, DGStream, is

the best with much better than all other compared stream algorithms. The time for clustering 8000 points from KDDCup'99 dataset is 1737 ms. While it is 16513 ms, 15295 ms, and 4458 ms for DenStream, DStream, and ClusTree respectively.

Table 4.2. Performance matrices for clustering 8000 data records from KDDCup'99 real-world stream data by using DenStream, DStream, ClusTree, and DGStream stream clustering algorithms.

	DenStream	DStream	ClusTree	DGStream
Time (ms)	16513	15295	4458	1737
F1-score	0.969	0.9995	0.979	0.99066
Purity	1	1	1	0.9815
Precision	1	1	1	0.98147
Recall	0.966	0.999	0.96	1

Since this data contains more than just 8000 points, we clustered more than this number of points to test and compare the scalability of the stream clustering algorithms. Again, we repeated the above process with the first 20000 observations, and the results are in the Figure (4.3) and Table (4.3). All algorithms, in clustering 20000 points from KDDCup'99, produce high purity clusters. As shown in Figure (4.3) it is clear that DGStream is the best in outputting high accurate clustering results.

(a) DenStream (b) DStream (c) ClusTree (d) DGStream

Figure 4.3. Clustering 20000 points from KDDCup'99 real-world stream data results.

DGStream's precision and recall values are nearly perfect and so its F1-score. The same applies to the results of DStream algorithm. The average running times, in the case of 20000 data records with the time horizon of value 1000 are 9091, 10917, 6081, and

4570 ms. for DenStream, DStream, ClusTree, and DGStream respectively are shown in Table (4.3)., DGStream is the fastest algorithm when compared with the other stream clustering algorithms.

Table 4.3. Performance matrices for clustering 20000 data records from KDDCup'99 real-world stream data by using DenStream, DStream, ClusTree, and DGStream stream clustering algorithms with 5000 time horizon.

	DenStream	DStream	ClusTree	DGStream
Time (ms)	67422	69552	9012	4570
F1-score	0.0498	1	0.95	0.9993
Purity	1	1	1	0.9987
Precision	1	1	1	0.9986
Recall	0.981	1	0.905	1

DGStream performs better than all other compared algorithms according to Figure (4.2) and Figure (4.3). When Table (4.2) and Table (4.3) are studied, however, it seems that there is a contradiction with the results depicted in the figures. The performance of the other algorithms like DStream displays perfect recall and perfect precision so perfect F1-score must be better. The explanation is that: for example DStream recall is perfect because the fraction of the relevant data records that are successfully retrieved is 100% but actually the relevant data records are the remaining ones as this algorithm deletes the past records hence the performance is based on the kept ones only. In DGStream, on the other hand, the remaining ones which are received up to that point in time for clustering process are more than the remaining points in DStream case. And even if DGStream do not retrieve the relevant records by 100%, it still seen better as the figures show. Therefore the shape of the final cluster set obtained is different in DStream from the result of the final cluster set in DGStream case.

4.2.2. Covertype Real-World Dataset Results

Real-world dataset, Covertype, appears to be a challenging one for most of the stream clustering algorithms. It contains about 581,012 data records where each record describes a defined area of forest. The information its attributes use to describe the area

are such as the area slope, the area shade or its elevation, and a class label attribute that is a number from one to seven which shows the forest cover type. The US Forest Service (USFS) determined the forest cover types for the observations (Blackard et al., 1998). In this thesis, we used the first and the third numerical attributes, then we standardized the dataset according to the number of points we operated on. Firstly, we consider clustering the first 8000 observations from this dataset with a time horizon of 1000, in order to make a fair comparison between all stream algorithms on both synthetic and real-world datasets. Table (4.4), Table (4.5), along with Figure (4.4) and Figure (4.5) show the results for clustering 8000 and 30000 data records from the dataset with DenStream, DStream, ClusTree, and our algorithm, DGStream respectively. It is clear that the DGStream clustering result is the highest quality compared with the other algorithms. It handles the outliers accurately and with high efficiency.

(a) Covertype dataset

(b) DenStream

(c) DStream

(d) ClusTree

(e) DGStream

Figure 4.4. Clustering 8000 points from Covertype real-world stream data results.

Table (4.4) shows the clustering results based on other performance metrics for Covertype dataset first 8000 observations. Most algorithms yield high purity after slowly increasing in purity to become perfect as the clusters adjust. F1-score in both DGStream and DStream are the highest due to the high value of their recall values, as F1-score depends on both precision and recall. While DenStream's F1-score is low depending on its recall measure, and in ClusTree case, F1-score is quite better also because its recall

is better. To test and compare the running time efficiency, we run the experiments many times for each algorithm and then compute the average time consumed for each algorithm. We observed the best performance is for our proposed algorithm, DGStream, and it is much faster than all other stream algorithms. While DStream is the worse one, time performance is 9383 ms and ClusTree is better than DStream and DenStream, its time performance is 4058 ms. But ours is the best, its time performance is 1899 ms, as shown in Table (4.4).

Table 4.4. Performance matrices for clustering 8000 data records from Covertype real-world stream data by using DenStream, DStream, ClusTree, and DGStream stream clustering algorithms.

	DenStream	DStream	ClusTree	DGStream
Time (ms)	8883	9383	4058	1899
F1-score	0.3051	1	0.882	0.973
Purity	1	1	1	0.9688
Precision	1	1	1	0.9687
Recall	0.18	1	0.79	0.97713

Again, we repeated the above process with the first 30000 observations, and the results are in the Figure (4.5) and Table (4.5), DGStream is the most successful algorithm to capture the dataset shape and in handling the outliers. All algorithms produce high purity clusters. DStream's F1-score is perfect due to the perfect values of its precision and recall values. Our algorithm has the second-best F1-score because of its high precision and mostly perfect recall values. The average running times, in this case, are 14423, 9589, 7974 and 6178 ms for DenStream, DStream, ClusTree, and DGStream respectively. Therefore, DGStream is the fastest among all algorithms in clustering 30000 data records from Covertype real-world dataset.

4.2.3. Adult Real-World Dataset Results

The Adult real-world dataset, also known as "Census Income" (Kohavi and Becker, 1996), predicts whether the income exceeds $50K/yr$ or not. We use the first and third numerical features of the Adult real-world dataset. We standardize the dataset according to

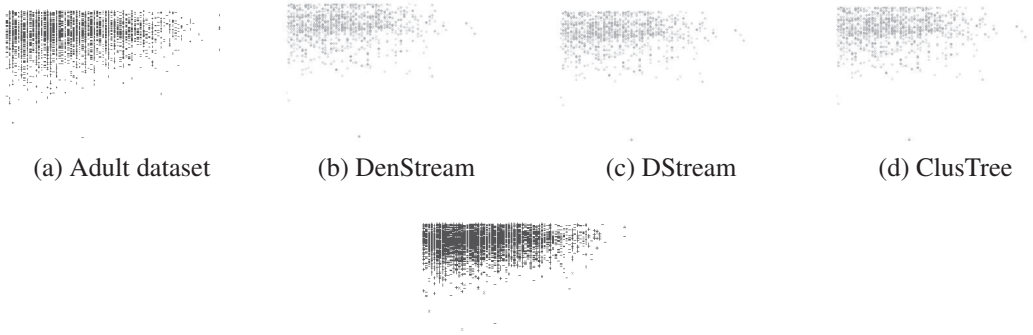
the number of points we operate on. Firstly and as we did with previously datasets all, we considered clustering the first 8000 observations from this dataset with a time horizon of 1000. Table (4.6), Table (4.7), along with Figure (4.6) and Figure (4.7) show the results for clustering the 8000 and 32500 of data records from the dataset with DenStream, DStream, ClusTree, and our algorithm, DGStream respectively. We observed in the previous experiments with the synthetic Chameleon, real-world KDDCup'99 and real-world Covertype datasets, that our algorithm is better in handling the outliers with less time complexity than all other stream algorithms. Clustering with Adult confirms the same result. Therefore, DGStream algorithm is better in both quality and efficiency among the most important algorithms for clustering data streams.

(a) DenStream (b) DStream (c) ClusTree (d) DGStream

Figure 4.5. Clustering 30000 points from Covertype real-world stream data results.

Table 4.5. Performance matrices for clustering 30000 from Covertype real-world stream data by DenStream, DStream, ClusTree, and DGStream stream clustering algorithms.

	DenStream	DStream	ClusTree	DGStream
Time (ms)	32858	40542	11605	6178
F1-score	0.6385	1	0.8833	0.9488
Purity	1	1	1	0.91
Precision	1	1	1	0.9098
Recall	0.469	1	0.791	0.9913



(a) Adult dataset (b) DenStream (c) DStream (d) ClusTree

(e) DGStream

Figure 4.6. Clustering 8000 points from Adult real-world stream data results.

For this data set, the clustering with DGStream gives good performance metric results as shown in Table (4.6) and Table (4.7). Purity is perfect with all stream algorithms for both 8000 and 32500 data records. For the F1-score, apart from the DenStream algorithm, all the other algorithms give very good results and that is due to the good outcomes for both precision and recall. DenStream's recall measure is bad and that is why its F1-score is poor. We can notice that our proposed algorithm's time performance is the best among all other compared stream algorithms. The average running time for clustering 8000 data points from Adult dataset is 1661 ms in DGStream algorithm. While it is 11451 ms, 2922 ms, and 5758 ms for DenStream, DStream, and ClusTree respectively as shown in Table (4.6). The average running time for clustering 32500 data points from Adult dataset is 4885 ms for DGStream algorithm. While it is 11959 ms, 6237 ms, and 5206 ms for DenStream, DStream, and ClusTree respectively as shown in Table (4.7).

4.2.4. Stock Marketing Real-World Dataset Results

In this experiment, we chose clustering the NSE Stocks real-world dataset. It is the National Stock Exchange of India's stock listings for each trading day of 2016 and 2017. The data is compiled to facilitate machine learning tasks on stocks, without disturbing the Stock APIs. The data has been obtained from the NSE official site (NSE, 2017), *The National Stock Exchange of India Ltd.* Retrieved from https://www.nseindia.com/.

Table 4.6. Performance matrices for clustering 8000 data records from Adult real-world stream data by using DenStream, DStream, ClusTree, and DGStream stream clustering algorithms.

	DenStream	DStream	ClusTree	DGStream
Time (ms)	11451	2922	5758	1661
F1-score	0.575	1	0.8538	0.99553
Purity	1	1	1	0.99115
Precision	1	1	1	0.9911
Recall	0.404	1	0.745	1

(a) DenStream (b) DStream (c) ClusTree (d) DGStream

Figure 4.7. Clustering 32500 points from Adult real-world stream data results.

In clustering open-ended data streams such as stock market data, it is important to capture temporal dependencies. While Bayesian networks (Buntine, 1991) and dependency networks (Heckerman et al., 2000) model the dependencies of variables, Dynamic Bayesian Networks model discrete time temporal dependencies (Dean and Kanazawa, 1988; Friedman et al., 1998). However, in our stream clustering, we want to model the continuous data record timestamps, that is the arrival times of data records. Therefore, sampling is a solution we can apply on continues variable in order to use such a technique. Nevertheless, the sampling rate would have to be determined. Slow sampling ends up with poor data representation, and fast sampling leads to a need for multiple steps of past dependence with costly clustering of the stream (Gunawardana et al., 2011). Continuous-Time Noisy-Or (Simma et al., 2012), Continuous Time Bayesian Networks (Nodelman et al., 2002, 2012), Poisson Networks (Rajaram et al., 2005; Truccolo et al., 2005), and Poisson Cascades (Simma and Jordan, 2012), are such recent solutions proposed for this problem. In clustering this real-world dataset, NSE Stocks, we used the numerical features; “OPEN” which is the opening market price of the equity symbol on the date, and the “TOTTRDQTY” which is the total traded quantity of the equity symbol.

Table 4.7. Performance matrices for clustering 32500 data records from Adult real-world stream data by using DenStream, DStream, ClusTree, and DGStream stream clustering algorithms.

	DenStream	DStream	ClusTree	DGStream
Time (ms)	49880	29723	11958	11615
F1 -score	0.685	1	0.848	0.9995
Purity	1	1	1	0.999
Precision	1	1	1	0.999
Recall	0.521	1	0.736	1

We standardize the dataset according to the number of points we operate on.

In this context, we try to discover the temporal dependencies and relations between intervals in NSE open-ended data stream states. DGStream learns how the recently arrived records affect the currently arriving ones and the near future arriving as well. Stock markets' consecutive data is all related and depends on each other. We have conducted experiments with a sampling that can benefit the stability setting. Therefore, we generated such perturbed versions of many samples from this dataset without replacement, and then added noise to these samples. Then by applying DGStream several times on them, we find that the most meaningful one is the seasonal sampling. The depicted clustering results for clustering the different same size samples from this real-world stream data by DGStream, and comparison with DenStream, DStream, and ClusTree clustering algorithms are shown in Figure (4.8). The average values of some performance matrices for all these experiments and comparisons are all in Table (4.8). In every sample, we cluster 10000 observations from the dataset with a time horizon of 1000. In this sampling, the results show how the stock records in the past affects related future stock records based on their types, daily prices and arrival time, and so they emerged in related micro-clusters and also the same macro-clusters . It is clear from the results in Table (4.8) that evaluating purity, precision, recall, and F1-score in all experiments, DGStream is still the top first or second most of the time. We again repeated the same sampling steps with replacement and applied our algorithm and the other compared algorithms several times on 40000 data points from NSE dataset. In this experiment, we measured and compared the time efficiency of our algorithm with the other algorithms. Figure (4.9) shows that DGStream is the best followed by ClusTree algorithm. It gives the best results in clustering quality and handling outliers compared to other algorithms.

In both cases sampling with replacement and without replacement, we notice that the resulted cluster set is the same cluster set every time we run DGStream on the sample. So applying the stability equation $\lim_{m\to\infty} Pr(A(W_m) = C_k)$ is $\lim_{m\to\infty} Pr(DGStream(W_{10000,40000}) = SameClusterSet)$. So, DGStream stability is 1.

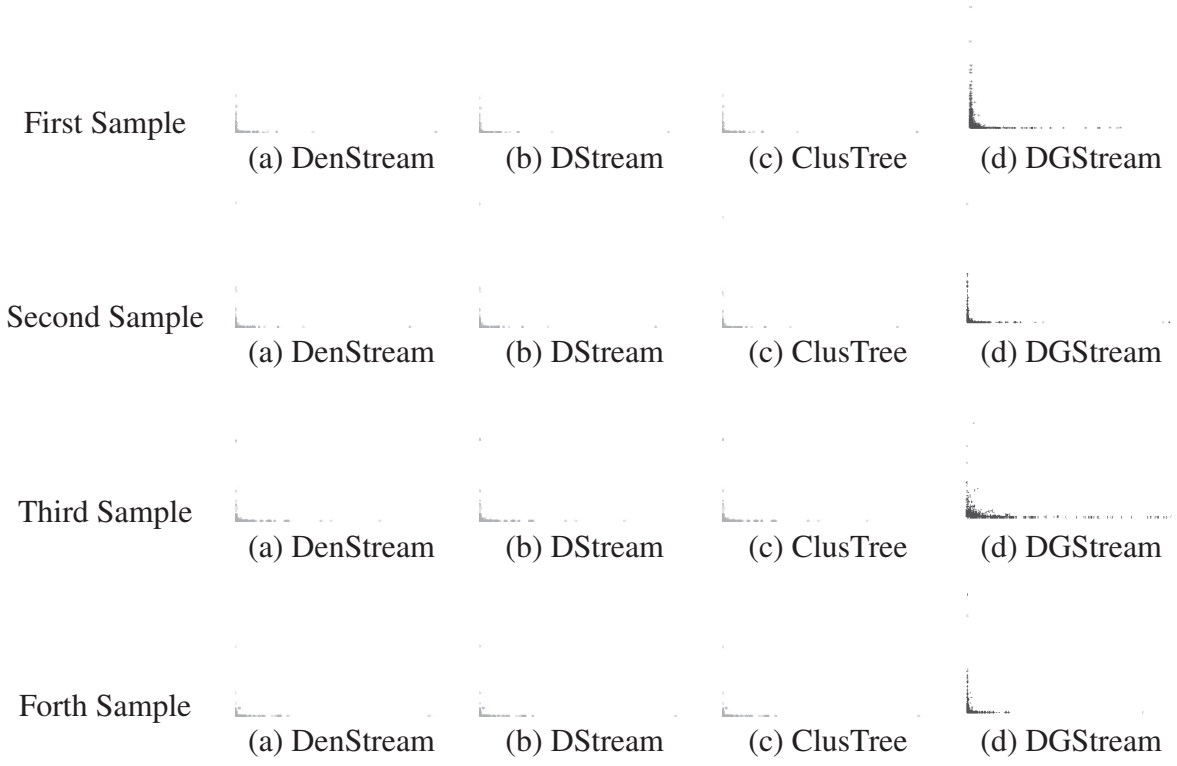


Figure 4.8. Clustering same size of different samples from NSE real-world stream data without replacement results.

In all previous experiments, we observed with both the synthetic and real-world datasets, that our algorithm outperforms all other algorithms in handling the outliers, learning the underlying dependency structure of data records, time performance, and equality . Applying the clustering algorithms to NSE dataset verifies the same claim that our proposed algorithm DGStream is the best in both quality and efficiency among the most important stream clustering algorithms.

Table 4.8. Performance matrices for clustering same size of different samples from NSE real-world stream data without replacement by DenStream, DStream, ClusTree, and DGStream stream clustering algorithms

		DenStream	DStream	ClusTree	DGStream
Time (ms)	Sample 1	34210	33896	7053	3430
	Sample 2	35283	36599	8938	3375
	Sample 3	57484	9500	5723	3287
	Sample 4	35574	39461	6825	3464
F1-score	Sample 1	0.97	1	0.98	0.9993
	Sample 2	0.97	1	0.98	1
	Sample 3	0.98	1	0.98	1
	Sample 4	0.98	1	0.98	1
Purity	Sample 1	1	1	1	0.9989
	Sample 2	1	1	1	1
	Sample 3	1	1	1	1
	Sample 4	1	1	1	0.9996
Precision	Sample 1	1	1	1	0.9994
	Sample 2	1	1	1	1
	Sample 3	1	1	1	0.9996
	Sample 4	1	1	1	1
Recall	Sample 1	0.95	1	0.95	0.9992
	Sample 2	0.95	1	0.96	0.9997
	Sample 3	0.98	1	0.96	1
	Sample 4	0.96	1	0.96	0.9989

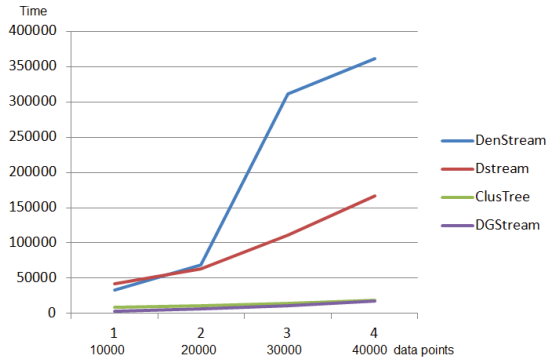


Figure 4.9. Time efficiency for clustering different sizes samples from NSE real-world stream data with replacement results.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1. Conclusion

In this study, we proposed a new stream clustering algorithm, DGStream. It is a density and grid-based algorithm with insightful implications for clustering stream data. DGStream algorithm has been tested and compared over many datasets, both synthetic and real-world datasets and under different scales and compared with DenStream, DStream, and ClusTree stream algorithms in the same field. So, experimentally under the same conditions and datasets, we have demonstrated that DGStream outperforms several well-known density-based stream clustering algorithms. It can find datasets with clusters of arbitrary shapes, multi-density and without the prior knowledge of parameters like the number of clusters. It is shown by many experiments that our proposed algorithm is significantly fastest among all the compared algorithms; it achieves the best time efficiency along with the best quality. Its recall measurement is always high due to its ability in assigning almost all relevant records to the corresponding correct clusters with, to a large extent, perfect purity, which means that our algorithm can create clusters much close to the true structure of the stream data. DGStream has also many good features; it is a strong and robust algorithm to noise and presence of outliers; needs only one-pass for processing stream data; it considers the evolving data by employing a decaying function that decreases the weights of the outdated data over time. Therefore, it is suitable for real-world applications where the most interest is in the recent information while the old information decreases over time like stock marketing. From all conducted experiments, we can say that our proposed algorithm outperformed all other density-based stream clustering algorithms in both efficiency and accuracy. However, we have to realize that stream clustering algorithms cluster streaming data from different points of view, and choosing between them depends on what we want to achieve from applying them, such as more accuracy is better than more reliability in some instances, and sometimes for particular application low time complexity is the most important property. Therefore, we can say for sure that for some applications or for a particular dataset and under specific conditions there is an algorithm that is much better than one another.

5.1.1. Future Work

As future work, it will be good to find a way to detect dense grids so that we can tune the parameters according to how much the density is and how many grids in the space have this density.

Another improvement may be changing the parameters dynamically according to the dataset we process on.

In addition, it is good to enter the prediction factor and trying to predict which grids might be useless in the future depending on how the incoming stream points map to grids so we can focus more on the grids at the borders of the clusters.

In our algorithm, we can improve the quality to become 100% by retrieving all the relevant data records in the stream and not delete or eliminate any point.

REFERENCES

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

Aggarwal, C. C. (2009). On classification and segmentation of massive audio data streams. *Knowledge and information systems 20*(2), 137–156.

Aggarwal, C. C., J. Han, J. Wang, and P. S. Yu (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pp. 81–92. VLDB Endowment.

Aggarwal, C. C., J. Han, J. Wang, and P. S. Yu (2004). A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pp. 852–863. VLDB Endowment.

Agrawal, R., J. Gehrke, D. Gunopulos, and P. Raghavan (1998). *Automatic subspace clustering of high dimensional data for data mining applications*, Volume 27. ACM.

Ahmed, R. D., G. Dalkılıç, and M. Erten (2018). Survey: Running and comparing stream clustering algorithms. CEUR Workshop Proceedings.

Alazeez, A. A. A., S. Jassim, and H. Du (2017). Einckm: An enhanced prototype-based method for clustering evolving data streams in big data. In *ICPRAM*, pp. 173–183.

Alhanjouri, M. A. and R. D. Ahmed (2012). New density-based clustering technique: Gmdbscan-ur. *New Density-Based Clustering Technique: GMDBSCAN-UR. 3*(1).

Amini, A., H. Saboohi, T. Ying Wah, and T. Herawan (2014). A fast density-based clustering algorithm for real-time internet of things stream. *The Scientific World Journal 2014*.

Ankerst, M., M. M. Breunig, H.-P. Kriegel, and J. Sander (1999). Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, Volume 28, pp. 49–60. ACM.

Arai, B., G. Das, D. Gunopulos, and N. Koudas (2007). Anytime measures for top-k algorithms. In *Proceedings of the 33rd international conference on Very large data bases*, pp. 914–925. VLDB Endowment.

Arthur, D. and S. Vassilvitskii (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035. Society for Industrial and Applied Mathematics.

Assent, I., R. Krieger, B. Glavic, and T. Seidl (2008). Clustering multidimensional sequences in spatial and temporal databases. *Knowledge and Information Systems 16*(1), 29–51.

Babcock, B., M. Datar, R. Motwani, and L. O'Callaghan (2003). Maintaining variance and k-medians over data stream windows. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 234–243. ACM.

Bandyopadhyay, S., C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta (2006). Clustering distributed data streams in peer-to-peer environments. *Information Sciences 176*(14), 1952–1985.

Barbará, D. (2002). Requirements for clustering data streams. *ACM sIGKDD Explorations Newsletter 3*(2), 23–27.

Barbará, D. and P. Chen (2000). Using the fractal dimension to cluster datasets. In *KDD*, pp. 260–264.

Beckmann, N., H.-P. Kriegel, R. Schneider, and B. Seeger (1990). The r*-tree: an efficient and robust access method for points and rectangles. In *Acm Sigmod Record*, Volume 19, pp. 322–331. Acm.

Beringer, J. and E. Hüllermeier (2006). Online clustering of parallel data streams. *Data & Knowledge Engineering 58*(2), 180–204.

Bhatia, S. K. et al. (2004). Adaptive k-means clustering. In *FLAIRS conference*, pp. 695–699.

Bifet, A., G. Holmes, R. Kirkby, and B. Pfahringer (2010). Moa: Massive online analysis. *Journal of Machine Learning Research 11*(May), 1601–1604.

Blackard, J. A., D. J. Dean, and C. Anderson (1998). The forest covertype dataset.

Buntine, W. (1991). Theory refinement on bayesian networks. In *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*, pp. 52–60. Morgan Kaufmann Publishers Inc.

Cao, F., M. Estert, W. Qian, and A. Zhou (2006). Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*, pp. 328–339. SIAM.

Carlsson, G. and F. MÃŠmoli (2010). Characterization, stability and convergence of hierarchical clustering methods. *Journal of machine learning research 11*(Apr), 1425–1470.

Carnein, M., D. Assenmacher, and H. Trautmann (2017). An empirical comparison of stream clustering algorithms. In *Proceedings of the Computing Frontiers Conference*, pp. 361–366. ACM.

Chen, Y. and L. Tu (2007). Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142. ACM.

Cheng, J., Y. Ke, and W. Ng (2008). A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems 16*(1), 1–27.

Dai, B.-R., J.-W. Huang, M.-Y. Yeh, and M.-S. Chen (2006). Adaptive clustering for multiple evolving streams. *IEEE Transactions on Knowledge and Data Engineering 18*(9), 1166–1180.

Dang, X. H., W.-K. Ng, and K.-L. Ong (2008). Online mining of frequent sets in data streams with error guarantee. *Knowledge and information systems 16*(2), 245–258.

De Silva, V. and G. E. Carlsson (2004). Topological estimation using witness complexes. *SPBG 4*, 157–166.

Dean, T. L. and K. Kanazawa (1988). Probabilistic temporal reasoning. In *AAAI*, pp. 524–529.

DeCoste, D. (2002). Anytime interval-valued outputs for kernel machines: Fast support vector machine classification via distance geometry.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological) 39*(1), 1–22.

Ester, M., H.-P. Kriegel, J. Sander, X. Xu, et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, Volume 96, pp. 226–231.

Farid, D. M., L. Zhang, A. Hossain, C. M. Rahman, R. Strachan, G. Sexton, and K. Dahal (2013). An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications 40*(15), 5895–5906.

Friedman, N., K. Murphy, and S. Russell (1998). Learning the structure of dynamic probabilistic networks. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pp. 139–147. Morgan Kaufmann Publishers Inc.

Gaber, M. M., A. Zaslavsky, and S. Krishnaswamy (2007). A survey of classification methods in data streams. In *Data streams*, pp. 39–59. Springer.

Gama, J. (2010). *Knowledge discovery from data streams*. Chapman and Hall/CRC.

Gama, J., I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR) 46*(4), 44.

Gan, J. and Y. Tao (2017). Dynamic density based clustering. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1493–1507. ACM.

Gong, S., Y. Zhang, and G. Yu (2017). Clustering stream data by exploring the evolution of density mountain. *Proceedings of the VLDB Endowment 11*(4), 393–405.

Guha, S., A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan (2003). Clustering

data streams: Theory and practice. *IEEE transactions on knowledge and data engineering 15*(3), 515–528.

Guha, S. and N. Mishra (2016). Clustering data streams. In *Data stream management*, pp. 169–187. Springer.

Guha, S., N. Mishra, R. Motwani, and L. o'Callaghan (2000). Clustering data streams. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 359–366. IEEE.

Guha, S., R. Rastogi, and K. Shim (2001). Cure: an efficient clustering algorithm for large databases. *Information systems 26*(1), 35–58.

Gunawardana, A., C. Meek, and P. Xu (2011). A model for temporal dependencies in event streams. In *Advances in Neural Information Processing Systems*, pp. 1962–1970.

Guttman, A. (1984). *R-trees: A dynamic index structure for spatial searching*, Volume 14. ACM.

Hahsler, M. and M. Bolaños (2016). Clustering data streams based on shared density between micro-clusters. *IEEE Transactions on Knowledge and Data Engineering 28*(6), 1449–1461.

Hartigan, J. A. (1981). Consistency of single linkage for high-density clusters. *Journal of the American Statistical Association 76*(374), 388–394.

Hartigan, J. A. (1985). Statistical theory in clustering. *Journal of classification 2*(1), 63–76.

Heckerman, D., D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie (2000). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research 1*(Oct), 49–75.

Hettich, S. and S. Bay (1999). The uci kdd archive [http://kdd. ics. uci. edu]. irvine, ca: University of california. *Department of Information and Computer Science 152*.

Hinneburg, A., D. A. Keim, et al. (1998). An efficient approach to clustering in large multimedia databases with noise. In *KDD*, Volume 98, pp. 58–65.

Isaksson, C., M. H. Dunham, and M. Hahsler (2012). Sostream: Self organizing density-based clustering over data stream. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pp. 264–278. Springer.

Jain, A., Z. Zhang, and E. Y. Chang (2006). Adaptive non-linear clustering in data streams. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pp. 122–131. ACM.

Kanageswari, V. and A. Pethalakshmi (2017). A novel approach of clustering using cobweb. *International Journal of Information Technology (IJIT) 3*(3).

Kohavi, R. and B. Becker (1996). Adult dataset [online] available: http://archive. ics. uci. edu/ml/d atasets.

Kranen, P., I. Assent, C. Baldauf, and T. Seidl (2011). The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and information systems 29*(2), 249–272.

Kranen, P., H. Kremer, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer (2010). Clustering performance on evolving data streams: Assessing algorithms and evaluation measures within moa. In *2010 IEEE International Conference on Data Mining Workshops*, pp. 1400–1403. IEEE.

Kranen, P., R. Krieger, S. Denker, and T. Seidl (2010). Bulk loading hierarchical mixture models for efficient stream classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 325–334. Springer.

Kranen, P. and T. Seidl (2009). Harnessing the strengths of anytime algorithms for constant data streams. *Data Mining and Knowledge Discovery 19*(2), 245–260.

Li, H.-F., M.-K. Shan, and S.-Y. Lee (2008). Dsm-fi: an efficient algorithm for mining frequent itemsets in data streams. *Knowledge and Information Systems 17*(1), 79–97.

Li, S. and X. Zhou (2017). An intrusion detection method based on damped window of

data stream clustering. In *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, Volume 1, pp. 211–214. IEEE.

Liu, H., X. Hou, and Z. Yang (2016). Design of intrusion detection system based on improved k-means algorithm. *Computer Technology and Development 1*, 101–105.

Lühr, S. and M. Lazarescu (2009). Incremental clustering of dynamic data streams using connectivity based representative points. *Data & Knowledge Engineering 68*(1), 1–27.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Volume 1, pp. 281–297. Oakland, CA, USA.

Manning, C., P. Raghavan, and H. Schütze (2010). Introduction to information retrieval. *Natural Language Engineering 16*(1), 100–103.

Mekky, A. R. (2016). Fuzzy neighborhood grid-based dbscan using representative points. *Feature Engineering in Hybrid Recommender Systems*, 63.

Nasraoui, O., C. Rojas, and C. Cardona (2006). A framework for mining evolving trends in web data streams using dynamic learning and retrospective validation. *Computer Networks 50*(10), 1488–1512.

Ng, W. and M. Dash (2010). Discovery of frequent patterns in transactional data streams. In *Transactions on large-scale data-and knowledge-centered systems II*, pp. 1–30. Springer.

Nodelman, U., C. R. Shelton, and D. Koller (2002). Continuous time bayesian networks. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pp. 378–387. Morgan Kaufmann Publishers Inc.

Nodelman, U., C. R. Shelton, and D. Koller (2012). Expectation maximization and complex duration distributions for continuous time bayesian networks. *arXiv preprint arXiv:1207.1402*.

Nutakki, G. C. and O. Nasraoui (2017). Clustering data streams with adaptive forgetting.

In *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 494–497. IEEE.

O'callaghan, L., N. Mishra, A. Meyerson, S. Guha, and R. Motwani (2002). Streaming-data algorithms for high-quality clustering. In *Proceedings 18th International Conference on Data Engineering*, pp. 685–694. IEEE.

Oh, S.-H., J.-S. Kang, Y.-C. Byun, G.-L. Park, and S.-Y. Byun (2005). Intrusion detection based on clustering a data stream. In *Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA'05)*, pp. 220–227. IEEE.

Pravilovic, S., A. Appice, and D. Malerba (2014). Integrating cluster analysis to the arima model for forecasting geosensor data. In *International Symposium on Methodologies for Intelligent Systems*, pp. 234–243. Springer.

Rajaram, S., T. Graepel, and R. Herbrich (2005). Poisson-networks: A model for structured point processes. In *Proceedings of the 10th international workshop on artificial intelligence and statistics*, pp. 277–284. Citeseer.

Ren, J. and R. Ma (2009). Density-based data streams clustering over sliding windows. In *2009 Sixth international conference on fuzzy systems and knowledge discovery*, Volume 5, pp. 248–252. IEEE.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics 20*, 53–65.

Ruiz, C., E. Menasalvas, and M. Spiliopoulou (2009). C-denstream: Using domain knowledge on a data stream. In *International Conference on Discovery Science*, pp. 287–301. Springer.

Ruiz, C., M. Spiliopoulou, and E. Menasalvas (2007). C-dbscan: Density-based clustering with constraints. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pp. 216–223. Springer.

Ruiz, C., M. Spiliopoulou, and E. Menasalvas (2010). Density-based semi-supervised clustering. *Data mining and knowledge discovery 21*(3), 345–370.

Seidl, T., I. Assent, P. Kranen, R. Krieger, and J. Herrmann (2009). Indexing density models for incremental learning and anytime classification on data streams. In *Proceedings of the 12th international conference on extending database technology: advances in database technology*, pp. 311–322. ACM.

Silva, J. A., E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, and J. Gama (2013). Data stream clustering: A survey. *ACM Computing Surveys (CSUR) 46*(1), 13.

Simma, A., M. Goldszmidt, J. MacCormick, P. Barham, R. Black, R. Isaacs, and R. Mortier (2012). Ct-nor: representing and reasoning about events in continuous time. *arXiv preprint arXiv:1206.3280*.

Simma, A. and M. I. Jordan (2012). Modeling events with cascades of poisson processes. *arXiv preprint arXiv:1203.3516*.

Spiliopoulou, M., I. Ntoutsi, Y. Theodoridis, and R. Schult (2006). Monic: modeling and monitoring cluster transitions. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 706–711. ACM.

Spinosa, E. J., A. P. de Leon F de Carvalho, and J. Gama (2007). Olindda: A cluster-based approach for detecting novelty and concept drift in data streams. In *Proceedings of the 2007 ACM symposium on Applied computing*, pp. 448–452. ACM.

Street, W. N. and Y. Kim (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 377–382. ACM.

Sun, H., G. Yu, Y. Bao, F. Zhao, and D. Wang (2005). Cds-tree: An effective index for clustering arbitrary shapes in data streams. In *15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05)*, pp. 81–88. IEEE.

Truccolo, W., U. T. Eden, M. R. Fellows, J. P. Donoghue, and E. N. Brown (2005). A point process framework for relating neural spiking activity to spiking history, neural

ensemble, and extrinsic covariate effects. *Journal of neurophysiology 93*(2), 1074–1089.

Udommanetanakit, K., T. Rakthanmanon, and K. Waiyamai (2007). E-stream: Evolution-based technique for stream clustering. In *International conference on advanced data mining and applications*, pp. 605–615. Springer.

Ueno, K., X. Xi, E. Keogh, and D.-J. Lee (2006). Anytime classification using the nearest neighbor algorithm with applications to stream mining. In *Sixth International Conference on Data Mining (ICDM'06)*, pp. 623–632. IEEE.

Van Leeuwen, M. and A. Siebes (2008). Streamkrimp: Detecting change in data streams. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 672–687. Springer.

Vlachos, M., J. Lin, E. Keogh, and D. Gunopulos (2003). A wavelet-based anytime algorithm for k-means clustering of time series. In *In proc. workshop on clustering high dimensionality data and its applications*. Citeseer.

Von Luxburg, U. et al. (2010). Clustering stability: an overview. *Foundations and Trends® in Machine Learning 2*(3), 235–274.

Wang, H., W. Fan, P. S. Yu, and J. Han (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 226–235. AcM.

Wang, Z., B. Wang, C. Zhou, and X. Xu (2004). Clustering data streams on the two-tier structure. In *Asia-Pacific Web Conference*, pp. 416–425. Springer.

xiong Liu, L., Y. fei Guo, J. Kang, and H. Huang (2009). A three-step clustering algorithm over an evolving data stream. *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems 1*, 160–164.

Yang, H. and S. Fong (2013). Countering the concept-drift problem in big data using iovfdt. In *2013 IEEE International Congress on Big Data*, pp. 126–132. IEEE.

Yang, Y., G. Webb, K. Korb, and K. M. Ting (2007). Classifying under computational

resource constraints: anytime classification using probabilistic estimators. *Machine Learning 69*(1), 35–53.

Zhang, T., R. Ramakrishnan, and M. Livny (1996). Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, Volume 25, pp. 103–114. ACM.

Zhou, A., F. Cao, W. Qian, and C. Jin (2008). Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems 15*(2), 181–214.

VITA

Date and Place of Birth: 22.11.1984, Gaza-Palestine

EDUCATION

2015 - 2018 Doctor of Philosophy in Computer Engineering

Graduate School of Engineering and Sciences, İzmir Institute of Technology, İzmir -Turkey

Thesis Title: DENSITY BASED STREAM CLUSTERING ALGORITHM

Supervisor: Assoc. Prof. Dr. Tolga Ayav, Prof. Dr. Yusuf Erten, Assoc. Prof. Dr. Gökhan Dalkılıç

2009 - 2011 Master of Science in Computer Engineering

Graduate School of Engineering and Sciences, Islamic University, Gaza -Palestine

Thesis Title: NEW DENSITY-BASED CLUSTERING TECHNIQUE

Supervisor: Assist. Dr. Mohammed Alhanjouri

2002 - 2007 Bachelor of Computer Engineering

Department of Computer Engineering, Faculty of Engineering, Islamic University, Gaza - Palestine

PROFESSIONAL EXPERIENCE

2007 - 2014 Research and Teaching Assistant, Academic Teaching

Instructor for many computer engineering courses in many universities and a Computer Engineer in many ministries, Gaza- Palestine

PUBLICATIONS

- Rowanda Ahmed, Gökhan Dalkılıç, Yusuf Erten, "DGStream: High Quality and Efficiency Stream Clustering Algorithm", Expert Systems With Applications, 2019, 112947.
- Rowanda Ahmed, Gökhan Dalkılıç, Yusuf Erten, "Survey: Running and Comparing Clustering Algorithms", Ulusal Yazılım Mühendisliği Sempozyumu UYMS' 18 Conference, Sabancı Universitesi, Istanbul, Turkey.
- Rwand Ahmed, Mohammed Al Hanjouri, "New Density-Based Clustering Technique: GMDBSCAN-UR", International Journal of Advanced Research in Computer Science (IJARCS).
- Aiman Abu Samra, Emad Kehail and Rwand Ahmed, "Software Framework Analysis for IUG", 3rd International Conference on Engineering and Gaza Reconstruction at IUG in 2010. Theme5.