DENSITY-BASED AND PARAMETERLESS CLUSTERING OF EMBEDDED DATA STREAMS

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

BY

ÖZLEM POYRAZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

DENSITY-BASED AND PARAMETERLESS CLUSTERING OF EMBEDDED DATA STREAMS

submitted by **ÖZLEM POYRAZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Mehmet Volkan Atalay
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Ferda Nur Alpaslan
Computer Engineering, METU _____

Prof. Dr. Mehmet Volkan Atalay
Computer Engineering, METU _____

Prof. Dr. Fazlı Can
Computer Engineering, Bilkent University _____

Date: 09.09.2021

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

Name, Surname: Özlem Poyraz

Signature :

ABSTRACT

DENSITY-BASED AND PARAMETERLESS CLUSTERING OF EMBEDDED DATA STREAMS

Poyraz, Özlem
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Mehmet Volkan Atalay

September 2021, 75 pages

With the accelerating digitalization of the world, the amount of high-speed data produced increases rapidly, and it is difficult to record and collectively process such a data-stream. This creates the need for processing as soon as it arrives without recording the data stream. Mostly, there is no prior information about data. Additionally, characteristics of data streams may change over time; this phenomenon is called concept drift. Since clustering works without actual labels, it is suitable to be used on data streams. Clustering algorithms for data streams should read the data only once, work in real-time, and adapt to the concept drift. With Density-Based and Parameterless Clustering of Embedded Data Streams (DBPCES) algorithm developed in this study, data streams are embedded into two dimensions and clustered with a parameterless density-based clustering algorithm. To embed the data stream into 2-dimensions, UMAP algorithm was adapted to handle data streams and concept drift. For clustering, DBSCAN algorithm was used on embedded data points. DBSCAN parameters were estimated with a heuristic so that data stream can be clustered without requiring any data-dependent parameters from the user. DBPCES algorithm was run on synthetic and real data streams that differ in actual cluster count, dimension count, and

concept drift rate. The performance of DBPCES was compared with DenStream and implementation of Zubaroğlu and Atalay. As evaluation metrics, adjusted rand index, purity, and silhouette coefficient were used. Additionally, execution times were compared as well. Although DBPCES was not as fast as DenStream, it achieved similar results with other algorithms.

Keywords: data stream, clustering, UMAP on data streams, parameterless DBSCAN, online

ÖZ

BOYUTU AZALTILMIŞ AKAN VERİNİN YOĞUNLUĞA DAYALI VE PARAMETRESİZ KÜMELENMESİ

Poyraz, Özlem
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Mehmet Volkan Atalay

Eylül 2021 , 75 sayfa

Dünyanın hızlı bir şekilde dijitalleşmesiyle üretilen veri miktarı ve verinin üretilme hızı gittikçe artmakta ve bu şekilde akan verinin kaydedilerek topluca işlenmesi zorlaşmaktadır. Bu da akan veriyi kaydetmeden geldiği anda işleme ihtiyacı doğurmaktadır. Çoğu zaman veriyi islemeden once verinin karakterstigi ile ilgili elimizde fazla bilgi bulunmamaktadır. Buna ek olarak, akan verinin özellikleri giderek değişebilmektedir (kavram kayması). Kümeleme metodları verilerin gerçek kümelerine ihtiyaç duymadığından, akan veri için uygundur. Akan veri kümeleme algoritmaları veriyi sadece bir kez okumalı, gerçek zamanda çalışmalı ve verideki kavram kaymasina uyum sağlayabilmelidir. Bu çalışmada geliştirilen Boyutu Azaltılmış Akan Verinin Yoğunluğa Dayalı ve Parametresiz Kümelenmemesi (DBPCES) algoritması ile akan veri 2 boyutlu hale getirilerek, parametre gerektirmeyen yoğunluk tabanlı bir algoritma ile kümelenebilecektir. Akan veriyi iki boyutlu hale getirebilmek için UMAP algoritması akan veriye ve verideki kaymaya duyarlı hale getirilmiştir. Kümeleme için DBSCAN algoritması iki boyutlu hale getirilen veri üzerinde kullanılmıştır. DBSCAN parametreleri geliştirilen bulgusal bir yöntemle tahmin edilerek kullanıcıdan

veriye bağlı hiçbir parametre almadan akan veri kümelenmiştir. DBPCES algoritması, gerçek küme sayısı, boyut sayısı ve kavram kayma hızı bakımından farklılık gösteren yapay ve gerçek veri akışları üzerinde çalıştırılmıştır. DBPCES algoritmasının performansı DenStream ve Zubaroğlu ve Atalay'ın algoritması ile karşılaştırılmıştır. Performans değerlendirme ölçütü olarak saflık, siluet skoru ve düzeltilmiş rand endeksi kullanılmıştır. Ayrıca uygulama süreleri de karşılaştırılmıştır. DBPCES, DenStream kadar hızlı olmasa da, kümeleme için kullanıcıdan veriye bağlı herhangi bir parametre almadan diğer algoritmalarla benzer sonuçlar elde etmiştir.

Anahtar Kelimeler: akan veri, kümeleme, akan veri üzerinde UMAP, parametresiz DBSCAN

To My Family

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Prof. Dr. Mehmet Volkan Atalay for his endless patience, encouragement, guidance, and priceless contributions to this thesis.

I'd also like to extend my gratitude to Alaettin Zubaroglu for his contributions.

I also want to thank my parents; my precious mother Canan Özdoğan and dear father Ayhan Özdoğan, and my lovely sister Ceren Özdoğan. I am deeply indebted to my family for their unfailing support and continuous encouragement throughout my years of study, and everlasting love to me.

I owe my great appreciation to my team lead İhsan Yayla for the support and convenience he provided.

I would like to specially thank to my dearest friends, Feyza Kerti, Özge Baş Aksu, Kübra Demirel, Aybala Duman, Furkan Aslan, Umut Serkan Yavuz for their continuous enthusiasm to help and support me all the time throughout this process.

Finally, I must express my very profound gratitude to my beloved husband Doğan Poyraz for his love, trust, providing me with encouragement and patience throughout the duration of this process as well as throughout my life.

This accomplishment would not have been possible without them.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

DBPCES	Density-Based and Parameterless Clustering of Embedded Data Streams
UMAP	Uniform manifold approximation and projection for dimension reduction
DBSCAN	Density-based spatial clustering of applications with noise
DenStream	Density-based clustering over an evolving data stream with noise
PCA	Principal Components Analysis
MDS	Multidimensional scaling
AE	Auto-Encoder
LDA	Linear Discriminant Analysis
MMC	Maximum Margin Criterion
IPCA	Incremental PCA
CCIPCA	Candid Covariance-free Incremental PCA
ILDA	Incremental LDA
IMMC	Incremental MMC
RP	Random Projection
CS	Compressed Sensing
HT	Hashing Trick
LSH	Locality Sensitive Hashing
Isomap	Isometric Mapping
tSNE	t-distributed Stochastic Neighbor Embedding
S-Isomap	Streaming Isomap
CEDAS	Clustering of Evolving Data-streams into Arbitrary Shapes

DBIECM	Davies-Bouldin Index Evolving Clustering Method for streaming data clustering
FEAC-Stream	Fast Evolutionary Algorithm for Clustering data streams
PMC	potential micro cluster
OMC	outlier micro cluster
CMC	core micro cluster
CF	cluster feature
ARI	Adjusted Rand Index
DSets	DSets Dominant Sets
KNNDBSCAN	K-nearest neighbors DBSCAN
KDE	kernel density estimation
VDBSCAN	Varied density based spatial clustering of applications with noise
DCRBD	Density Clustering Based on Radius of Data
rDenStream	DenStream with retrospect
SDStream	Density-based data streams clustering over sliding windows
IKPCA	Fast Iterative Kernel PCA
BP-NN	Back-Propagating Neural Network
FDB	Feature Database
FIKDR	Fast Iterative Kernel Dimensionality Reduction
FIKOCFrame	Fast Iterative Kernel Online classification

CHAPTER 1

INTRODUCTION

With the digitalization of the world, almost any device can connect to the internet and generates data. As the number of connected devices increases rapidly, the amount of generated data also grows dramatically [2]. This generated data is one of the most important and valuable assets in such a world because lots of information can be extracted from this generated data.

The data that is continuously generated by these devices is called a data stream. Data streams are slightly different than offline data. Data streams are continuous, generally high dimensional, and may evolve over time. Storing such big and continuous data is not feasible since data size may become enormous in very short period of time. Even if it can be stored, processing such huge data at once would be very slow. Additionally, there is usually no prior information available for data streams. Traditional data analysis algorithms may require information about data such as the distributional form of the data, number of classes, and actual labels. Since processing whole data at once is not feasible because of storage and computational constraints, examining such information at the end of the stream is not possible. In other words, parameters cannot be deduced from whole data. Therefore, data stream analysis algorithms should require as few data-dependent parameters as possible. Additionally, most of the data analysis algorithms suffer from high dimensionality which is called "Curse of Dimensionality". With the increase of number of features that a data point has, data sparsity and distance concentration problems arise. In higher dimensions, the more training data points are required to generate data model and similarity or dissimilarity of data points becomes unclear. Data stream analysis algorithms should overcome curse of dimensionality since data streams are generally high dimensional.

Clustering is an unsupervised task that groups data points according to their similarity. Clustering algorithms are used in various fields such as medical diagnosis, anomaly and fraud detection, and product recommendation [3]. Mostly, traditional clustering algorithms (offline algorithms) need whole data, pass several times from data for processing and require user-defined parameters which are inferred by examining whole data. Therefore, traditional clustering algorithms are not directly applicable to data streams. Data stream clustering algorithms should process the data only once, work in real-time, and adapt themselves to change in data characteristics over time. Additionally, data stream clustering algorithms should not require data-dependent parameters since prior information is not available. Data stream clustering also suffers from curse of dimensionality because it is based on distance calculations for similarity. Therefore, data streams clustering algorithms are also overcome problems of high dimensionality. While several algorithms exist for data stream clustering [1], most of them require parameters that depend on prior information about whole data. In this study, we describe a data stream clustering algorithm Density-Based and Parameterless Clustering of Embedded Data Streams (DBPCES) that requires no prior information about data. With DBPCES, data streams are embedded into two dimensions using Uniform manifold approximation and projection for dimension reduction (UMAP) to overcome curse of dimensionality and clustered with Density-based spatial clustering of applications with noise (DBSCAN) algorithm. UMAP is selected as dimensionality reduction algorithm among several algorithms because the data points can be embedded using previously generated model. This makes UMAP applicable to data streams. The values of parameters of DBSCAN algorithm (ϵ and $minPoints$) are estimated using a heuristic. A modified version of binary search is performed for estimation of $minPoints$ parameter value. The interval of $minPoints$ parameter value is divided into two parts and middle values of each part are selected as candidate values for $minPoints$ parameter. For both selected candidate $minPoints$ parameter values, candidate ϵ values are determined with k-dist graph. Using these candidate (ϵ, $minPoints$) pairs, DBSCAN algorithm is executed and silhouette score of clustering is calculated for each pair of parameter values. The part of the $minPoints$ interval that gives smaller silhouette score is eliminated until there is no interval to divide. (ϵ, $minPoints$) pair that gives the highest silhouette score is determined as the actual parameters of DBSCAN algorithm. In this way, clustering is performed without any

user-defined data-dependent parameter.

DBPCES consists of 5 phases: Initialization, Parameter Value Selection, Embedding, Clustering, Label Matching. DBPCES adapts itself to change in the characteristics of data by periodically running 5 phases of the algorithm. DBPCES requires 3 hyper-parameters: $re - initializationperiod$ (rp), $initializationsize$ (is), and $window - size$ (ws). In the initialization phase, UMAP model is generated using is data points at the beginning of each period. Values of both parameters of DBSCAN are estimated in the parameter value selection phase using is data points. The remaining data is embedded window by window (ws data points in each window) using generated UMAP model in the embedding phase. After that, data points in that period are clustered in the clustering phase using DBSCAN with estimated parameter values. Each label of data points in the current period is matched with one of the labels in the previous period in the label matching phase. Using different DBSCAN parameters and different UMAP models in each period make DBPCES successful in evolving data streams.

Contributions of this study are on a few different topics. The data stream clustering algorithm that is developed in this study (DBPCES) does not require any data-dependent parameter and can adapt itself to the evolution of data. DBPCES is based on a state-off-art offline clustering algorithm DBSCAN that is not directly applicable to data streams. It can be said that DBPCES makes DBSCAN applicable to data streams and makes it a parameterless algorithm. Additionally, DBPCES makes UMAP adapt to the evolution of the data stream. Unlike DenStream, DBPCES makes it possible to visualize the entire clustering result because data streams are embedded into two dimensions and labels of different periods are matched.

The rest of the manuscript is organized as follows: Chapter 2 describes traditional data clustering, dimensionality reduction, data stream clustering, and clustering evaluation metrics, and examines DBSCAN and DenStream algorithms in detail. Chapter 3 contains a review of literature regarding the enhancements of DBSCAN algorithm, data stream clustering algorithms, dimensionality reduction algorithms, and the algorithms that use dimensionality reduction as a pre-processing step. Chapter 4 describes the algorithm developed in this study in detail. Then, in Chapter 5 the results

of DBPCES algorithm is presented and compared with DenStream and implementation of Zubaroğlu and Atalay. Finally, Chapter 6 concludes the study and describes possible future work.

CHAPTER 2

BACKGROUND INFORMATION

This chapter gives background information about the concepts that are used in our proposed algorithm. The chapter is divided into four parts. The first part describes what traditional clustering is. Additionally, it explains the methodologies used in each of the five main categories of traditional clustering algorithms and gives advantages and disadvantages of each category. It also investigates DBSCAN algorithm which is one of the most popular traditional density-based clustering algorithms and which is also used in this thesis. The second part describes what dimensionality reduction is. It also presents the categorization of dimensionality reduction algorithms and explains UMAP which is a popular graph-based dimensionality reduction algorithm. The third part describes data stream clustering and how it is different from traditional clustering. It elaborates on the categorization of data stream clustering algorithms and explains DenStream which is a popular density-based stream clustering algorithm based on DBSCAN. The last part gives information about 3 different clustering evaluation metrics which are used in the literature and evaluation of the algorithms used in this thesis.

2.1 Traditional Data Clustering

Data clustering is the task of grouping points of a dataset such that data points in the same group have more similarity than data points in the other groups. Data clustering is an unsupervised task which means it does not require actual labels of data points.

There are several clustering algorithms since there is no precise definition of a "cluster". Each clustering algorithm defines a cluster differently, therefore use a different

methodology to find clusters. Clustering algorithms can be divided into five main categories according to their methodology: hierarchical, partitioning, density-based, grid-based, and model-based clustering [4] [5] [6].

Hierarchical: Hierarchical clustering algorithms build a hierarchy of clusters. Hierarchical clustering algorithms are categorized into two groups: agglomerative and divisive. Agglomerative algorithms follow a bottom-up approach and start by assigning each data point to its own cluster and merges clusters step by step. Divisive algorithms follow a top-down approach and start by assigning all data points to a single cluster and divides clusters step by step. Both approaches use a dendrogram which is a tree structure for results. The dendrogram makes it easy to understand the structure of clusters. However, the time complexity of hierarchical clustering algorithms is very high and this makes them inapplicable even for medium-sized datasets. In addition, hierarchical clustering algorithms are very sensitive to outliers.

Partitioning: Partitioning clustering algorithms assume the number of clusters is known and splits data points according to cluster number using distance to the chosen cluster centroids. k cluster centers are chosen and the data points are assigned to the nearest cluster center. Partitioning clustering algorithms are easy to implement these algorithms require cluster count as an input although the number of clusters is often unknown. In addition, partitioning clustering algorithms are only suitable for hyper-spherical clusters.

Density-based: According to density-based clustering algorithms, a cluster is a high-density data region. Density-based clustering algorithms identify high-density regions and low-density regions. Then, high-density regions are labeled as clusters. Density-based clustering algorithms do not require the number of clusters as an input and can find arbitrary-shaped clusters. However, density-based clustering algorithms require some parameters as input and these algorithms are generally very sensitive to these parameters. Finding multi-density clusters is another problem of density-based clustering algorithms.

Grid-based: Grid-based clustering algorithms divide data space into a set of grid cells and assigns each object to an appropriate grid cell. Clusters are formed using these generated grid cells. After the generation of grid cells, low-density grid cells

are eliminated and adjacent dense cells are merged to form clusters. Grid-based algorithms can also find arbitrary-shaped clusters. The complexity of grid-based algorithms does not depend on data point count. They need grid size as an input. They are suitable for low dimensional data because their complexity depends on the number of features.

Model-based: Model-based algorithms assume that data distribution fits a mathematical model and try to find values of the parameters of such a model. Model-based algorithms are robust to noise and generally do not require many parameters. However, the performance of model-based clustering algorithms strongly depends on the chosen mathematical model.

Traditional clustering algorithms with their categorization are presented in the Figure 2.1. Additionally, the advantages and disadvantages of different clustering approaches are summarized in Table 2.1 [7].

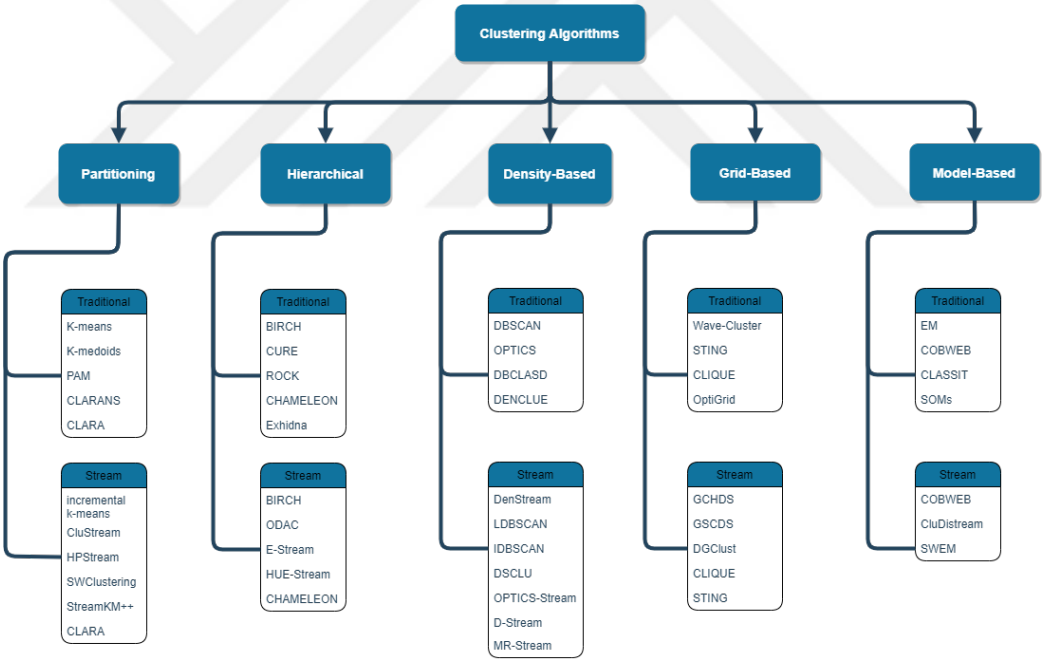


Figure 2.1: Clustering algorithms with their categories.

Density-based spatial clustering of applications with noise (DBSCAN) [8] is one of the most popular density-based clustering algorithms. DBSCAN forms the basis of density-based clustering algorithms. DBSCAN is a very powerful algorithm that can find arbitrary-shaped clusters without knowing the number of clusters. Most of the

Table 2.1: Advantages and disadvantages of clustering approaches.

Algorithm	Advantage	Disadvantage
Hierarchical	Informative output (dendrogram) No need for cluster count	High Complexity in high dimensional space Outlier sensitivity
Partitioning	Scalable and simple Suitable for hyper-spherical clusters	Predefined number of clusters Unsuitable for arbitrary-shaped clusters
Density-based	Arbitrary shaped clusters Noise robustness	Multi-density cluster difficulties Sensitive to predefined parameters
Grid-based	Arbitrary shaped clusters Fast execution time for large datasets Noise robustness	Predefined grid size Suitable for low dimensional data
Model-based	Noise robustness Small number of parameters	Strong dependency on the model Difficult estimation of cluster count

algorithms in the literature try to enhance the performance of DBSCAN algorithm and compares themselves with DBSCAN [9]. In this work, we have also proposed a density-based stream clustering algorithm that is based on DBSCAN. Therefore, DBSCAN algorithm is given in detail in the next section.

2.1.1 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

The main idea behind DBSCAN algorithm is to locate regions of high density. These high-density regions are separated from one another by regions of low density. The algorithm requires two parameters: ϵ and $minPoints$.

- $\boldsymbol{\epsilon}$: Maximum distance to consider two points as neighbors. In DBSCAN algorithm, a heuristic that uses a k-dist graph is suggested to select the $\boldsymbol{\epsilon}$ value. For plotting the k-dist graph, the distance between each point and its $\boldsymbol{k^{th}}$ nearest neighbor is calculated. These distances are sorted in ascending order. Then, the k-dist graph is plotted where the $\boldsymbol{y}$ axis corresponds to the sorted distance between each point and its $\boldsymbol{k^{th}}$ nearest neighbor. Maximum curvature (knee) in the k-dist graph is suggested as the optimal $\boldsymbol{\epsilon}$ value because it means that this distance corresponds to a density level and points whose distance to their $\boldsymbol{k^{th}}$ nearest neighbor is greater than this knee have different density. Figure 2.2 shows an example of a k-dist graph for k=3. For this example, the candidate $\boldsymbol{\epsilon}$

value is 0.25 which is shown with a dashed line in the figure.

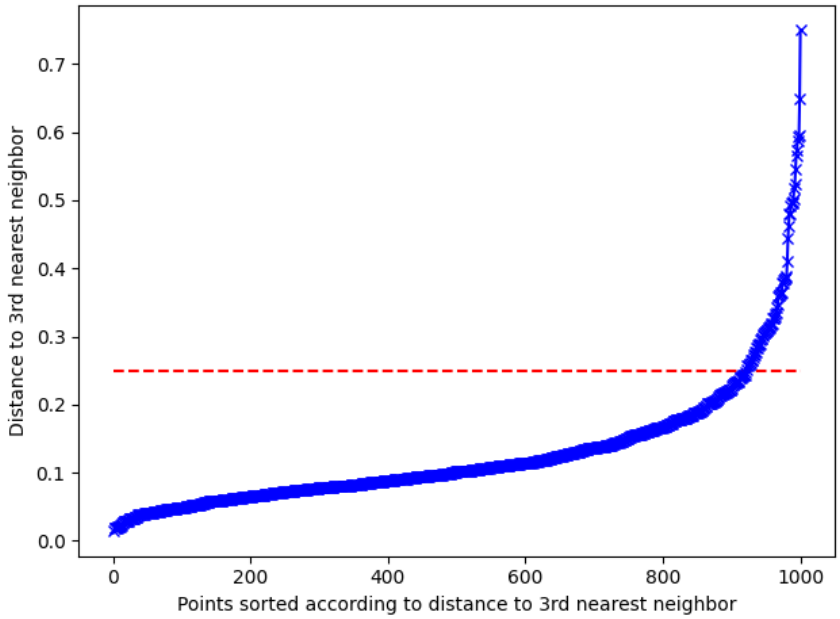


Figure 2.2: A sample k-dist graph.

- $\boldsymbol{minPoints}$: The minimum number of points to form a dense region.

ϵ-neighborhood: Set of all points in the circle with radius ϵ around a given point. The ϵ neighborhood of point p in the point set P is defined in Equation 2.1.

$$N_\epsilon(p) = \{q \in P \mid dist(p, q) \leq \epsilon\} \tag{2.1}$$

Density at point p: Number of points in ϵ-neighborhood of point p.

Dense region: For each point in the cluster, the circle with radius ϵ contains at least the minimum number of points ($minPoints$).

Core point: A point whose density is greater than $minPoints$.

Border point: A point whose density is smaller than $minPoints$ but it is in ϵ-neighborhood of a core point.

Noise point: A point which is neither core point nor border point.

Figure 2.3 shows examples of core, border, and noise points.

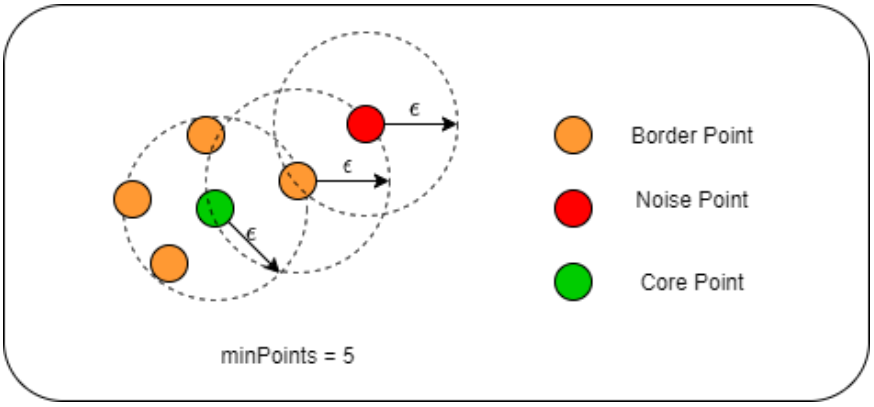


Figure 2.3: Examples of core point, border points and noise point.

Directly Density Reachable: A point b is directly density reachable from point a if the following conditions hold:

- a is a core point.
- b is in the ϵ-neighborhood of a ($b \in N(a)$).

Density Reachable: A point b is density reachable from point a if the following conditions hold:

- There is chain of points($a_1, a_2, a_3, ..., a_n$) where $a = a_1$ and $b = a_n$.
- For each i, a_i is directly density reachable from point a_{i+1}.

Density Connected: A point b is density connected to point a if there is a point c which both a and b is density reachable from point c.

Detailed pseudo-code of DBSCAN algorithm is given Algorithm 2.1. Algorithm 2.1 is the main algorithm that uses Algorithm 2.2 to find the neighborhood of a point and Algorithm 2.3 to create and expand clusters from core points. First, an arbitrary point p is selected and points at most ϵ away from this selected point ($N(p)$) is found using Algorithm 2.2. If this selected point p is a core point, a new cluster c is created and expanded by adding points in this ϵ-neighborhood ($N(p)$) to the cluster. If any of the newly added points is a core point, the points in the ϵ neighborhood of this newly added core point are also added to the same cluster recursively by Algorithm 2.3. After all density connected points are added to cluster c, a new arbitrary unvisited point

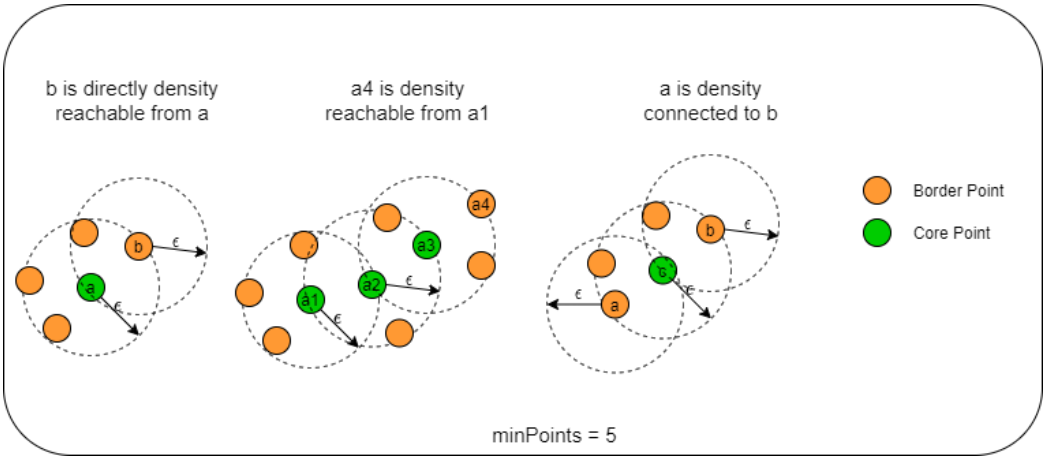


Figure 2.4: Density definitions of DBSCAN.

is selected and the same process is executed for this point. The algorithm finishes execution if there are no unvisited points left.

Algorithm 2.1 $DBSCAN(allPoints, epsilon, minPoints)$

for each $unvisited\ p \in allPoints$ **do**
  $p.visited \leftarrow True$
  $N_\epsilon(p) \leftarrow getNeighborhoods(p, allPoints)$
  if $|N(p)| >= minPoints$ **then**
    $C \leftarrow \{p\}$
    $expandCluster(C, allPoints, N(p))$
  else
    $point.visited = False$
  end if
end for

2.2 Dimensionality Reduction

As dimensionality increases, set of problems arises while analyzing data, this phenomenon is called "Curse of Dimensionality" [10]. With the increase of dimensionality, data points become sparse and data analysis algorithms need more training data because there are more feature combinations and data analysis algorithms should learn them to be successful on unseen data. This is called data sparsity and it is

Algorithm 2.2 $getNeighborhoods(p, allPoints)$

$neighbors \leftarrow \{\}$
for each $unvisited\ neighbor \in allPoints$ **do**
  $dist \leftarrow dist(p, neighbor)$
  if $dist < \epsilon$ **then**
    $neighbors \leftarrow neighbors \cup neighbor$
  end if
end for
return $neighbors$

Algorithm 2.3 $expandCluster(C, allPoints, neighborhood)$

for each $neighbour \in neighborhood$ **do**
  if $neighbour\ is\ not\ visited$ **then**
    $neighbour.visited \leftarrow True$
    $N_\epsilon(neighbour) \leftarrow getNeighborhoods(neighbour, allPoints)$
    if $|N(neighbour)| >= minPoints$ **then**
      $expandCluster(C, allPoints, N_\epsilon(neighbour))$
    end if
  end if
  if $neighbour\ is\ not\ belong\ to\ a\ cluster\ yet$ **then**
    $C \leftarrow C \cup neighbour$
  end if
end for

the first aspect of curse of dimensionality. The second aspect of curse of dimensionality is distance concentration. Most of the data analysis algorithms uses pairwise distances between points. However, in high dimensions pairwise distances between points converge to the same value. This makes differentiating similar or dissimilar data points harder.

To overcome curse of dimensionality, dimensionality reduction techniques are available in the literature [11] [12]. Dimensionality reduction is defined as the projection of high dimensional data into a low dimensional space by selecting a subset of features (feature selection) or constructing a new set of features in a lower-dimensional space (feature transformation or feature extraction). Dimensionality reduction techniques can be categorized into three main groups: data-dependent, data-independent, and graph-based [12].

Data-dependent techniques need the whole data to be able to perform projection. Principal Components Analysis (PCA) [13], Multidimensional scaling (MDS) [14], Auto-Encoder (AE), Linear Discriminant Analysis (LDA) [15], Maximum Margin Criterion (MMC) [16] are examples of data-dependent dimension reduction algorithms. Applying these algorithms to data streams is not possible because they need whole data. In the literature, there are modified versions of these algorithms which try to make them applicable to data streams. Incremental PCA (IPCA) [17], Candid Covariance-free Incremental PCA (CCIPCA) [18], Fast Iterative Kernel PCA [19] are the modified versions of PCA algorithm. iMDS, and scMDS [20] improve MDS algorithm in order to make it applicable on data streams. Incremental LDA (ILDA) [21] and IDR/QR [22] are the streaming versions of LDA algorithm. Incremental MMC (IMMC) [23] makes MMC suitable to stream environment.

Data independent techniques use random projections and they do not require the whole data for transformation. Random Projection (RP) [24], Compressed Sensing (CS) [25], Hashing Trick (HT) [26], Locality Sensitive Hashing (LSH) [27] are data independent dimensionality reduction techniques in the literature. Data independent techniques are convenient for data streams by nature [12].

Graph-based techniques are also data-dependent. They build a graph-based representation of data points according to data point similarity such as neighborhood graphs.

Projections are made based on this representation. Isometric Mapping (Isomap) [28], t-distributed Stochastic Neighbor Embedding (tSNE) [29], Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) [30] are graph-based dimensionality reduction techniques in the literature. Nonlinear Manifold Learning For Data Stream [31], Streaming Isomap (S-Isomap) [32] are incremental versions of Isomap that are applicable to data streams. Unlike other graph-based techniques, UMAP has the capability to embedding new data points to an existing low dimensional space.

Traditional dimensionality reduction algorithms and data stream dimensionality reduction algorithms along with their categorization are provided in the Figure 2.5 [12].

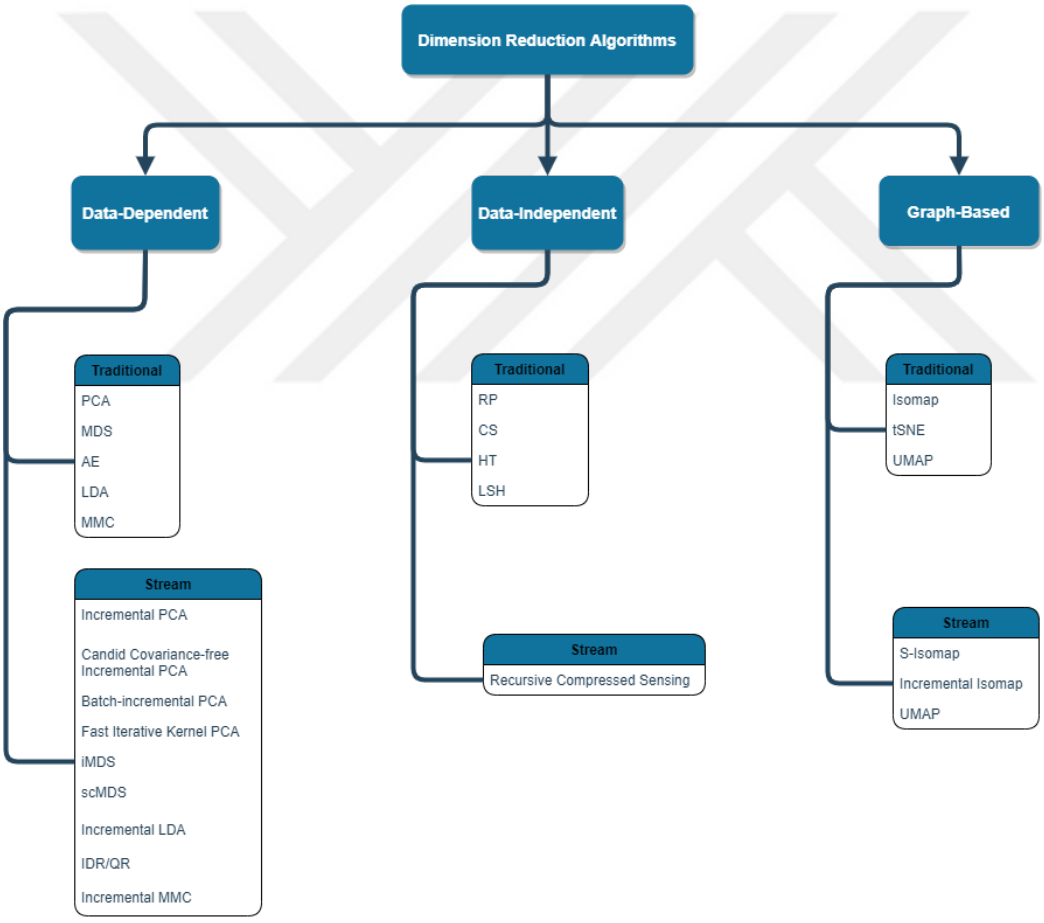


Figure 2.5: Dimensionality reduction algorithms with their category.

As mentioned before, data streams are often high-dimensional. Therefore, dimensionality reduction can be used as a pre-processing step before applying traditional

clustering algorithms. The proposed algorithm in this study also uses UMAP which is a graph based dimensionality reduction algorithm as pre-processing step before clustering. Therefore, UMAP algorithm is given in detail in the next section.

2.2.1 Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP)

UMAP [30] is one of the most popular non-linear dimensionality reduction algorithms which preserves most of the global structure of dataset and has a low execution time. UMAP is a manifold learning technique based on the topological structure in multi-dimensional space. In order to construct topological structure of data in its original dimension, UMAP builds a weighted graph where edges connect pair of data points [33]. The weights of edges correspond to likelihood of connection of data points. A radius outwards each data point is extended and only pair of points whose radii overlap is connected by an edge. The radius is selected locally using distance to each point' s k^{th} nearest neighbor. The weights of edges are also reduced as the radius grows. After the construction of topological structure of data, UMAP creates low-dimensional space using this weighted graph. With UMAP, it is possible to embed new unseen points into an existing low-dimensional space. UMAP has the capability of transforming part of the data to lower dimensions and learning this low-dimensional space. Therefore, the new data points can be transformed into this learned low dimensional space by UMAP. This makes UMAP applicable to data streams. However, UMAP does not have the capability to adapt concept drift [34].

2.3 Data Stream Clustering

A data stream is a continuous and infinite sequence of data points that are generated in real-time. It has no discrete beginning or end. Each data point comprises a set of attributes. Equation 2.2 shows the structure of a data stream S where n is the number

of points and d is the number of attributes.

$$S = \{x_1, x_2, x_3, ..., x_n\}\ where\ (n \to \infty)$$
$$x_i = [x_i^1, x_i^2, ..., x_i^d] \tag{2.2}$$

A data stream point is generated in real-time and mostly without a true cluster label. Data stream clustering is the task of clustering data that arrives continuously. Clustering a data stream has different challenges than traditional data clustering:

- A data stream is infinite, often high dimensional, and it dynamically changes.
- Storing whole data for post-processing is not possible because of memory constraints. Therefore data needs to be processed in a single pass, online, with low execution time.
- Data stream points might evolve over time. If it happens, concept drift arises and the data clustering model no longer holds. The data stream clustering algorithm should update the model in order to adapt to concept drift for accurate results.

It is almost impossible to apply traditional clustering algorithms to data streams because of the challenges described above. There are several data stream clustering algorithms in the literature. Data stream clustering algorithms can be divided into two categories according to how they process data:

Fully Online Clustering Approach: Fully online clustering algorithms always keep an up-to-date clustering result. Clusters are updated after receiving each data point. Clustering of Evolving Data-streams into Arbitrary Shapes (CEDAS) [35], Davies-Bouldin Index Evolving Clustering Method for streaming data clustering (DBIECM) [36], Fast Evolutionary Algorithm for Clustering data streams (FEAC-Stream) [37], Adaptive Streaming k-Means [38] are examples of fully online stream clustering algorithms.

Online-Offline (Two-Phase) Clustering Approach: To be able to handle massive and high-dimensional data, most of the data stream clustering algorithms apply online-offline clustering approach. Online-offline clustering approach consists of two steps. The first step is called the online phase or data abstraction phase. Data points are summarized, similar points are grouped together and stored in special data structures in the online phase. These groups are called micro-clusters. Whenever a clustering request arrives or at certain periods, the offline phase which is called the clustering phase is performed on the results of the online phase. In the offline phase, a modified version of one of the traditional clustering algorithms is applied to the micro-clusters, and similar micro-clusters are grouped together as macro-clusters. Figure 2.6 shows how the online-offline clustering approach works on the data stream [1].

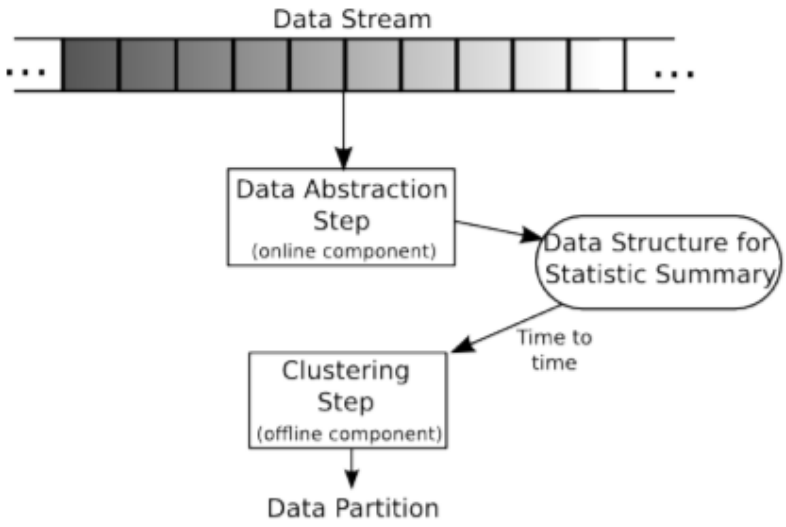


Figure 2.6: Online-offline clustering approach (adapted from Silva et al. [1]).

Data stream clustering algorithms also can be categorized into five main groups according to their clustering model similar to traditional clustering algorithms: hierarchical, partitioning, density-based, grid-based, and model-based clustering. Data stream clustering algorithms with their categorization are also listed in the Figure 2.1.

DenStream [39] is one of the most popular two-phase density-based stream clustering algorithms. DenStream does not require the number of clusters as input and can successfully find arbitrary-shaped clusters. The offline phase of DenStream is a variation

of DBSCAN algorithm. In this study, we also improve DBSCAN for data streams and compare our results with DenStream algorithm. DBSCAN is described in Section 2.1.1 and DenStream algorithm is given in the next section.

2.3.1 DenStream

DenStream is a density based two-phase data stream clustering algorithm which uses a DBSCAN variation in its online phase. DenStream follows the micro-cluster concept to summarize data streams effectively and to record regions that are dense or might become dense in the future. Micro-cluster concept is extended to potential micro cluster (PMC) and outlier micro cluster (OMC) in DenStream. PMCs and OMCs are circular regions which is described with a radii, weight and center. The PMCs and OMCs differs in density. At the beginning of DenStream, DBSCAN is applied on $initialdata$ whose size is determined with user-defined is parameter. The first PMCs are the clusters generated by DBSCAN in the initialization. Each point is associated with a weight which is equal to 1 in its arrival time and decreases with time. For each incoming data point, the nearest PMC is found and the algorithm merges this point to its nearest PMC if merging does not violates PMC' s merging conditions. If the point can not be merged to nearest PMC, the algorithm tries to merge it to nearest OMC. If merging is unsuccessful, a new OMC is created and the point is merged into that newly generated OMC. Both PMCs and OMCs are associated with a weight which is the sum of weights of points they cover. DenStream maintains PMC list and OMC list in its online phase. A OMC can transform to a PMC if it satisfies density requirement of PMC. The algorithms checks weight of PMCs and OMCs at certain periods and removes them from list if their weight below threshold. At certain periods (rp), online phase is run on PMC list. DBSCAN is modified such that the inputs are not data points, they are circular regions (PMCs) created in online phase.

In the online phase, the data stream is summarized and similar points are grouped as micro-clusters. A special data structure called cluster feature (CF) given in Equation

2.3 is used for this purpose.

$$CF = (CF^1, CF^2, w) \tag{2.3}$$

For a given d-dimensional n (number of points in the micro-cluster) data points $X_1, X_2, ..., X_n$ and their creation time $T_1, T_2, ..., T_n$, CF^1 given by Equation 2.4 is the linear sum of data points, $CF2^d$ given by Equation 2.5 is the sum of squared data points and w given by Equation 2.6 is the sum of the weight of points [39]. The center c and radius r of these data points are calculated using Equation 2.7 and Equation 2.8, respectively [39].

$$CF^1 = \sum_{j=1}^{n} X_j \tag{2.4}$$

$$CF^2 = \sum_{j=1}^{n} {X_j}^2 \tag{2.5}$$

$$w = \sum_{j=1}^{n} f(t - T_j) \tag{2.6}$$

$$c = \frac{CF^1}{n} \tag{2.7}$$

$$r = \sqrt{\frac{CF^2}{n} - (\frac{CF^1}{n})^2} \tag{2.8}$$

DenStream associates each point with a weight according to its arrival time. As mentioned before, the characteristics of data points in data stream can change over time. To keep micro-clusters suitable to recent data, new data points should have more effect on the result. To achieve this, damped window model was used in DenStream. By this way, DenStream can adapt to concept drift. As an aging function, the exponential fading function given in Equation 2.9 was used.

$$f(t) = 2^{-\lambda(t_c - t_o)} \tag{2.9}$$

In the above equation, λ is the fading factor, t_c is the current time and t_o is creation time. With the use of the damped window model, previous data points are not discarded completely but newer points become more important with higher weight. The fading factor λ determines the importance of previous data points. A higher value of λ means, lower importance of previous data. Points in the same window (processing in same iteration) gets the same weight.

DenStream algorithm has seven main parameters:

• is : Number of data points clustered with DBSCAN for initialization.

• rp : Offline phase execution period in terms of number of data points.

• ws : Number of data points received together according to speed of stream.

• λ : It is used in fading function. It defines how much older points' weights decrease.

• ϵ : It is used in setting limits of the neighborhood. Points that are at most epsilon distance away from the specific point are used as neighbors of this point.

• β : There are two types of micro-clusters in online phase of DenStream: potential micro-clusters and outlier micro-clusters. The role of potential micro-clusters and outlier micro-clusters could interchange. The parameter β is used with parameter μ to determine whether a micro-cluster is outlier or potential as defined in 2.10b and 2.12b.

• μ : It is used in off-line phase as density condition of core micro cluster as defined in 2.11b.

DenStream extends micro-cluster concept to potential micro-cluster (PMC), outlier micro-cluster (OMC), and core micro-cluster (CMC). In the online phase, OMC list and PMC list are maintained. Definition of types of micro-clusters are as follows:

Potential Micro-Cluster: A potential micro-cluster, given in Equation 2.10 , is a core micro-cluster candidate due to its weight being larger than the predefined threshold $\beta * \mu$.

$$PMC = (CF^1, CF^2, w) \tag{2.10a}$$

$$w \geq \beta * \mu \tag{2.10b}$$

$$r < \epsilon \tag{2.10c}$$

$$0 < \beta \leq 1 \tag{2.10d}$$

Core Micro-Cluster: A core micro-cluster, given in Equation 2.11 , is a core micro-cluster due to its weight being larger than the predefined threshold μ.

$$CMC = (w, c, r) \tag{2.11a}$$

$$w \geq \mu \tag{2.11b}$$

$$r < \epsilon \tag{2.11c}$$

Outlier Micro-Cluster: An outlier micro-cluster, given in Equation 2.12, is not a potential micro-cluster because its weight is smaller than the predefined threshold $\beta * \mu$. Still, its weight is large enough to make it a potential micro-cluster candidate.

$$OMC = (CF^1, CF^2, w, t_o) \tag{2.12a}$$

$$w < \beta * \mu \tag{2.12b}$$

$$r < \epsilon \tag{2.12c}$$

$$0 < \beta \leq 1 \tag{2.12d}$$

The algorithm tries to merge a new data point to the nearest PMC first. If PMC radius condition given in Equation 2.10c continues to hold after adding data point to nearest PMC, data point is merged to nearest PMC. Otherwise, data point is merged to nearest OMC if radius condition given in Equation 2.12c is not broken for that OMC after merging. If it is merged with neither the nearest OMC nor PMC, the algorithm generates a new OMC, and this OMC is added OMC list. OMCs can transform into PMCs in the online phase in case the PMC density condition given in Equation 2.10b holds after merging the new point to OMC. In addition, OMCs and PMCs that do not satisfy density requirements are removed from OMC list and PMC list.

In the offline phase, a variation of DBSCAN is used for clustering. The input of the offline phase is the potential micro-clusters with their weights which are the output of the online phase. Each potential micro-cluster is regarded as a virtual point at the center of that PMC with weight w. DenStream changes directly density-reachability definition of DBSCAN algorithm because it uses PMCs which have weight and radius instead of data points. A PMC can contain points at most ϵ away from its center because its radius is restricted to ϵ by definition. Therefore, a PMC c_p is directly density-reachable from another PMC c_q, if the weight of c_q is above μ and the distance between centers of c_q and c_p is not greater than $2 * \epsilon$.

2.4 Clustering Evaluation Metrics

Evaluating the performance of clustering algorithms is not as trivial as calculating the precision and recall of classification algorithms. Clustering algorithms do not have actual labels of data points. They only group data points according to their similarity. Therefore, it is not possible to calculate the precision or recall of the clustering algorithm. There are several clustering evaluation metrics in the literature that try to measure the performance of clustering. Some of the evaluation metrics which is also used in the evaluation of the algorithm suggested in this study are as follows:

Silhouette Score: It is used to evaluate the performance of clustering by measur-

ing separation distance between clusters. For a data point i, silhouette score $S(i)$ is calculated through the data point's average distance to the points in the same cluster ($a(i)$) and the smallest average distance to all points in any other cluster ($b(i)$). For a data point i in the cluster C_i, silhouette score ($S(i)$) is calculated using Equation 2.13 where $d(i, j)$ is the distance between point i and point j in the cluster C_i. Possible values of silhouette score are in the range of [-1,1]. If $S(i)$ is close to 1, the data point is appropriately clustered. If it is close to -1, the data point is clustered incorrectly.

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{2.13}$$

where

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j)$$

Adjusted Rand Index (ARI): ARI can be used to compare the results of the two clustering algorithms (without actual labels) to determine whether label assignments of two clusterings have similar results. Additionally, it can be also used in order to evaluate the performance of a clustering algorithm according to actual class labels, if they are available. We use ARI in the latter way.

Rand Index (RI) is the basis of ARI. Rand Index calculates the percentage of matched pairs of two clusterings. Suppose a dataset contains n data points. There are $\binom{n}{2}$ unordered pairs of data points. For the calculation of RI, number of pairs of data points that are in same cluster in two clusterings (a) and number of pairs of data points that are in different clusters in two clusterings (b) are calculated. Then, RI is calculated using Equation 2.14. The range of the rand index value is [0, 1]. RI value closer to 1 means that the results of the two clustering algorithms are similar.

$$RI = \frac{a + b}{\binom{n}{2}} \tag{2.14}$$

When the number of clusters increases in two clusterings, number of agreed data point pairs may increase even for random clustering. This leads to high RI for random clustering. To solve this, ARI is used for evaluation. ARI is the corrected-for-chance

version of RI evaluation metric. For the calculation of ARI, the contingency table is created first.

Contingency Table: Assume that there is a dataset with n data points. The first clustering result comprises of r clusters:$\{X_1, X_2, \ldots, X_r\}$ and the second clustering algorithm partitioned data into s clusters:$\{Y_1, Y_2, \ldots, Y_s\}$. Overlapping of two clustering algorithms are summarized in contingency table where each n_{ij} denotes the number of objects in common between X_i and Y_j. When $n_{ij} > 0$, it represents an overlap between the partitions. Sum of each row (a_i) is equal to the number of elements in the partition class X_i, and the sum of each column (b_j) represents the number of elements in the partition class Y_j. Figure 2.7 shows the structure of the contingency table. In the case of the existence of actual labels, they can be used as one of the clustering results in the contingency table. The contingency table is filled by looping through each cluster X_i and counting the number of objects whose actual class is Y_j.

Contingency Table

X \ Y	Y1	Y2	...	Ys	sums
X1	n_{11}	n_{12}	...	n_{1s}	a_1
X2	n_{21}	n_{22}	...	n_{2s}	a_2
⋮	...	...	...	...	...
Xr	n_{r1}	n_{r2}	...	n_{rs}	a_r
sums	b_1	b_2	...	b_s	

Figure 2.7: Contingency table

Adjusted rand index is calculated using the formula in Equation 2.15 with values in the contingency table. Terms in the ARI formula are explained below:

Index: The sum of the number of pairs of data points that are in same cluster in both clusterings and the number of pairs of data points that are in different clusters in both clusterings.

Expected Index: Expected value of a cell n_{ij} in a contingency table is calculated by $(a_i * b_j)/n$ [40]. For ARI calculation, this formula is applied to pairs of data points.

Maximum Index: Index value in case of perfect clustering. Only one cell in each row and column is greater than zero if the clustering algorithm clusters data perfectly. Therefore, this value can be obtained from row and column sums.

The range of the adjusted rand index value is [-1, 1]. ARI value closer to 1 means that the results of the two clustering algorithms are perfectly matched.

$$ARI = \frac{\overbrace{\sum_{ij}\binom{n_{ij}}{2}}^{Index} - \overbrace{[\sum_{i}\binom{a_i}{2}\sum_{j}\binom{b_j}{2}]/\binom{n}{2}}^{Expected\ Index}}{\underbrace{\frac{1}{2}[\sum_{i}\binom{a_i}{2}+\sum_{j}\binom{b_j}{2}]}_{Maximum\ Index} - \underbrace{[\sum_{i}\binom{a_i}{2}\sum_{j}\binom{b_j}{2}]/\binom{n}{2}}_{Expected\ Index}} \tag{2.15}$$

Purity: Evaluation of clustering can be made by measuring to what extend predicted clusters contain data points from a single class. First, a contingency table is created with clustering results and actual labels. Then, purity is calculated using Equation 2.16 with values in the contingency table where n is the number of data points, r is the number of actual clusters and s is the number of predicted clusters. Possible values of purity are in the range of [0,1]. A purity value closer to 1 means good clustering because it is the percentage of data points clustered correctly.

$$Purity = \frac{1}{n}\sum_{i=1}^{r} max_j|n_{ij}| \qquad \textit{where } 1 < j < s \tag{2.16}$$

CHAPTER 3

RELATED WORK

This chapter provides a review of studies related to our study in 4 parts. The first part contains algorithms in the literature that are variations of DBSCAN. The articles given in the first part try to enhance the DBSCAN algorithm by providing solutions to some of the problems of DBSCAN. The second part consists of overview of data stream clustering algorithms in the literature. The third part contains reviewed papers about dimensionality reduction. Finally, the last part contains algorithms in the literature which use dimensionality reduction as a pre-processing step before data analysis.

DBSCAN is a very successful algorithm in finding arbitrary-shaped clusters although it does not require the number of clusters as an input parameter. However, it has some problems:

(**P.1**) It has the problem of high complexity. The worst-case complexity of DBSCAN is $O(n^2)$ where n is the number of data points.

(**P.2**) It has two important parameters which have a significant effect on the performance of the algorithm: ϵ and $minPoints$. They are hard to decide and require domain knowledge. In DBSCAN, k-dist graph is suggested for the estimation of ϵ value but the knee in the k-dist graph is determined manually.

(**P.3**) It can not find clusters that have large differences in densities, because it uses a single $\epsilon - minPoints$ combination for all clusters.

(**P.4**) As a distance metric, it is very common to use Euclidean distance. However, in high dimensions differentiating similar points becomes harder as explained in detail in Section 2.2.

3.1 Variations of DBSCAN

In literature, there are variations of the DBSCAN algorithm which attempt to propose solutions to the problems of DBSCAN.

El-Sonbaty et al. [41] tried to solve the high computational cost problem of the DBSCAN algorithm defined at (**P.1**) by using CLARANS [42] which is a partitioning clustering algorithm based on randomized search. In the pre-processing step, the dataset is analyzed and partitioned using CLARANS. DBSCAN scans the whole dataset to find core objects and this leads to high computational cost. With the use of CLARANS, the search space is partitioned and DBSCAN can be run on each partition in parallel. Therefore, expensive neighborhood calculations are made between points in a single partition. Finally, the nearest dense regions from different partitions are merged to find the final clustering. El-Sonbaty et al. speed up DBSCAN by limiting search space.

Hou et al. [43] developed DSets-DBSCAN to make a parameter-free DBSCAN algorithm. DSets-DBSCAN is designed to work only on image segmentation. DSets-DBSCAN algorithm consists of two main steps. In the first step, histogram equalization is applied to the pairwise similarity matrix of input data in order to eliminate the need for user-defined parameters defined at (**P.2**). Then, DSets [44] algorithm is used in creating dominant sets. After the DSets algorithm, DBSCAN extends dominant sets into clusters. Histogram equalization transformation eliminates parameters by generating identical clustering results for all parameters. DSets-DBSCAN makes a parameter-free version of DBSCAN but it works only on images.

Yu et al. [45] developed K-nearest neighbors DBSCAN (KNNDBSCAN) algorithm in order to reduce the number of user-defined density threshold parameters. KNNDBSCAN requires only a single user-defined input parameter "k". In other words, KNNDBSCAN partially solves the problem of DBSCAN defined at (**P.2**). The algorithm is composed of two steps. In the first step, the dataset is partitioned into fuzzy clusters (FCs). For this purpose, kernel density estimation (KDE) based K-nearest neighbor algorithm is developed. For each fuzzy cluster, local ϵ and local $minPoints$ are calculated according to entropy theory. Entropy is a measure of data order. Entropy is

smaller for ordered data and larger for scattered data. The algorithm uses entropy for the evaluation of kernel density estimation performance. In the second step, each fuzzy cluster is clustered in parallel using DBSCAN, and global ϵ is calculated using local ϵ of each fuzzy cluster. KNNDBSCAN requires only a single user-defined parameter and speeds up DBSCAN because final clustering is performed in parallel.

VDBSCAN [46] enhances the DBSCAN algorithm in order to find clusters with different densities. VDBSCAN tries to solve the problem of DBSCAN defined at (**P.3**). It consists of two steps. In the first step, it plots sorted k-dist graph as it suggested in DBSCAN algorithm. If the dataset contains varied density clusters, there will be multiple smooth curves connected by variational ones. Each smooth curve corresponds different density level. The algorithm uses each knee in the plot as a different ϵ value. After multiple ϵ values are determined, it runs DBSCAN for each ϵ value starting from the smallest one. Labeled points are excluded from the dataset and are not clustered again. It can find different density clusters because it does not use a single ϵ value. However, determination of ϵ value is not done automatically and VDBSCAN still requires a user-defined $minPoints$ parameter.

Density Clustering Based on Radius of Data (DCRBD) [47] is another enhancement of the DBSCAN algorithm. DCRBD tries to solve the problem of determining density threshold parameters defined at (**P.2**). DCRBD consists of two stages. In the first stage, data space is partitioned into circular regions. Each point must belong to at least one circle. Since one point may belong to more than one circle, the partitioned dataset consists of several overlapped circular regions. The optimal value of ϵ is determined from the overlapped circle region that spans all dataset. In the second stage, DBSCAN is applied to each region. DCRBD solves the determination of the density parameters problem of DBSCAN. In addition, DCRBD is more efficient than DBSCAN because it calculates pairwise distances only between points in each region.

Manisha et al. [48] and Elbatta et al. [49] developed their algorithms based on DBSCAN in order to find varied density clusters. These algorithms solve two important problems of DBSCAN defined at (**P.3**) and (**P.2**). Both used a k-dist graph to automatically estimate different ϵ values automatically for each density level similar to VDBSCAN. However, the algorithm proposed by Manisha et al. does not eliminates

all parameters. It estimates only ϵ values.

There are many variations of DBSCAN. Some of them eliminate parameter value determination problem while others try to solve cost and curse of dimensionality problems. However, none of them make DBSCAN applicable to data streams.

3.2 Data Stream Clustering Algorithms

Considering the characteristics of data streams, it is not possible to run directly the DBSCAN algorithm on data streams. There are clustering algorithms in the literature that can be applied directly to data streams [50]. While CEDAS, DBIECM, FEAC-Stream, Adaptive Streaming k-Means and C^{2}ICM are fully online data stream clustering algorithms, DenStream and its variations adopt two phase data stream clustering approach.

CEDAS [35] is a density-based fully online clustering algorithm. Data summary is kept in micro-clusters and micro-clusters that are above a pre-defined threshold and their connectivity with other clusters is kept in a graph structure. The final clustering result can be extracted from this graph structure.

DBIECM [36] is a distance-based fully online clustering algorithm. DBIECM tries to add each newly received data to an existing cluster. For this, it calculates the distance of this new data point to all existing clusters. The data point is added to all clusters whose radius is greater than the new point's distance to the cluster. If the distance between each cluster and the data point is greater than the user-defined maximum radius r_0, a new cluster is generated with this data point.

FEAC-Stream [37] is a partitioning fully online clustering algorithm. FEAC-Stream is developed based on k-means algorithm. The number of clusters is estimated using an evolutionary algorithm. Evaluation of FEAC-Stream is made during execution and the algorithm adjusts its parameters to improve evaluation results. FEAC-Stream maintains up-to-date clustering results and does not store data summary like any other fully online clustering algorithms.

Adaptive Streaming k-Means [38] is another partitioning fully online clustering al-

gorithm based on k-means algorithm similar to FEAC-Stream. Adaptive Streaming k-means eliminates the need for the number of clusters parameter from a user by estimating k value based on data distribution. Estimation of k value is made for each feature of the data separately because each feature can show different distribution. Therefore, there exist different candidate k values. Clustering is made using all of the candidate k values and the silhouette coefficient is calculated for the result of each clustering and k value which has the maximum silhouette coefficient is selected.

Cover-Coefficient-based Incremental Clustering Methodology (C^2ICM) [51] is a incremental clustering algorithm for dynamic information processing. In a document database, documents might be added or deleted over time. Therefore, the clusters should be updated with these changes. This is the motivation behind C^2ICM. C^2ICM is a centroid-based incremental clustering algorithm that does not require the number of clusters. The number of clusters and cluster seeds are determined with Cover-Coefficient-Based Clustering Methodology (C^3M) [52]. For the estimation of the number of clusters, a document by term matrix (D-matrix) is created. In the document by term matrix, rows correspond to documents in the database and columns correspond to words in the documents. The matrix is filled with the number of occurrences of each word in each document. Then, the document by document matrix (C-matrix) is computed using this D-matrix. The elements in C-matrix indicate the relationship between documents based on a two-stage probability experiment. From the C-matrix, seed powers of documents are calculated and cluster seed documents are selected. Then, non-seed documents are assigned to one of the seeds. C^2ICM is the incremental version of C^3M. With each update of the document database, cluster seed powers of the documents are calculated and cluster seeds are selected. The algorithm tries to assign newly added documents, the documents cannot be merged any cluster in the previous step and members of clusters whose seed document is removed to the new cluster seeds. These steps are repeated in each document database update. C^2ICM requires a D-matrix which makes it applicable to only text clustering.

DenStream described section 2.3.1 is one of the popular density-based stream clustering algorithms. There are some variations of the DenStream algorithm in the literature. Chen et al. [53] tried to improve the accuracy of DenStream algorithm by adding a new retrospect phase to it. rDenStream, which means DenStream with ret-

rospect phase, gives a second chance to the micro-clusters which are discarded by the DenStream algorithm. SDStream algorithm [54] is another density-based stream clustering algorithm which is a variant of the DenStream algorithm. The difference between DenStream and SDStream is their window models. SDStream uses a sliding window model. However, the variations of DenStream require data-dependent parameters that have tremendous effects on the performance of clustering similar to DenStream.

Although there are many data stream clustering algorithms in the literature, most of them require data-dependent parameters while some of them suffer from curse of dimensionality or concept drift. There is no algorithm that solve all these three problems.

3.3 Dimensionality Reduction Algorithms

High dimensionality makes data analysis difficult. Therefore, there are many dimensionality reduction algorithms in the literature. PCA [13] is a dimensionality reduction technique that linearly maps data to lower-dimensional space by maximizing the variance of the data in low-dimensional space. PCA constructs the covariance matrix of the data and computes eigenvectors on constructed matrix. Eigenvectors that have the largest eigenvalues are the principal components that reconstruct most of the variance of the original data. PCA cannot be applied to data streams because the covariance matrix is constructed using all of the data. However, there are variations of PCA that make it applicable to data streams. Bahri et al. [12] analyze dimensionality reduction techniques that are applicable to data streams in their paper. Incremental PCA (IPCA) tries to update eigenvectors of images using previous coefficients [12]. Candid Covariance-free Incremental PCA (CCIPCA) tries to update eigenvectors without computing the covariance matrix for each new incoming data point [12]. The limitation of these two modifications is that both use images as high-dimensional data and are not tested on different types of data [12]. Ross et al. [55] develop a batch-incremental version of PCA. Yu et al. [56] enhance PCA such that it iteratively update the covariance matrix. However, PCA and its variations are linear dimensionality reduction algorithms and non-linear relationship between data points

cannot be preserved in low-dimensional space with a linear dimensionality reduction algorithms.

3.4 Dimensionality Reduction as a pre-processing step

There are a few data stream mining algorithms in the literature that apply dimensionality reduction before execution to solve high computation and storage cost of data analysis on high dimensional data.

Feng et al. [57] proposed an online classification algorithm FIKOFrame. FIKOFrame proposes FIKDR algorithm that is extension of Fast Iterative Kernel PCA (IKPCA) [19] algorithm which is a variant of PCA. FIKDR makes convergence of IKPCA faster and reduces storage complexity. FIKOFrame consists of three steps. The first step is initialization which creates Back-Propagating Neural Network (BP-NN). In the second step, BP-NN is trained with training dataset and Feature Database (FDB) is created. Both initialization and training is offline. In the online step (classification step), each data point is fed to classification engine. Classification engine extracts features nonlinearly using kernel functions and reduces dimensionality using features in FDB. After dimensionality reduction, data stream is classified.

Bahri et al. [58] proposed an algorithm UMAP-kNN that uses UMAP for dimensionality reduction before kNN classification on data streams in a batch-incremental manner. UMAP-kNN is composed of 3 steps. In the first step, data is divided into fixed size disjoint chunks. The algorithm assumes that a data stream arrives as a group of data points. The second step of the UMAP-kNN is data pre-processing. Each chunk is embedded into a lower dimension using UMAP independently. In order to preserve topological information of the stream and keep stability between embeddings, each chunk' s embedding is fed with the embedding of previous chunk. The last step of the UMAP-kNN is the kNN classification step. After embedding of a chunk, the first half of the chunk is fed into sliding window and the labels of second half is predicted. Computational cost of kNN on high dimensional data is decreased by embedding of data stream into low dimension using UMAP. The performance of neighborhood based kNN is not affected by embedding because UMAP preserves both local and

global structure of the data.

Zubaroğlu and Atalay [34] proposed a batch incremental data stream clustering algorithm that applies UMAP before k-means algorithm to data streams. The main limitation of the algorithm is the user-defined parameter k. In addition, the algorithm can find only hyper-spherical clusters because of k-means limitations.

As mentioned previously, there is no prior information about data streams. Therefore, a data stream clustering algorithm should require as few data-dependent parameters as possible. Additionally, a data stream clustering algorithm should cluster data in a single pass and with a low execution time. In this study, we develop a density-based data stream clustering algorithm that clusters data without any density parameter. The density parameters are estimated using a heuristic developed in this study. Because of the limited time constraint, data points are embedded into two dimensions, and estimation is made based on embedded points. Embedding also reduces the execution time of DBSCAN algorithm used in the clustering part. Additionally, visualization of clusters is possible with the use of the label matching algorithm and dimensionality reduction.

CHAPTER 4

PROPOSED METHOD

Our proposed algorithm DBPCES is inspired by the algorithm developed by Zubaroğlu and Atalay [34]. DBPCES continuously embeds streaming data into two dimensions using UMAP and clusters this embedded data using DBSCAN. We use DBSCAN which is a density-based clustering algorithm because it does not require the number of clusters as a parameter and it can find arbitrary-shaped clusters.

UMAP has the capability of embedding new data using a previously generated model on preceding data. This makes UMAP applicable to streaming data. However, concept drift is very common in streaming data. Using a single model on the whole data causes the clustering algorithm to fail if there is a concept drift. Therefore, DBPCES generates UMAP model periodically. It uses $initial\ data$ in each period in order to generate a model and embeds each $embedding\ data$ into this model until the upcoming period. After each period, DBSCAN algorithm runs in order to cluster the data. Parameters of DBSCAN which are described in Section 2.1.1 are very important for clustering correctly and hard to decide. Therefore, DBPCES develops a heuristic to estimate the values of parameters of DBSCAN. Since DBSCAN runs after each period, there is no correspondence between labels of periods. To match labels of each period, a certain amount of data ($matching\ data$) from the previous period is also embedded and clustered in the current period. Algorithm 4.1 and Figure 4.1 show the main flow of our proposed algorithm.

Descriptions of parameters of DBPCES algorithm are presented below:

• **re-initialization period (rp):** DBSCAN execution period in terms of number of data points.

Algorithm 4.1 $DBPCES(datastream, is, rp, ws)$

loop

$embedding \leftarrow \{\}$

% 1: Initialization

Get $initial\ data$

$initial\_embedding \leftarrow$ Generate UMAP model and embed initial data

$embedding \leftarrow embedding \cup initial\_embedding$

% 2: Parameter Value Selection

$eps, minPts \leftarrow findDBSCANParameters(inital\_embedding)$

% 3: Embedding

while not re-initialization period reached **do**

Get $embedding\ data$

$window\_embedding \leftarrow$ embed $embedding\ data$ with generated UMAP model

$embedding \leftarrow embedding \cup window\_embedding$

end while

% 4: Clustering

Cluster $embedding$ with DBSCAN using eps and $minPts$

% 5: Label Matching

if not first run **then**

Match labels of current period with previous period

end if

end loop

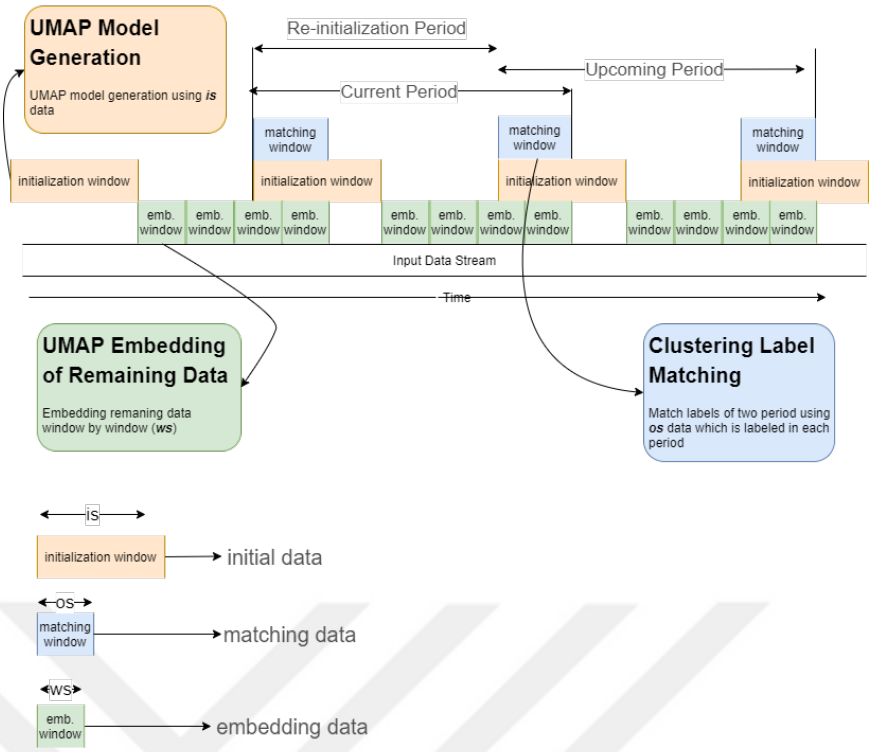


Figure 4.1: Streaming UMAP

• **initialization size (is):** Number of data points used for UMAP model generation. The algorithm generates UMAP model using is data at the beginning of each period in order to adapt itself to concept drift. is points at the beginning of each period is referred as $initial\ data$.

• **window size (ws):** Number of data points processed together. In each period, after UMAP model is generated using $initial\ data$, data is embedded window by window using this model. Each ws points after $initial\ data$ in a period is referred as $embedding\ data$.

• **overlap size (os):** Number of data points used in the current period from the previous period. Labels of consecutive periods are matched using this $matching\ data$. It can be used as half of the is. The os points at the end of a period is referred as $matching\ data$.

Each period of our algorithm can be divided into 5 phases: Phase I: Initialization, Phase II: Parameter Value Selection, Phase III: Embedding, Phase IV: Clustering and Phase V: Label Matching. Details of each phase are given in the following sections.

4.1 Phase I: Initialization

The first phase of the algorithm is initialization. Initialization phase runs periodically to adapt concept drift. is of data ($initial\ data$) in each period is used for UMAP model generation. The data space is learned using $initial\ data$ and the new data space in two-dimensions is generated. UMAP model is generated using only $initial\ data$ in each period. The remaining data has no effect on UMAP model. The remaining data is embedded using this generated UMAP model as described in Section 4.3.

4.2 Phase II: Parameter Value Selection

In the second phase, values of parameters of DBSCAN are estimated using a heuristic. In each period, embedded $initial\ data$ is considered as the representative of whole data in that period. Therefore, DBSCAN parameter values ($minPoints$ and ϵ) are estimated using the embedded $initial\ data$.

Our DBSCAN parameter estimation algorithm adapts a heuristic to predict the values of both parameters. Our parameter value estimation algorithm is a modified version of binary search algorithm. It selects $minPoints$ parameter using binary search algorithm and selects one of the partitions by comparing silhouette scores. Our parameter value estimation algorithm assumes that silhouette scores increase (or decrease) linearly. The algorithm discards the partition with lower silhouette score because that partition can not include greater silhouette score because of previous assumption.

The first parameter of DBSCAN is $minPoints$. Assume that there are N points in a data stream. A cluster can consist of 1 to N points. The minimum number of points to form a cluster ($minPoints$) can take a value between 1 to N. In our heuristic, we repeatedly divide search interval of $minPoints$ (initially [1,N]) into two parts and use the middle of each part as candidate $minPoints$.

As suggested by DBSCAN algorithm [8] and described in Section 2.1.1, we predict ϵ parameter for each candidate $minPoints$ using k-dist graph. For ϵ value estimation, the algorithm calculates each points' distance to k^{th} nearest neighbor, sorts them and plots sorted k-dist graph where k is candidate $minPoints$. The knee in the graph is

the candidate ϵ value. Algorithm 4.2 shows how our estimation algorithm calculates the candidate ϵ value for the corresponding candidate $minPoints$.

Algorithm 4.2 $findEpsilon(embedding, k)$

distances $\leftarrow \{\}$
for each $point\ p \in embedding$ **do**
  knn $\leftarrow k^{th}$ nearest neighbor of point p
  dist $\leftarrow$ dist(p, knn)
  distances $\leftarrow$ distances $\cup$ dist
end for
Sort distances in ascending order
Plot sorted distances
$\epsilon \leftarrow$ knee in sorted distances graph
return ϵ

After selecting the candidate ϵ and $minPoints$ value pairs, the algorithm clusters embedded data using DBSCAN with each pairs of parameter values and calculates the silhouette scores of both clusterings to discard partition with lower silhouette score. The algorithm divides the search interval of $minPoints$ in half by choosing a part with a greater silhouette score and repeats the same process until there is no interval to divide. At the end, it selects $(\epsilon, minPoints)$ pair with the maximum silhouette score. Algorithm 4.3 shows the main flow of our proposed DBSCAN parameter estimation algorithm.

Figure 4.2 shows an example flow of parameter value selection algorithm for a dataset with 10 data points. The search interval of $minPoints$ is [1-10] for this dataset. It is divided into two parts and middle of each part (3 and 8) is selected as candidate $minPoints$ ($minPoints1,\ minPoints2$). Then candidate ϵ values ($eps1\ and\ eps2$) is calculated using k-dist graph and the dataset is clustered using both set of parameters: ($minPoints1,\ eps1$) and ($minPoints2,\ eps2$). Then silhouette scores of each clustering is calculated: $sil1$ and $sil2$. The part with a smaller silhouette score is eliminated ($sil2$) and estimation continuous from the part with a greater silhouette score.

Algorithm 4.3 $findDBSCANParameters(embedding, start\_index, end\_index)$

if Nothing to divide **then**

return $(eps, minPoints)$ pair with maximum silhouette

else

Divide interval $[start\_index, end\_index]$ into two parts

minPoints1 $\leftarrow$ middle of first part

minPoints2 $\leftarrow$ middle of second part

eps1 $\leftarrow$ $findEpsilon$(embedding, minPoints1)

eps2 $\leftarrow$ $findEpsilon$(embedding, minPoints2)

Run DBSCAN1 with eps1 and minPoints1

Run DBSCAN2 with eps2 and minPoints2

silScore1 $\leftarrow$ silhouette score of DBSCAN1

silScore2 $\leftarrow$ silhouette score of DBSCAN2

if silScore1 >= silScore2 **then**

return $findDBSCANParameters$(embedding, start_index, (end_index - start_index) / 2 + start_index)

else

return $findDBSCANParameters$(embedding, (end_index - start_index) / 2 + start_index, end_index)

end if

end if

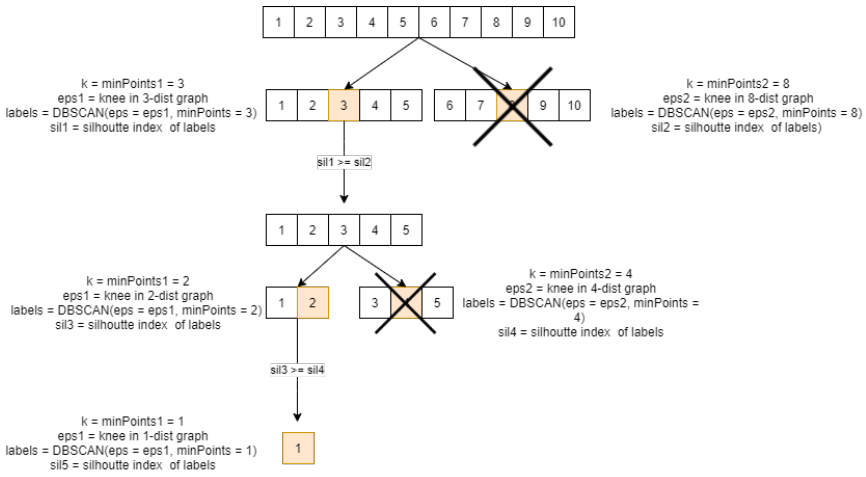


Figure 4.2: Parameter Value Selection

4.3 Phase III: Embedding

In this phase, data is embedded window by window using UMAP model generated in the initialization part. User-defined ws parameter defines window size in the Algorithm 4.1. Embedding continues until the re-initialization period (rp) is reached.

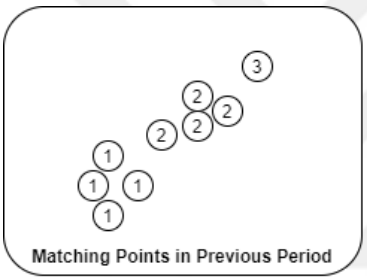
4.4 Phase IV: Clustering

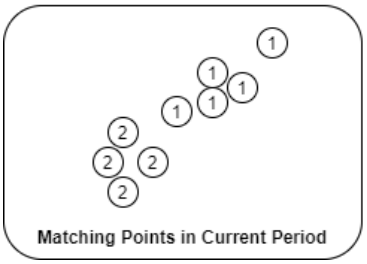
This is the part of the algorithm which generates clusters. The input of the DBSCAN is the embedding of the data in the current period. Since data is two-dimensional, DBSCAN clusters it easily. Although DBSCAN has two important parameters ϵ and $minPoints$, our algorithm does not require any user-defined parameter because our parameter selection algorithm described in Section 4.2 estimates values of both parameters of DBSCAN.

4.5 Phase V: Label Matching

Our proposed algorithm 4.1 embeds and clusters data in each period separately, therefore there is no coherence between labels of each period. In order to match labels of current period with previous period, os points (matching points) from the end of the

previous period is also embedded and clustered in the current period. These matching points have labels in the previous and current periods. We calculate overlapping element count between previous and current labels of matching points. We match current labels with one of the previous labels that have the maximum number of overlapping elements. Figure 4.3 shows an example of our matching algorithm. In this example, matching points have two labels in the current period and three labels in the previous period. Rows in the label match table correspond to labels in the current period and columns correspond to labels in the previous period. "Label 1" in the current period has 4 overlapping elements with "Label 2" and 1 overlapping element with "Label 3" in the previous period. Therefore, our algorithm changes labels of points with "Label 1" in the current period with "Label 2". "Label 2" in the current period has 4 overlapping elements with "Label 1" in the previous period. Therefore, our algorithm changes labels of points with "Label 2" in the current period with "Label 1".





		Labels in Previous Period		
		1	2	3
Labels in Current Period	1	0	4	1
	2	4	0	0

Figure 4.3: Label matching

CHAPTER 5

EMPIRICAL STUDY AND RESULTS

5.1 Data Streams

We have conducted our experiments on both synthetic and real data streams. All data streams are available at `https://gitlab.com/alaettinzubaroglu/emcstream/` [59].

5.1.1 Synthetic Data Streams

Synthetic data streams are generated using $DSD\_RandomRBFGeneratorEvents$ function of $streamMOA$ [60] package of R [61]. All of the synthetic streams except for $Stream\ 8$ evolve over time (contains concept drift). Each of them contains 50,000 data points in total. They are generated to observe the effect of dimension, the number of clusters, and speed of concept drift. Therefore, they differ in terms of dimension, the number of clusters, and speed of concept drift. The properties of each synthetic stream are given in Table 5.1.

5.1.2 Real Data Streams

Meteorological Data: We conducted experiments on two different real meteorological data streams. Each meteorological data stream contains 87,650 data points of 6 features: temperature, precipitation, snowfall, snow mass, air density, and cloud cover. The weight of temperature and air density features are increased by doubling these two features after normalizing whole data because these two features are the most important ones in all cases to distinguish cities. The first data stream contains

Table 5.1: Properties of synthetic streams.

	Number of Clusters	**Dimension Count**	**Concept Drift**
Stream 1	10	50	medium speed
Stream 2	10	100	medium speed
Stream 3	10	10	medium speed
Stream 4	40	50	medium speed
Stream 5	4	50	medium speed
Stream 6	10	50	high speed
Stream 7	10	50	low speed
Stream 8	10	50	no drift

weather data of two cities in Turkey while the second one contains weather data of two cities in the United States. Both contain hourly measurements of 5 years from the beginning of 2015 till the end of 2019. Cities in both data streams are selected such that they have different climate characteristics. Both data streams contain concept drift by nature because of changes in weather conditions from daytime to night, from season to season, and in a year.

Keystroke Data: Keystroke data contains typing characteristics of users of ".tie5Roanl" password. The password is typed 50 times in eight different sessions (400 times for each user). Each data point is formed of 31 features. Data streams contain concept drift because of changes in users' practice. Original data streams contain typing characteristics of 51 users while $keystroke-2$, $keystroke-3$, and $keystroke-4$ are subsets generated from original data streams and contain typing characteristics of 2, 3, 4 users respectively.

Table 5.2 shows properties of real data streams that are used for the evaluation of DBPCES algorithm.

Table 5.2: Properties of real streams.

	Number of Data Points	Number of Clusters	Dimension Count
meteo-tr	87,650	2	6
meteo-us	87,650	2	6
keystroke-2	800	2	31
keystroke-3	1,200	3	31
keystroke-4	1,600	4	31

5.2 Experimental Setup

5.2.1 Environment

For the experiments, a computer with an Intel Core i5-6200U CPU 2.40 GHz processor and 12 GB memory and Windows 10 20H2 operating system was used. The proposed method was implemented with Python version 3.6. The implementation of DBPCES is available at `https://github.com/ozlempoyraz/DBPCES` [62]. For the sake of fairness, we also implemented DenStream algorithm with the same version of Python.

5.2.2 Parameters

Common parameters of DBPCES, implementation of Zubaroğlu and Atalay, and DenStream (rp, is, and ws) were selected intuitively. All algorithms were run several times using different values of common parameters during each execution. The parameter values that yield the best results for most of the algorithms were selected for the final configuration. For all synthetic streams and meteorological data, rp was chosen as one-tenth of the number of all data points. In other words, clustering was run 10 times for synthetic and meteorological data. The number of data points used for initialization (is) was selected as one-fifth of the number of data points in each period. Additionally, the number of data points processed together (ws) was selected as one-tenth of is. The number of data points in keystroke data is much smaller compared to other streams. Therefore, rp was chosen as one-quarter of the number of data

Table 5.3: Common parameters of all algorithms for all data streams.

	number of data points	reinit period (rp)	init size (is)	window size (ws)
Synthetic Streams	50,000	5,000	1,000	100
Meteorological data	87,650	8,765	1,590	175
keystroke-2	800	200	80	40
keystroke-3	1,200	300	180	60
keystroke-4	1,600	400	240	80

points for keystroke data. Unlike other data streams, is was selected proportionally more for keystroke data because the performance of DBPCES was affected negatively from small is because of the parameter estimation algorithm. is was selected as 80, 180, 240 data points and ws was selected as 40, 60, 80 data points for $keystroke\ 2$, $keystroke\ 3$ and $keystroke\ 4$ data streams respectively. The performance of DenStream is very dependent on the user-defined ϵ parameter. Therefore, ϵ was selected using grid-search with the intervals of 0.05 for each data stream to be able to compare its performance with other algorithms fairly. The results given here are with ϵ value that makes the performance of DenStream the best.

The overlap size (os) parameter is valid only for implementation of Zubaroğlu and Atalay, and DBPCES and it was used as half of is for all data streams. Table 5.3 shows common parameters of all algorithms.

5.3 Results

We run three different clustering algorithms on the same data streams in our experiments to evaluate their performance: Our proposed algorithm DBPCES, implementation of Zubaroğlu and Atalay [34] which is the baseline for our algorithm, DenStream [39] which is a popular stream clustering algorithm.

The results of all algorithms are compared using total purity and total adjusted rand index values. Since both DBPCES and implementation of Zubaroğlu and Atalay use label matching algorithm, the predicted labels are preserved between periods. $matching\ data$ at the end of the previous period is also clustered in the current period. In this way, labels of data points in the current period are replaced with their

matches in the previous period. Therefore, the total predicted labels can be obtained by concatenating labels of periods. Total purity and total adjusted rand index are calculated using total predicted labels. Total predicted labels are also used to visualize the total clustering result of DBPCES algorithm. Additionally, we compared their performances using purity, ARI, and silhouette score of each period separately for fair comparison because DenStream does not match the labels of periods. Although DenStream does not have label matching between periods and results of entire clustering are not presented in the original paper [39], total purity and total ARI values are also obtained using the same method with other algorithms in this study to show the absence of matching.

Additionally, we develop DBPCES without UMAP in order to observe the effects of dimensionality reduction before clustering. DBPCES without UMAP also clusters data in each period separately and uses the same algorithm to match the labels of periods. As in DBPCES, it uses $initial\ data$ in each period to estimate parameter values of DBSCAN. DBPCES without UMAP stores data in the current period in memory as is until the upcoming period while DBPCES stores embedded data in the current period. Therefore, DBPCES without UMAP requires more memory than DBPCES.

5.3.1 Evaluation on Synthetic Data Streams

Table 5.4 shows total purity values and Table 5.5 shows total adjusted rand index values of the entire clustering for each synthetic data stream. Performance results of DBPCES algorithm and implementation of Zubaroğlu and Atalay in terms of total purity and total ARI are similar and close to 1 (good clustering in terms of both metrics) for all synthetic data streams.

Total purity and total ARI values of DBPCES algorithm and implementation of Zubaroğlu and Atalay on $Stream\ 6$ are slightly less than 1. The main difference of $Stream\ 6$ from other data streams is its high concept drift speed. As mentioned earlier, we run UMAP model generation and parameter estimation phases in every 5,000 points for synthetic streams (rp was selected intuitively as 5,000 for all synthetic streams). In order to test whether the initialization phase should be run more frequently for

Table 5.4: Total purity results on synthetic data streams.

	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DenStream**
Stream 1	1.0	1.0	0.380
Stream 2	0.917	1.0	0.349
Stream 3	0.999	0.999	0.322
Stream 4	0.961	1.0	0.254
Stream 5	1.0	1.0	0.598
Stream 6	0.919	0.979	0.344
Stream 7	1.0	1.0	0.871
Stream 8	1.0	1.0	1.0

$Stream\ 6$ because of its higher concept drift speed, we reduced all common parameters of algorithms to half of the initial configuration given in Table 5.3. The total purity and total ARI results for both configurations are presented in Table 5.6. The results of both implementation of Zubaroğlu and Atalay, and DBPCES are better with the second configuration as presented in Table 5.6. It can be concluded that UMAP model generation and parameter estimation algorithm should be run frequently (rp should be smaller) for data streams that evolve quickly.

The performance of DBPCES on $Stream\ 2$ is not as good as on other data streams in terms of total purity and total ARI. $Stream\ 2$ has more dimensions than other data streams. DBPCES embeds data streams into two dimensions using UMAP and estimates DBSCAN parameters using embedded data points. UMAP tries to preserve both local and global structures. However, some information may be lost while embedding data points with high dimensions into two dimensions and parameter estimation of DBPCES algorithm may be affected negatively by this loss.

Unlike implementation of Zubaroğlu and Atalay and DBPCES, the performance of DenStream is very poor in terms of total purity and total adjusted rand index for all data streams except for $Stream\ 8$ which is the only stationary data stream. We noticed that there is no correspondence between labels of periods of clustering. As mentioned earlier, DenStream uses the micro-cluster concept for summarizing data

Table 5.5: Total adjusted rand index (ARI) results on synthetic data streams.

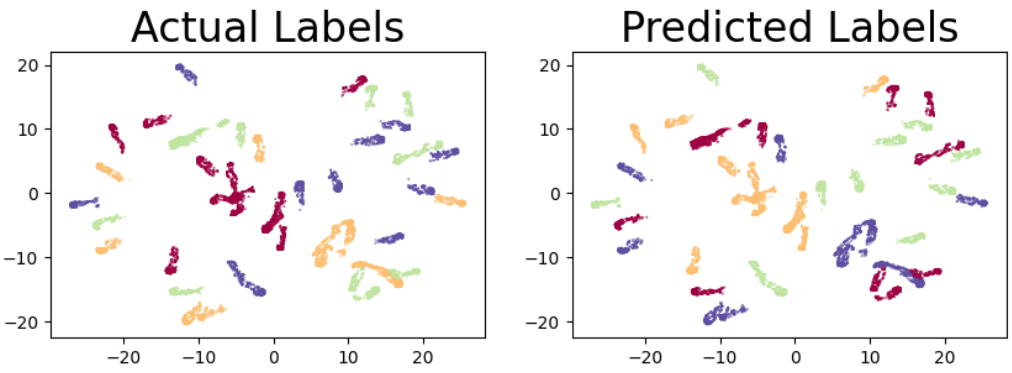
	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DenStream**
Stream 1	0.977	1.0	0.150
Stream 2	0.895	1.0	0.161
Stream 3	0.999	0.999	0.098
Stream 4	0.917	1.0	0.125
Stream 5	1.0	1.0	0.204
Stream 6	0.881	0.959	0.110
Stream 7	1.0	1.0	0.711
Stream 8	1.0	1.0	1.0

Table 5.6: Total Purity and total ARI results on $Stream\ 6$ with different common parameters.

	rp	**DBPCES**	**Implementation of Zubaroğlu and Atalay**
Purity	**5,000**	0.919	0.979
	2,500	0.995	1.0
ARI	**5,000**	0.881	0.959
	2,500	0.980	1.0

streams in the online phase. At certain periods or with a user action, DBSCAN algorithm is run on potential micro-clusters and core micro-clusters are generated. These core micro-clusters can be labeled differently in the upcoming period or micro-clusters can be eliminated according to their weight and core micro-clusters can change. Therefore, the labels of the core micro-clusters can change every clustering period. This makes concatenating labels of each period unreasonable for DenStream. In other words, calculating purity and adjusted rand index correctly for the whole stream is not possible for DenStream. This is one of the most important drawbacks of DenStream. The reason for high total purity and high total ARI values of DenStream on $Stream\ 8$ is that micro-clusters are not eliminated until at the end of the stream since $Stream\ 8$ does not contain concept drift. Both DBPCES and implementation of Zubaroğlu and Atalay match the labels of periods. Therefore, it is possible to observe the entire clustering results of both DBPCES and implementation of Zubaroğlu and Atalay. In addition, both algorithms embed data points into two dimensions and clusters them. This makes visualization of clustering results possible. As an example, visualization of the total clustering result of DBPCES for $Stream\ 5$ with actual clusters is given in Figure 5.1. Points in the figure correspond embedding of multi-dimensional data points into two dimensions. The colors of the points represent actual labels in Figure 5.1a and predicted labels in Figure 5.1b. Figure 5.1 shows that most of the points that are drawn with the same color (has same actual label) in Figure 5.1a are also drawn with the same color (has same predicted label) in Figure 5.1b. As an example, the data points that are colored with bordeaux in Figure 5.1a are colored with orange in Figure 5.1b. In other words, labels of most of the points are predicted correctly.

As mentioned previously, there is no correspondence between cluster labels of each period of DenStream. Therefore, we calculated and presented the results of each period separately to be able to fairly compare the performance of DenStream with the other two algorithms. Table 5.7 shows the minimum, maximum, and average silhouette score statistics of periods of DBPCES, implementation of Zubaroğlu and Atalay, and DenStream. In addition for each period, the purity statistics are presented in Table 5.8 and the adjusted rand index statistics are given in Table 5.9. The algorithms perform similarly for most of the data streams.



(a) (b)

Figure 5.1: DBPCES total clustering result for Stream 5.

Table 5.7: Min-max-average silhouette score of each period on synthetic data streams.

	MIN			MAX			AVERAGE		
	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**
Stream 1	0.895	0.891	0.618	0.924	0.915	0.924	0.907	0.903	0.822
Stream 2	0.599	0.865	0.815	0.892	0.904	0.953	0.694	0.882	0.913
Stream 3	0.876	0.856	0.616	0.919	0.919	0.741	0.900	0.896	0.699
Stream 4	0.667	0.939	0.758	0.951	0.951	0.923	0.850	0.946	0.861
Stream 5	0.853	0.853	0.540	0.931	0.931	0.944	0.900	0.900	0.832
Stream 6	0.600	0.621	0.556	0.859	0.894	0.762	0.729	0.826	0.674
Stream 7	0.931	0.933	0.973	0.945	0.945	0.979	0.938	0.938	0.975
Stream 8	0.883	0.883	0.992	0.908	0.908	0.992	0.895	0.895	0.992

As in the total results, implementation of Zubaroğlu and Atalay and DBPES on $Stream\ 6$ are not as successful as on other streams according to results of each period as expected. Similarly, the performance of DenStream algorithm on $Stream\ 6$ is the worst of the three algorithms in terms of silhouette score and ARI. $Stream\ 6$ differs from $Stream\ 7$ in only concept drift speed. The concept drift speed of $Stream\ 6$ is the highest. To enhance the performance of DCPCES and implementation of Zubaroğlu and Atalay on data streams that have a high concept drift speed, we recommend changing the configuration such that the algorithm runs initialization more frequently (lower rp). The performance of DenStream can be enhanced on data streams that have a high concept drift speed by configuring λ parameter which controls the importance of older points to clustering.

DenStream is very successful in terms of purity for almost all data streams according to results in Table 5.8. However, the success of DenStream in terms of purity is not

Table 5.8: Min-max-average purity of each period on synthetic data streams.

	MIN			MAX			AVERAGE		
	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**
Stream 1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Stream 2	0.898	0.993	1.0	1.0	1.0	1.0	0.921	0.998	1.0
Stream 3	0.999	0.993	0.970	1.0	1.0	1.0	0.999	0.999	0.992
Stream 4	0.880	1.0	0.980	1.0	1.0	1.0	0.964	1.0	0.995
Stream 5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Stream 6	0.809	0.896	0.962	1.0	1.0	1.0	0.921	0.977	0.991
Stream 7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Stream 8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 5.9: Min-max-average ARI results of each period on synthetic data streams.

	MIN			MAX			AVERAGE		
	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**
Stream 1	0.990	1.0	0.927	1.0	1.0	1.0	0.999	1.0	0.959
Stream 2	0.889	0.986	0.965	1.0	1.0	1.0	0.914	0.995	0.993
Stream 3	0.998	0.985	0.825	1.0	1.0	0.981	0.999	0.998	0.910
Stream 4	0.811	1.0	0.930	1.0	1.0	1.0	0.948	1.0	0.973
Stream 5	1.0	1.0	0.840	1.0	1.0	1.0	1.0	1.0	0.949
Stream 6	0.812	0.889	0.769	1.0	1.0	0.941	0.919	0.974	0.874
Stream 7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Stream 8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

consistent with other evaluation metrics. As an example, the purity of DenStream algorithm on $Stream\ 6$ is very close to 1 which means perfect clustering. However, the performance of DenStream in terms of ARI and silhouette score is not as good as in terms of purity. As mentioned earlier, purity is high if the clusters contain data points from the same class. According to this definition, the purity is also high if data points in the same class are divided into multiple clusters. Therefore, the success of Denstream in terms of only purity metric means that DenStream clusters data points into more clusters than actual classes.

We present the execution time of each period (for rp points) for DenStream, DBPCES, and implementation of Zubaroğlu and Atalay in Table 5.10. DenStream is the fastest algorithm among them. Speeds of DBPCES and implementation of Zubaroğlu and Atalay are very close. Neither dimension count nor concept drift speed affects the speed of any of the algorithms. However, DenStream clusters $Stream\ 4$, which has more clusters than others, almost twice slower than other streams.

As mentioned before, DBPCES does not require any user-defined parameter for the clustering part while DenStream requires density threshold parameters and imple-

Table 5.10: Min-max-average execution time (second) of each period on synthetic data streams.

	MIN			MAX			AVERAGE		
	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM
Stream 1	7.29	6.61	1.40	17.32	24.41	5.43	8.46	8.92	1.97
Stream 2	7.31	7.24	1.40	17.93	18.37	5.35	8.43	9.55	1.95
Stream 3	7.08	6.67	1.59	17.34	19.27	5.73	8.21	8.79	2.42
Stream 4	5.94	6.30	4.70	16.51	20.11	9.05	7.53	8.20	5.55
Stream 5	7.54	6.98	0.89	17.78	25.19	5.30	8.70	9.36	1.42
Stream 6	7.26	6.90	1.70	17.59	22.60	6.32	8.34	9.47	2.42
Stream 7	7.01	6.25	1.38	17.37	16.63	6.06	8.15	7.39	1.96
Stream 8	5.10	4.28	1.40	15.61	14.56	6.12	6.22	5.38	1.95

mentation of Zubaroğlu and Atalay requires the number of clusters. Therefore, the number of clusters predicted by DBPCES is not always the same as the actual number of clusters. Figure 5.2 shows the actual and predicted number of clusters for DBPCES algorithm for each synthetic stream. The number of clusters is predicted very close to the actual number of clusters for all of the streams by DBPCES. Comparing this result with the other two algorithms is neither reasonable nor fair. Because the number of clusters is a user-defined parameter in implementation of Zubaroğlu and Atalay and DenStream does not match the labels of periods.

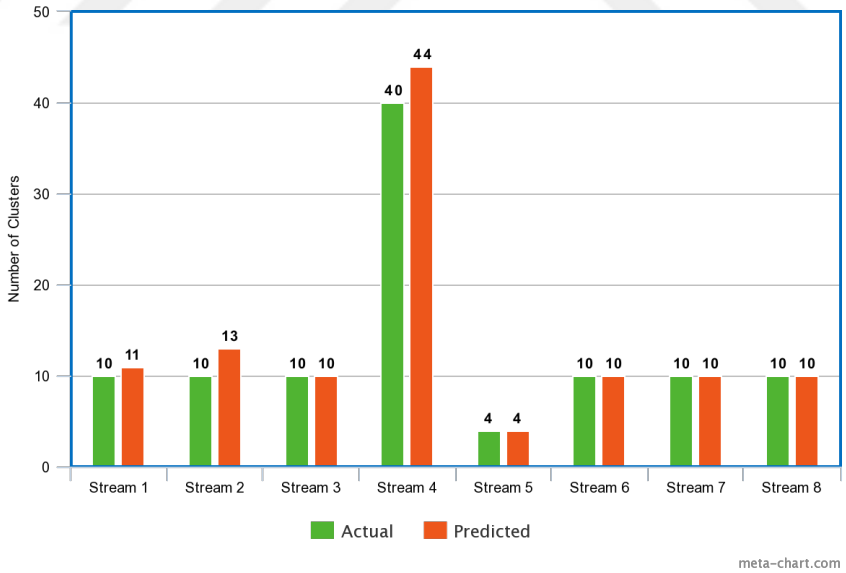


Figure 5.2: DBPCES actual-predicted number of clusters for synthetic data streams.

Table 5.11 presents total purity, total ARI and total execution time result of DBPCES and DBPCES without UMAP for all synthetic data streams. For all synthetic streams

Table 5.11: DBPCES and DBPCES without UMAP comparison on synthetic streams

	Total Purity		Total ARI		Total Execution Time (second)	
	DBPCES	**DBPCES without UMAP**	**DBPCES**	**DBPCES without UMAP**	**DBPCES**	**DBPCES without UMAP**
Stream 1	1.0	0.999	0.977	0.999	84.6	23.14
Stream 2	0.917	0.999	0.895	0.994	84.3	21.09
Stream 3	0.999	0.999	0.999	0.999	82.1	17.83
Stream 4	0.961	0.070	0.917	0.001	75.3	24.76
Stream 5	1.0	0.999	1.0	0.999	87.0	22.82
Stream 6	0.919	0.859	0.881	0.715	83.4	22.82
Stream 7	1.0	0.999	1.0	0.999	81.5	22.36
Stream 8	1.0	0.999	1.0	0.999	62.2	23.20

except for $Stream\ 4$, both algorithm gives similar results in terms of total purity and total ARI. DBPCES is much more successful on $Stream\ 4$ than DBPCES without UMAP with respect to total purity and total ARI. The difference of $Stream\ 4$ from other synthetic streams is the number of clusters. $Stream\ 4$ has the most clusters. As a result of curse of dimensionality, euclidean distances between points and clusters become equidistant as dimensionality increases. This makes differentiating clusters harder. Differentiating clusters becomes even harder as the number of clusters increases together with the number of clusters. This is the reason for the poor results of DBPCES without UMAP on $Stream\ 4$. Embedding the data before clustering improves cluster quality on $Stream\ 4$ as the results of DBPCES show. The execution time of DBPCES without UMAP is very low compared to DBPCES although DBPCES is very successful on all synthetic streams.

5.3.2 Evaluation on Real Data Streams

The performance of DBPCES on real data streams is compared with that of implementation of Zubaroğlu and Atalay and DenStream. Table 5.12 presents total purity results for each algorithm although it is not reasonable for DenStream. All algorithms for all data streams give similar results in terms of total purity. All total purity results are very close to 1 which means all clusters contain data points from a single class. The total purity results of DBPCES and implementation of Zubaroğlu and Atalay are

Table 5.12: Total purity results on real data streams

	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DenStream**
meteo-tr	1.0	1.0	1.0
meteo-us	1.0	1.0	1.0
keystroke-2	1.0	1.0	1.0
keystroke-3	0.982	0.982	0.954
keystroke-4	0.954	0.955	0.933

consistent with our expectation. However, total purity results of DenStream are also very close to 1 unlike our expectations although it does not match labels of periods. It can be concluded that potential micro clusters are not eliminated until the end of data streams and somehow core micro clusters are labeled with the same number in each period.

As mentioned before, DBPCES and DenStream do not require the number of clusters as input. Both of them give high purity for all data streams. To be sure whether the reason for high purity is splitting a single class into more clusters or not, total ARI results of all algorithms for all real data streams are also presented in Table 5.13. Total ARI values are also equal to 1 for all algorithms for meteorological data streams and $keystroke-2$. This is surprising for DenStream that does not match the labels of periods. This means that somehow most of the potential micro clusters remain until the end of the data stream because of the way concept drifts and core micro clusters are labeled the same in each period. The performance of DBPCES and implementation of Zubaroğlu and Atalay on $keystroke-3$ and $keystroke-4$ in terms of total ARI is similar and good. However, DenStream is not as good as the others on these data streams as expected. Total ARI results of DBPCES show that the reason for high purity is not more clusters than actual classes. DBPCES clusters all real streams as successful as implementation of Zubaroğlu and Atalay.

We present minimum, maximum and average silhouette score, purity, and ARI results of periods in Table 5.14, Table 5.15 and Table 5.16 respectively. The silhouette scores of each algorithm are similar and not much high for $meteo-tr$ and $meteo-us$

Table 5.13: Total adjusted rand index (ARI) results on real data streams.

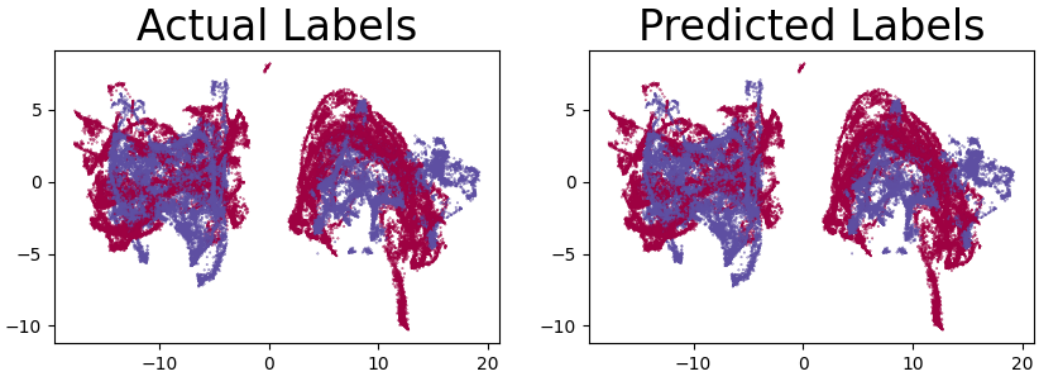
	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DenStream**
meteo-tr	0.982	1.0	1.0
meteo-us	1.0	1.0	1.0
keystroke-2	0.995	1.0	1.0
keystroke-3	0.948	0.948	0.869
keystroke-4	0.883	0.884	0.698

Table 5.14: Min-max-average silhouette score results of each period on real data streams.

	MIN			**MAX**			**AVERAGE**		
	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**
meteo-tr	0.512	0.656	0.583	0.776	0.776	0.680	0.698	0.714	0.619
meteo-us	0.627	0.627	0.681	0.737	0.737	0.765	0.683	0.683	0.713
keystroke-2	0.519	0.920	0.423	0.944	0.944	0.703	0.826	0.928	0.603
keystroke-3	0.812	0.812	0.286	0.921	0.921	0.501	0.877	0.877	0.414
keystroke-4	0.678	0.678	0.250	0.882	0.906	0.433	0.798	0.833	0.353

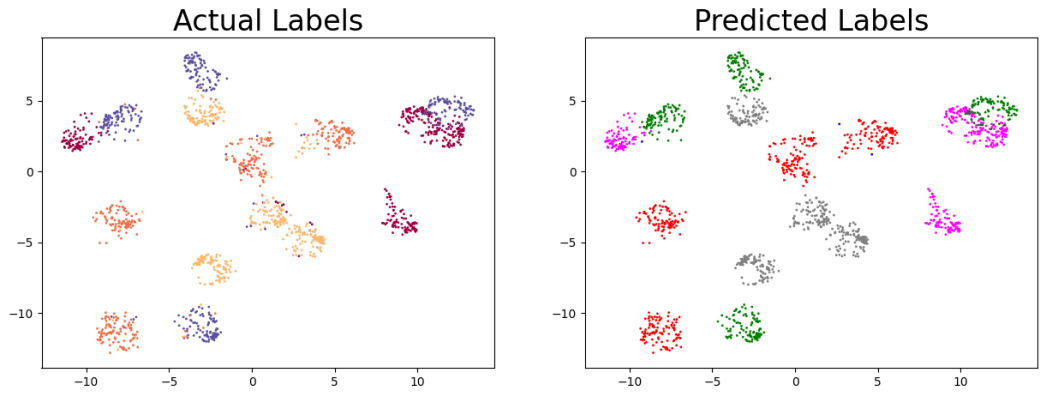
data streams. Figure 5.3 shows the total clustering result of DBPCES algorithm as an example. Actual clusters are not well-separated as can be seen from the Figure 5.3a. This is the cause of the low silhouette score. Additionally, silhouette scores of DBPCES and implementation of Zubaroğlu and Atalay are higher than DenStream algorithm for keystroke data streams. The total clustering result of DBPCES algorithm for $keystroke - 4$ data stream is presented in Figure 5.4 as an example. Clusters in keystroke data streams are more separated than meteorological data streams. Therefore, silhouette scores of keystroke data streams are higher than meteorological data streams.

For all real data streams, purity and ARI of periods of all algorithms are compatible with each other and very high as presented in Table 5.15 and Table 5.16. DBPCES algorithm clustered all real data streams successfully without any user-defined clustering parameter. Consistency of purity with ARI shows that DBPCES predicts DBSCAN parameters successfully and finds an almost equal number of clusters with an actual number of classes. Figure 5.5 presents the comparison of the actual number



(a) (b)

Figure 5.3: DBPCES total clustering result for meteo-us.



(a) (b)

Figure 5.4: DBPCES total clustering result for keystroke-4.

Table 5.15: Min-max-average purity results of each period on real data streams.

	MIN			MAX			AVERAGE		
	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM
meteo-tr	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
meteo-us	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
keystroke-2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
keystroke-3	0.971	0.971	0.933	0.996	0.996	0.983	0.984	0.984	0.954
keystroke-4	0.925	0.925	0.917	0.975	0.975	0.965	0.957	0.957	0.937

Table 5.16: Min-max-average ARI results of each period on real data streams.

	MIN			MAX			AVERAGE		
	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM	DBPCES	Implementation of Zubaroğlu and Atalay	DENSTREAM
meteo-tr	0.862	1.0	1.0	1.0	1.0	1.0	0.986	1.0	1.0
meteo-us	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
keystroke-2	0.983	1.0	1.0	1.0	1.0	1.0	0.995	1.0	1.0
keystroke-3	0.918	0.918	0.814	0.989	0.989	0.950	0.954	0.954	0.871
keystroke-4	0.813	0.813	0.710	0.933	0.933	0.844	0.889	0.890	0.760

of clusters and predicted number of clusters by DBPCES. DBPCES finds one extra cluster for $keystroke-4$, $keystroke-2$, and $meteo-tr$ data streams. Figure 5.4, Figure 5.6, and Figure 5.7 visualize embedding of data points in $keystroke-4$, $keystroke-2$, and $meteo-tr$ data streams respectively. The colors at the left graph of all figures correspond to actual labels while the colors at the right graph correspond to predicted labels. The number of points that are colored with blue in Figure 5.4b and Figure 5.6b is very small compared to the number of points that are drawn with other colors. Similarly, the number of points that are colored with purple is also very small in Figure 5.7b. Even though, DBPCES finds one extra cluster for these data streams, there are only a few points in these extra clusters.

Unlike the high performance of DBPCES in terms of all evaluation metrics, DBPCES is much slower than DenStream for all of the real streams. Table 5.17 shows execution time statistics of periods of each algorithm. The execution times of DBPCES and implementation of Zubaroğlu and Atalay on keystroke data streams are almost the same but DenStream is very fast compared to the other two. Close execution times for DBPCES and implementation of Zubaroğlu and Atalay show that the main cause of high execution time is not clustering algorithms that are used by DBPCES and implementation of Zubaroğlu and Atalay. The main cause is UMAP algorithm that is used for embedding data points into 2-dimensions. DenStream is also very fast

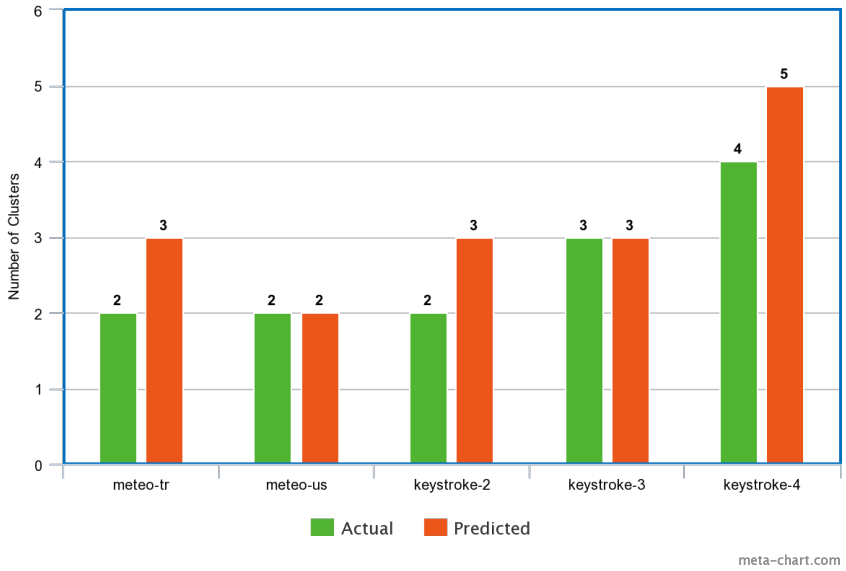


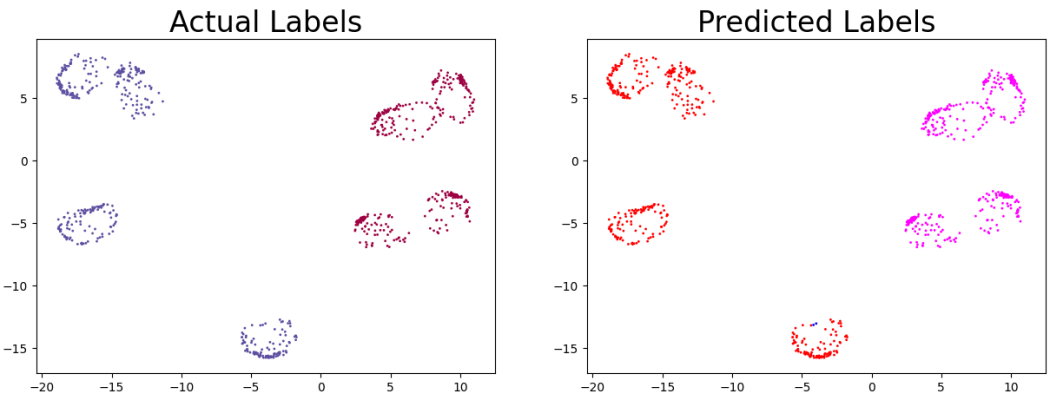
Figure 5.5: DBPCES actual-predicted number of clusters for real data streams.

Table 5.17: Min-max-average execution time (second) of each period on real data streams.

	MIN			MAX			AVERAGE		
	DBPCES	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**	**DBPCES**	**Implementation of Zubaroğlu and Atalay**	**DENSTREAM**
meteo-tr	9.26	6.84	0.95	20.04	17.76	12.68	10.85	8.36	2.18
meteo-us	9.18	6.84	0.98	20.20	17.36	12.11	10.51	8.05	2.12
keystroke-2	0.312	0.27	0.01	11.07	10.86	0.06	3.01	2.94	0.03
keystroke-3	0.578	0.51	0.03	11.07	11.18	0.2	3.21	3.19	0.08
keystroke-4	0.781	0.73	0.07	11.23	11.22	0.45	3.40	3.37	0.17

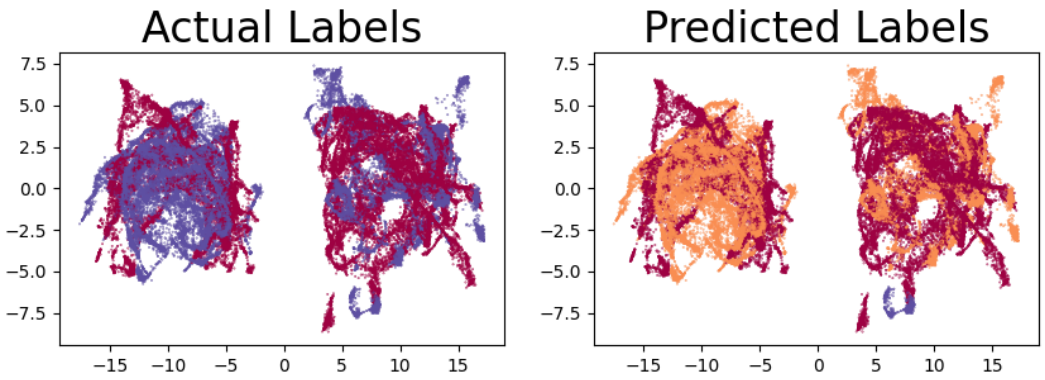
compared to the other two algorithms on meteorological data. The execution times of DBPCES on meteorological data streams are a few seconds longer than the execution times of Implementation of Alaettin and Atalay. The parameter selection algorithm used in DBPCES takes 2 seconds on average for meteorological data streams. The parameter selection algorithm is independent from embedding of *embedding data*. Therefore, it can be run in parallel with the embedding phase described in Section 4.3.

Table 5.18 presents total purity, total ARI and total execution time result of DBPCES and DBPCES without UMAP for all real data streams. Clustering performance of DBPCES without UMAP on keystroke data streams is very poor in terms of total purity and total ARI as clearly seen in Table 5.18. This is because of the first reason



(a) (b)

Figure 5.6: DBPCES total clustering result for keystoke-2.



(a) (b)

Figure 5.7: DBPCES total clustering result for meteo-tr.

Table 5.18: DBPCES and DBPCES without UMAP comparison on real streams

	Total Purity		Total ARI		Total Execution Time (second)	
	DBPCES	**DBPCES without UMAP**	**DBPCES**	**DBPCES without UMAP**	**DBPCES**	**DBPCES without UMAP**
meteo-tr	1.0	0.999	0.982	0.997	108.5	65.44
meteo-us	1.0	0.999	1.0	0.994	105.1	43.34
keystroke-2	1.0	0.55	0.995	0.009	12.04	0.24
keystroke-3	0.982	0.344	0.948	0.0003	12.84	0.50
keystroke-4	0.954	0.255	0.883	0	13.60	0.72

of curse of dimensionality problem: data sparsity. Both DBPCES and DBPCES with UMAP use $initial\ data$ in each period as representative of data points in that period and estimate parameters of DBSCAN using $initial\ data$. The number of data points in keystroke data streams is very small compared to other streams even though the number of dimensions of keystroke data streams is not small. As the dimension increases, the number of data points used for training should also increase in order to overcome the problem of curse of dimensionality because training data should contain different combinations of features as representative. This is the reason for the poor performance of DBPCES without UMAP on keystroke data streams while DBPCES performs very well. Similar to execution time results of synthetic, DBPCES without UMAP is much faster than DBPCES because of high execution time of UMAP.

5.4 Discussion

5.4.1 Selection of Appropriate Hyper-parameters for DBPCES

In case of the existence of prior knowledge about data streams, hyper-parameters of DBPCES can be selected in the light of guidelines described below.

• **rp:** If the speed of concept drift is high, it is better to run both initialization and parameter value selection phases frequently. These two phases are run to adapt to the change in the characteristics of data. In the case of high concept drift speed, data characteristics change faster. By selecting lower rp, UMAP model generation and DBSCAN parameter value selection are repeated more frequently. In this way,

DBPCES adapt itself to the high speed of concept drift.

• **is:** If the concept drift or the number of features is high, the algorithm performs better with higher is. DBPCES uses is data in each period as a representative of data points in that period. When the dimension is high, DBPCES needs more training data to learn all features because of data sparsity. Additionally, DBPCES needs more initial data to learn data characteristics if the concept drift is high.

• **ws:** This parameter does not affect the performance of the algorithm. It can be selected according to the speed of the stream.

5.4.2 Selection of Appropriate Algorithm

In the light of the results of experiments presented here, a guideline is suggested below to help to select appropriate algorithms according to need and data stream properties.

• **DenStream:** The execution time of DenStream is the smallest for most of the data streams. DenStream is the best choice if the speed of the stream is high, the data stream is high dimensional and prior knowledge is available to infer DenStream parameters because the execution time of algorithms that uses UMAP (DBPCES and implementation of Zubaroğlu and Atalay) is very high compared to DenStream and the performance DBPCES without UMAP is poor for high dimensional data streams. However, the entire clustering result can not be observed in DenStream because of the absence of a label matching algorithm between periods.

• **DBPCES:** The performance of DBPCES is very close to the other three algorithms in terms of all evaluation metrics although it does not require any data-dependent parameters. Although DBPCES is slower than DenStream, its speed is fast enough to cluster data streams such as meteorological data streams. DBPCES clusters approximately 800 meteorological data instances in a second as an example although meteorological data instances consist of hourly measurements. Although the speed of DBPCES is not as good as DenSteam, it can also be used for data streams that are not that fast.

• **DBPCES without UMAP:** Its performance is almost the same for most of the data streams in terms of all evaluation metrics and it is faster than DBPCES and implementation of Zubaroğlu and Atalay. DBPCES without UMAP also does not require data-dependent parameters. However, it performs poorly if the dimension is high and the training size is small. It becomes a good choice if the stream speed is high, data is not high dimensional and no prior information is available.

5.4.3 Scalability

The performance of DBPCES is not affected by the number of data points in the data stream if the speed of the stream is not too fast. If the speed of the data stream is faster than the processing speed of DBPCES, data cannot be clustered by DBPCES because DBPCES does not use a buffer to store data points that are not processed. Even if a buffer mechanism is used, the size of the buffer may increase in a short period of time according to the speed of the stream. In this case, periods of DBPCES can be run parallel because data points in each period are embedded and clustered independently. However, how the labels of periods would be matched become a problem in that case.

As mentioned previously, the performance of the algorithm is not affected by the number of dimensions. In order to observe the effect of the number of dimensions on the clustering, DBPCES is run artificial streams which differ in the number of dimensions. Change in the number of dimensions affects neither execution time nor the goodness of clustering of DBPCES algorithm.

5.4.4 Robustness

DBPCES assumes all the attributes of a data point are available. However, it is very common to have missing attributes in real time scenarios. There are several methods that try to predict value of missing attributes to make algorithms more robust. Assigning average or most common value of attributes to missing values is among the most popular methods. These methods can be also used for predicting missing values of data points in DBPCES. DBPCES can store the average of each attributes until that time and assign these averages to missing values of these attributes.

CHAPTER 6

CONCLUSION

With the increase of devices that generate data streams, data stream clustering becomes a more popular research area because of the valuable information that can be inferred from clusters. There are lots of data stream clustering algorithms in the literature. Most of them require user-defined parameters that can be extracted from whole data. A data stream continuously arrives and it can not be recorded because of storage constraints. Therefore, the extraction of these parameters from whole data is not possible.

In this study, we have developed a data stream clustering algorithm, Density-Based and Parameterless Clustering of Embedded Data Streams (DBPCES). DBPCES algorithm embeds data streams into two dimensions and clusters them using DBSCAN algorithm. DBPCES does not require any data-dependent parameter for clustering. Parameters of DBSCAN are estimated using a heuristic developed in this study. Data points are embedded using UMAP because it has the capability of embedding new data points using previously generated model. UMAP algorithm is adapted to concept drift in DBPCES. Additionally, DBSCAN parameters are estimated and updated periodically to adapt concept drift. DBPCES algorithm makes visualization of data points and also clusters possible because it embeds data points into two dimensions. The labels after processing of each period of DBPCES can be concatenated because labels of each period is matched using the matching algorithm of DBPCES. In this way, clusters that are generated from the whole stream can be visualized at the end of the stream. DBPCES requires 3 hyper-parameters: rp, is, and ws. rp corresponds to the period of clustering in terms of the number of data points while is defines the number of data points used in the UMAP model generation and DBSCAN parameter

estimation. After initial data (is data points at the beginning of each period), each ws data is embedded together using generated UMAP model.

We compared the performance of our algorithm with two different data stream clustering algorithms using both synthetic and real data streams. The first algorithm used for comparison is DenStream which is a state-of-the-art data stream clustering algorithm that requires two important user-defined parameters that are inferred by examining whole data. The second algorithm is implementation of Zubaroğlu and Atalay which is the basis of DBPCES. Unlike DBPCES, implementation of Zubaroğlu and Atalay requires the number of clusters as input, and uses k-means in the clustering part. Purity, adjusted Rand index, and silhouette score were used as evaluation metrics. Additionally, execution times were also compared. The performance of DBPCES is similar for almost all data streams to both algorithms in terms of all evaluation metrics. However, DenStream is very fast compared to DBPCES and implementation of Zubaroğlu and Atalay. The execution times of DBPCES and implementation of Zubaroğlu and Atalay is close which means that UMAP is the bottleneck for the execution time. To conclude, DBPCES achieves similar results with the other two algorithms although it does not require any data-dependent user-defined parameter.

The clustering results of DBPCES are evaluated by comparing its performance with DenStream and implementation of Zubaroğlu and Atalay on the same data sets. Although DBPCES clusters data streams successfully according to evaluation metrics used, there may not be a statistically significant difference between clusters. Therefore, some statistical tests can be performed to evaluate whether or not a statistically significant difference exists between predicted clusters. There are several methods available in the literature such as ANOVA, MANOVA, and t-test. Appropriate statistical tests should be determined to be performed according to data stream properties. Additionally, DBPCES estimates the parameters of DBSCAN using a heuristic in this study. Although DBPCES is successful with these estimated parameters, it should be tested that its success with estimated parameters is significantly different than selection of these parameters randomly. These tests are left as future work in this study.

Although DBPCES is performed well for all data streams, DBPCES is not as fast as DenStream because of UMAP that is used in dimensionality reduction. Additionally,

DBPCES adapts to concept drift by periodically running initialization and parameter selection phases instead of detecting concept drift. Reducing execution time of embedding and detecting concept drift are also left as future work.

REFERENCES

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

[2] "How much data is created every day? [27 staggering stats]." `https://seedscientific.com/how-much-data-is-created-every-day/`. Accessed: 2021-08-10.

[3] "Clustering: How it works (in plain english!)." `https://blog.dataiku.com/clustering-how-it-works-in-plain-english/`. Accessed: 2021-08-10.

[4] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[5] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *IEEE transactions on emerging topics in computing*, vol. 2, no. 3, pp. 267–279, 2014.

[6] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[7] A. Bhagat, N. Kshirsagar, P. Khodke, K. Dongre, and S. Ali, "Penalty parameter selection for hierarchical data stream clustering," *Procedia Computer Science*, vol. 79, pp. 24–31, 2016.

[8] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[9] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, "Dbscan: Past, present and future," in *The fifth international conference on the applications of*

digital information and web technologies (ICADIWT 2014), pp. 232–238, IEEE, 2014.

[10] "Understanding curse of dimensionality." `https://www.mygreatlearning.com/blog/understanding-curse-of-dimensionality/`. Accessed: 2021-08-29.

[11] C. O. S. Sorzano, J. Vargas, and A. P. Montano, "A survey of dimensionality reduction techniques," *arXiv preprint arXiv:1403.2877*, 2014.

[12] M. Bahri, A. Bifet, S. Maniu, and H. M. Gomes, "Survey on feature transformation techniques for data streams," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

[13] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[14] F. Wickelmaier, "An introduction to mds," *Sound Quality Research Unit, Aalborg University, Denmark*, vol. 46, no. 5, pp. 1–26, 2003.

[15] G. J. McLachlan, *Discriminant analysis and statistical pattern recognition*, vol. 544. John Wiley & Sons, 2004.

[16] H. Li, T. Jiang, and K. Zhang, "Efficient and robust feature extraction by maximum margin criterion," *Advances in neural information processing systems*, vol. 16, no. 16, pp. 97–104, 2004.

[17] M. Artac, M. Jogan, and A. Leonardis, "Incremental pca for on-line visual learning and recognition," in *Object recognition supported by user interaction for service robots*, vol. 3, pp. 781–784, IEEE, 2002.

[18] J. Weng, Y. Zhang, and W.-S. Hwang, "Candid covariance-free incremental principal component analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 1034–1040, 2003.

[19] S. Günter, N. Schraudolph, S. Vishwanathan, *et al.*, "Fast iterative kernel principal component analysis," 2007.

[20] X. Zhang, H. Huang, K. Mueller, and S. Yoo, "Streaming classical multidimensional scaling," in *2018 New York Scientific Data Summit (NYSDS)*, pp. 1–2, IEEE, 2018.

[21] S. Pang, S. Ozawa, and N. Kasabov, "Incremental linear discriminant analysis for classification of data streams," *IEEE transactions on Systems, Man, and Cybernetics, part B (Cybernetics)*, vol. 35, no. 5, pp. 905–914, 2005.

[22] J. Ye, Q. Li, H. Xiong, H. Park, R. Janardan, and V. Kumar, "Idr/qr: An incremental dimension reduction algorithm via qr decomposition," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 9, pp. 1208–1222, 2005.

[23] J. Yan, B. Zhang, S. Yan, Q. Yang, H. Li, Z. Chen, W. Xi, W. Fan, W.-Y. Ma, and Q. Cheng, "Immc: incremental maximum margin criterion," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 725–730, 2004.

[24] S. S. Vempala, *The random projection method*, vol. 65. American Mathematical Soc., 2005.

[25] D. L. Donoho, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[26] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 1113–1120, 2009.

[27] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, 2004.

[28] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[29] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[30] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[31] M. H. Law, N. Zhang, and A. K. Jain, "Nonlinear manifold learning for data stream," in *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 33–44, SIAM, 2004.

[32] F. Schoeneman, S. Mahapatra, V. Chandola, N. Napp, and J. Zola, "Error metrics for learning reliable manifolds from streaming data," in *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 750–758, SIAM, 2017.

[33] R. M. Parra-Hernández, J. I. Posada-Quintero, O. Acevedo-Charry, and H. F. Posada-Quintero, "Uniform manifold approximation and projection for clustering taxa through vocalizations in a neotropical passerine (rough-legged tyrannulet, phyllomyias burmeisteri)," *Animals*, vol. 10, no. 8, p. 1406, 2020.

[34] A. Zubaroğlu and V. Atalay, "Online embedding and clustering of data streams," in *Proceedings of the 2019 3rd International Conference on Big Data Research*, pp. 142–146, 2019.

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

[36] K.-s. Zhang, L. Zhong, L. Tian, X.-y. Zhang, and L. Li, "Dbiecm-an evolving clustering method for streaming data clustering," *Amse Journals-Amse Iieta*, vol. 60, no. 1, pp. 239–254, 2017.

[37] J. de Andrade Silva, E. R. Hruschka, and J. Gama, "An evolutionary algorithm for clustering data streams with a variable number of clusters," *Expert Systems with Applications*, vol. 67, pp. 228–238, 2017.

[38] D. Puschmann, P. Barnaghi, and R. Tafazolli, "Adaptive clustering for dynamic iot data streams," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 64–74, 2016.

[39] F. Cao, M. Estert, W. Qian, and A. Zhou, "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, 2006.

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

[41] Y. El-Sonbaty, M. A. Ismail, and M. Farouk, "An efficient density based clustering algorithm for large databases," in *16th IEEE international conference on tools with artificial intelligence*, pp. 673–677, IEEE, 2004.

[42] R. T. Ng and J. Han, "Clarans: A method for clustering objects for spatial data mining," *IEEE transactions on knowledge and data engineering*, vol. 14, no. 5, pp. 1003–1016, 2002.

[43] J. Hou, H. Gao, and X. Li, "Dsets-dbscan: A parameter-free clustering algorithm," *IEEE Transactions on Image Processing*, vol. 25, no. 7, pp. 3182–3193, 2016.

[44] M. Pavan and M. Pelillo, "Dominant sets and pairwise clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 1, pp. 167–172, 2006.

[45] X. Yu, D. Zhou, and Y. Zhou, "A new clustering algorithm based on distance and density," in *Proceedings of ICSSSM'05. 2005 International Conference on Services Systems and Services Management, 2005.*, vol. 2, pp. 1016–1021, IEEE, 2005.

[46] P. Liu, D. Zhou, and N. Wu, "Vdbscan: varied density based spatial clustering of applications with noise," in *2007 International conference on service systems and service management*, pp. 1–4, IEEE, 2007.

[47] A. Fahim, A. Salem, F. Torkey, and M. Ramadan, "Density clustering based on radius of data (dcbrd)," *World Academy of Science, Engineering and Technology*, 2006.

[48] M. N. Gaonkar and K. Sawant, "Autoepsdbscan: Dbscan with eps automatic for large dataset," *International Journal on Advanced Computer Theory and Engineering*, vol. 2, no. 2, pp. 11–16, 2013.

[49] M. T. Elbatta and W. M. Ashour, "A dynamic method for discovering density varied clusters," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 6, no. 1, 2013.

[50] A. Amini, T. Y. Wah, and H. Saboohi, "On density-based data streams clustering algorithms: A survey," *Journal of Computer Science and Technology*, vol. 29, no. 1, pp. 116–141, 2014.

[51] F. Can, "Incremental clustering for dynamic information processing," *ACM Transactions on Information Systems (TOIS)*, vol. 11, no. 2, pp. 143–164, 1993.

[52] F. Can and E. A. Ozkarahan, "Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases," *ACM Transactions on Database Systems (TODS)*, vol. 15, no. 4, pp. 483–517, 1990.

[53] L.-x. Liu, H. Huang, Y.-f. Guo, and F.-c. Chen, "rdenstream, a clustering algorithm over an evolving data stream," in *2009 international conference on information engineering and computer science*, pp. 1–4, IEEE, 2009.

[54] J. Ren and R. Ma, "Density-based data streams clustering over sliding windows," in *2009 Sixth international conference on fuzzy systems and knowledge discovery*, vol. 5, pp. 248–252, IEEE, 2009.

[55] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International journal of computer vision*, vol. 77, no. 1, pp. 125–141, 2008.

[56] W. Yu, Y. Gu, J. Li, S. Liu, and Y. Li, "Single-pass pca of large high-dimensional data," *arXiv preprint arXiv:1704.07669*, 2017.

[57] W. Feng, Z. Yan, L. Ai-ping, and W. Quan-yuan, "Online classification algorithm for data streams based on fast iterative kernel principal component analysis," in *2009 Fifth International Conference on Natural Computation*, vol. 1, pp. 232–236, IEEE, 2009.

[58] M. Bahri, B. Pfahringer, A. Bifet, and S. Maniu, "Efficient batch-incremental classification using umap for evolving data streams," in *International Symposium on Intelligent Data Analysis*, pp. 40–53, Springer, 2020.

[59] “Emcstream; an algorithm for online embedding and clustering of evolving data streams..” https://gitlab.com/alaettinzubaroglu/emcstream/. Accessed: 2021-08-10.

[60] M. Hahsler, M. Bolanos, and J. Forrest, “streammoa: Interface for moa stream clustering algorithms (2015),” *URL https://cran. r-project. org/web/packages/streamMOA*.

[61] R. C. Team *et al.*, “R: A language and environment for statistical computing,” 2013.

[62] “Density-based and parameterless clustering of embedded data streams (dbpces).” https://github.com/ozlempoyraz/DBPCES. Accessed: 2021-09-25.