# Mining Complex Data in Highly Streaming Environments

A thesis submitted in fulfilment of the requirements for
the degree of Doctor of Philosophy

**Zhinoos Razavi Hesabi**
Master of Computer Engineering, Azad University-South Tehran Branch

School of Science
College of Science, Engineering and Health
RMIT University

JUNE, 2019

# DECLARATION

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Zhinoos Razavi Hesabi
School of Science
RMIT University
$20^{th} March, 2019$

# ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my advisor Prof. Timos Sellis for his continuous support of my PhD study and related research, for his patience, motivation, and immense knowledge. His guidance helped me throughout my research and writing of this thesis. I am thankful for having him as my supervisor.

I would also like to thank to my other supervisor, A/Prof.Jenny Zhang, for her insightful comments and encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without my supervisory team.

My sincere thanks also goes to Prof.Zahir Tari, A/Prof.Vic.Ciesielski, Prof.James Harland and A/Prof.Sebastian Sardina who provided me with an opportunity to join their teaching team as tutor, and who provided me with an opportunity to govern and develop my teaching skills in the area of my expertise.

Last but not least, I would like to thank my family: my parents, my brother and sisters for supporting me spiritually throughout writing this thesis and my life in general.

# Table of Contents

# List of Figures

# List of Tables

# Abstract

Data is growing at a rapid rate because of advanced hardware and software technologies and platforms such as e-health systems, sensor networks, and social media. One of the challenging problems is storing, processing and transferring this big data in an efficient and effective way. One solution to tackle these challenges is to construct synopsis by means of data summarization techniques. Motivated by the fact that without summarization, processing, analyzing and communicating this vast amount of data is inefficient, this thesis introduces new summarization frameworks with the main goals of reducing communication costs and accelerating data mining processes in different application scenarios. Specifically, we study the following big data summarizaion techniques: (i) dimensionality reduction; (ii) clustering, and (iii) histogram, considering their importance and wide use in various areas and domains.

In our work, we propose three different frameworks using these summarization techniques to cover three different aspects of big data: "**V**olume", "**V**elocity" and "**V**ariety" in centralized and decentralized platforms. We use dimensionality reduction techniques for summarizing large 2D-arrays, clustering and histograms for processing multiple data streams.

With respect to the importance and rapid growth of emerging e-health applications such as tele-radiology and tele-medicine that require fast, low cost, and often lossless access to massive amount of medical images and data over bandlimited channels, our first framework attempts to summarize streams of large volume medical images (e.g. X-rays) for the purpose of compression. Significant amounts of correlation and redundancy exist across different medical images. These can be extracted and used as a data summary to achieve better compression, and consequently less storage and less communication overheads on the network. We propose a novel memory-assisted compression framework as a learning-based universal coding, which can be used to complement any existing algorithm to further eliminate redundancies/similarities across images. This approach is motivated by the fact that, often in medical applications, massive amounts of correlated images from the same family are available as training data for learning the dependencies and deriving appropriate reference or synopses models. The models can then be used for compression of any new image from the same family. In particular, dimensionality reduction techniques such as Principal Component Analysis (PCA) and Non-negative Matrix Factorization (NMF) are applied on a set of images from training data to form the required reference models. The proposed memory-assisted compression allows each image to be processed independently of other images, and hence allows individual image

access and transmission.

In the second part of our work, we investigate the problem of summarizing distributed multidimensional data streams using clustering. We devise a distributed clustering framework, DistClusTree, that extends the centralized ClusTree approach. The main difficulty in distributed clustering is balancing communication costs and clustering quality. We tackle this in DistClusTree through combining spatial index summaries and online tracking for efficient local and global incremental clustering. We demonstrate through extensive experiments the efficacy of the framework in terms of communication costs and approximate clustering quality.

In the last part, we use a multidimensional index structure to merge distributed summaries in the form of a centralized histogram as another widely used summarization technique with the application in approximate range query answering. In this thesis, we propose the index-based Distributed Mergeable Summaries (iDMS) framework based on $kd$-trees that addresses these challenges with data generative models of Gaussian mixture models (GMMs) and a Generative Adversarial Network (GAN). iDMS maintains a global approximate $kd$-tree at a central site via GMMs or GANs upon new arrivals of streaming data at local sites. Experimental results validate the effectiveness and efficiency of iDMS against baseline distributed settings in terms of approximation error and communication costs.

**Keywords:** Big data, Data Summarization, Dimensionalty reduction, Compression, Clustering, Histograms, Distributed computing, Index data structures

CHAPTER 1

# INTRODUCTION

With the advent of online technologies such as filmless imaging, online Global Positioning System and sensory data, huge volumes of data with large variety and high velocity are produced daily. These are so-called *big data*. Although a collection of this huge data is beneficial in serving customers more efficiently, the problems of storing, transmitting and processing big data are still challenging.

As data volumes grow rapidly, their applications need to adapt with the speed of generating data to answer requirements of users on demand. For example, many queries, database processing and mining algorithms need to be performed in an efficient way while it is in contrast with online demands of users and the speed of data streams. One solution to deal with this problem is to process and analyse summarized data instead of the complete data. Therefore, summarization is considered as a descriptive task in data mining to provide a synopsis- based representation of data. It makes many tasks such as pre-processing, analysis and management of data easier and faster by reducing the size of data, thus overcoming the space, transmission and time limitations.

## 1.1 Background

Although summarization can be performed in various ways, the aim of this thesis is to introduce and develop new big data summarization algorithms and frameworks using a variety of machine learning techniques (e.g. dimensionality reduction, clustering) and

statistical distributions to lessen the limitations of traditional big data summarization approaches, especially in the context of streaming data. The focus is on the development of new summarization frameworks compatible with much more complex data such as positional data (e.g. GPS coordinates), big two-Dimensional arrays (e.g. large medical images) or distributed streaming data (e.g. sensor data) which can be used for different objectives such as those listed below.

- **Approximate Query Processing**: Today, most queries need to be answered online. Processing vast amounts of data which are produced continuously is not efficient and practical with traditional data processing techniques. Therefore, data summarization is an efficient approach to quickly process queries at the expense of having only approximate answers. Sampling, histograms and sketches are examples of summarization algorithms that can be used to answer the query estimation problem [8].

- **Data Aggregation**: In many parallel and distributed systems, calculating aggregate statistics over the entire distributed data streams is required. Approximation of frequency counts, quantiles and heavy hitters are examples of such applications which can be resolved by using summarization techniques such as histograms or sketches [9].

- **Telecommunication Applications**: Recent advances in information technology such as tele-medicine and tele-radiology require storage and transmission of huge amounts of data. The rate of data growth is much faster than the rate of growth of technologies. As a result, transmission and storage of these large data are problematic. Data compression is a solution to reduce the size of data and thus communication costs. Compression techniques discover patterns and redundancy in the data and transform original data into a compact version. Summarization and sketching techniques such as dimensionality reduction can be used to extract these patterns from data to build a compact version of that data.

- **Data Mining Applications**: In some predictive analysis and machine learning algorithms such as concept drift detection, statistical properties of data such as mean or variance are enough to predict and detect changes over time. This means it does not need to have all and actual data points to detect changes and temporal synopses are adequate to track the behavior of the stream. Summarization techniques such as clustering can be useful to detect concept drift effectively [9].

In this thesis, we also consider different aspects when dealing with big data, in order to devise and develop those algorithms that are suitable for the specific cases encountered.*Volume* of the data is the first and obvious important characteristic to deal with when summarizing big data compared to conventional data summarization, as this requires substantial changes in the architecture of storage systems. Another important characteristic of big data is *Velocity*. This requirement leads to high demand for online processing of data where rapid processing is required to deal with data flows. *Variety* is the third characteristic, where different data types such as numerical data (one-Dimensional values) and images (two-Dimensional arrays) are produced from various sources of big data such as sensors, telemedicine applications and mobile phones. These three "Vs" (Volume, Velocity and Variety) are the core characteristics of big data which we take into account when designing data summarization algorithms. In this thesis we cover these aspects and new concerns of big data by proposing new summarization/mining frameworks, then test and evaluate our algorithms on various synthetic and real data sets.

The design and development of a summarization algorithm depends on the type of data and the application being used. For instance, a summarization algorithm for an aggregation problem might be different from summarization algorithm that is used for a compression problem. In this thesis we aim to design summarization algorithms that are multi-purpose and applicable to a variety of data streams considering their efficiency issues of time and space constraints. There are various methods for constructing a summary but we study only three of them, listed below, with the focus on reducing communication costs, time and space.

- **Dimensionality Reduction**: Dimensionality reduction techniques reduce the number of random variables by discovering a set of principal variables. Some of dimension reduction techniques linearly or non-linearly transform high dimensional data to a space with fewer dimensions. For example, Principal Component Analysis (PCA) is one of the main linear reduction techniques that transforms data from a higher dimensional space to a lower dimension space by maximizing variance of data in lower dimension space and decorelation of data.

- **Micro-Clustering**: Micro-clustering is a stream mining technique which can be used for constructing summaries. Micro-clustering consists of two phases: online and offline. In the online phase, a summary of data is collected and stored in a data structure. This summary is used for further mining analysis in the offline phase. Micro-clustering can be used for both single and multi-dimensional data.

- **Histograms**: Histograms are another technique of summarizing data. A histogram maintains frequency counts of items within specific intervals. There are different types of histograms. One way of representing a histogram is by discretizing data into equal width partitions (i.e. buckets), and storing the frequency of occurrence of each data item within these buckets. Equi-width histograms can be easily used to answer range queries as the only requirement is defining user-specified intervals/buckets. However, this type of histogram may inject inaccuracy in answering range queries. The reason is that by dividing a range into equi-width partitions and assigning data points unequally across different buckets, if these buckets contain the range boundary of a query, it causes an inaccurate query estimation. Thus a solution to this problem is to uniformly divide data into equi-height/depth buckets. In this case, each range encompasses approximately an equal number of data points. Construction of equi-depth histograms is the same as construction of quantiles in data streams where equi-depth buckets determine different quantiles in the data.

## 1.2 Motivation

We have designed and developed new summarization/approximation algorithms using the above summarization techniques and taking into account the following application requirements.

- Offline vs. online
- Single stream vs. multiple stream
- Centralized vs. decentralized (aggregation)

In *Offline summarization*, the whole data set is available and data can be accessed many times, while in *online summarization* the entire data set is not available and data

can only be scanned once. Many studies have investigated offline summarization through clustering while a few more recent works have concentrated on online clustering.

On the other hand, with advances in data collection and generation technologies, we are faced with environments that are equipped with *multiple distributed computing nodes* that generate *multiple streams* compared to generating a single stream such as sensor networks. Therefore many systems use a centralized model for clustering multiple streams while some use decentralized (distributed) stream clustering.

Most of the literature focuses on processing a *single stream in a centralized fashion*. However, centralized mining has some limitations over the distributed one including: long response times, possible bottlenecks, and excessive power consumption due to excessive data communication (since all streaming data need to be transmitted to a central site in order to be processed). Hence, more effective *distributed online* mining algorithms are needed.

## 1.3 Research Questions and Contributions

The main research questions that are addressed in this thesis are as follows.

1. **How to summarize big volume 2D-array data to overcome limitations of conventional storage and transmission approaches?**

   Conventional storage and transmission of large 2D-arrays such as medical images can be very expensive and time consuming. We reduce the limitations of conventional storage and transmission by answering this research question as follows.

   - We have proposed a novel framework called Memory-Assisted Compression (MAC) technique. The rationale behind MAC is learning source statistics at some intermediate entities, and then leveraging the memorized context to reduce redundancy of the universal compression of final length sequences. Hence, the communication overhead to send the sequence from sender to receiver is reduced if memorized context (i.e. summarized data) is available to the encoder and the decoder.
   - We improve the proposed MAC framework in which redundancy across images (2D-arrays) is extracted to represent data in a more compact (i.e. summarized) way, by using a more effective factorization matrix.

2. **How to summarize and maintain multiple data streams through clustering technique in centralized and decentralized manner to deal with high demands of online processing?**

   Answering this research question covers the velocity aspect of big data which aims to minimise the response time.

   - First, we propose a centralized concurrent multi-stream clustering algorithm that relies on an index data structure for storing and maintaining a compact view of the current clustering (online phase).
   - We then extend this algorithm to a distributed one where a centralized continuous clustering is run over the union of online clusters received from all local sites in a central location.

3. **How to efficiently merge and maintain multidimensional local summaries in the form of histograms in distributed environments?**

   - We have proposed a new framework that enables one to merge and maintain local summaries in a global index structure. The centralized index maintains a compact summary over all summaries from all local sites.
   - We have proposed a new method using an index structure to maintain statistical distributions (i.e. a histogram) instead of just maintaining actual data points.

More specifically, we outline our main contributions in this PhD thesis.

1. Although many works have addressed offline summarization of big data, there are still some gaps that can be explored by new summarization algorithms, which can then be adapted to online and may be enhanced to be compatible with the distributed model. We designed and developed an offline compression (summarization) algorithm by reducing entropy of large volume of 2D-arrays (i.e. large raw medical images) with respect to its application in filmless imaging technology (such as telemedicine). We have first shown that significant amounts of correlation and redundancy exist across different large medical images. Such a correlation can be utilized to achieve better compression, and consequently less storage and fewer communication overheads on the network. We proposed a novel MAC technique as a learning-based universal coding. This can be used to complement any existing algorithm to further eliminate redundancies across images. The approach is

motivated by the fact that often in medical applications, large numbers of correlated images from the same family are available as training data for learning the dependencies and deriving appropriate reference models. Such models can then be used for compression of any new image from the same family. We further improved the performance of the algorithm by taking advantage of some other matrix factorization techniques and adding an extra layer in the proposed framework.

2. Many works have focused on online single stream summarization, so called micro-clustering. However, less work has been done on multiple stream clustering. Therefore, we propose a new concurrent multiple-streams clustering algorithm in a centralized model. The algorithm is based on a well-known micro-clustering techniques (ClusTree) [13] in the literature which includes two phases: online and offline. We captured the summary statistics of multiple data streams in an online phase. Then we maintained statistical information about the data locality in terms of micro-clusters in a parallel index data structure for further offline clustering. In the online phase, this index maintains cluster feature tuples instead of storing all incoming objects. We also extended the centralized ClusTree into a distributed one. Although some distributed clustering have been studied in the literature, the lack of being near real-time of these works motivated us to proposed a new online distributed clustering algorithm which not only reduces the communication cost but also reduces processing time of clustering distributed data.

3. Multidimensional data indexing structures have been utilized in centralized data bases and data mining communities. Although index structures are good tools to represent summaries of data distributions as histograms, quantiles and data clusters, a lack of their extension and usage is noticeable in distributed settings and parallel processing. Therefore, it is of interest to develop new practical algorithms that can benefit from using index data structures in these environments. In this thesis, we studied a two-fold project. First, how to keep quantiles and histograms in the union of distributed summaries in multidimensional spaces. Second, how to keep online distributed tracking of a multidimensional function in the union of the entire data, where function is an index tree. We have introduced a new practical framework capable of maintaining quantiles over union of distributed multidimensional data stream summaries. In our framework, we keep distributed data summaries in index structures. We merge index trees to discover quantiles in the entire distributed data. Merging index trees means that, given two index trees

on two data sets, there is a method to merge them into a global index tree while preserving error and size of index structure. We use kd-tree, an index structure, to represent overall quantiles in a homogeneous distributed multidimensional space. The proposed framework constructs a global approximate kd-tree over the union of distributed summaries. Our method incrementally updates the global kd-tree. Experimental results provided evidence on the performance of the proposed method when it has been utilized in parallel and distributed settings in terms of communication costs, error rate and the practicality of our algorithm. We also proposed a new method to create a data structure which stores and maintains clusters of data in entire distribution. Our method is adjustable to be used in a streaming scenario. It is a good tool for use in a load balancing algorithm to optimize resource use, maximize throughput, minimize response time and avoid overload of any single resource.

## 1.4 Publications

Research outcomes of this PhD study have been published/submitted in chronological order as follows.

1. **Z.R.Hesabi**, M.Sardari, A.Beirami, F.Fekri, M.Deriche and A. Navarro, "A memory assisted lossless compression algorithm for medical images", In Proc.Of 39th International Conference on Acoustics, Speech and Signal Processing(ICASSP), Florence, Italy, 2014.The content of this paper is included in Chapter 3.

2. **Z.R.Hesabi**, Z. Tari, A. Goscinski, A. Fahad, I. Khalil, and C. Queiroz. "Data Summarization Techniques for Big Data - A Survey", In Handbook on Data Centers, pp. 1109-1152, Springer New York, 2015.The content of this chapter is included in Chapter 2.

3. **Z.R.Hesabi**, B.Kazimipour, M.Deriche and A.Navarro, "A multilevel memoryassisted lossless compression algorithm for Medical Images", In Proc. of the 23rd European Signal Processing Conference(EUSIPCO), Nice, France, 2015. The content of this chapter is included in Chapter 3.

4. **Z.R.Hesabi**, T.Sellis, X.Zhang, "Anytime Concurrent Clustering of Multiple Streams with an indexing Tree", In Proc. of the 4th International Workshop on Big Data, Streams and Heterogeneous Source Mining(BigMine 2015), JMLR WCP,vol.41, Sydney, Australia, 2015. The content of this chapter is included in Chapter 4.

5. **Zhinoos Razavi Heasbi**, Timos Sellis, Kewn Liao, "DistClusTree: A Framework for Distributed Stream Clustering", 29th Australasian Database Conference, 2018. The content of this chapter is included in Chapter 4.

6. **Zhinoos Razavi Hesabi**, Timos Sellis, Kewn Liao, Shahab Razavi, Prince Hemrajani, "iDMS: An Index-based Framework for Tracking DistributedMultidimensional Streams", to be submitted. The content of this chapter is included in Chapter 5.

## 1.5 Thesis Overview

The rest of this thesis is organized as follows. In Chapter 2 we broadly review related works on the most important summarization techniques in the literature. In Chapter 3 we propose a medical image summarization framework using dimensional reduction techniques with the focus on data volume. In Chapter 4 our contribution on speed of summarization using clustering in centralized and decentralized platforms is presented. The focus of Chapter 5 is on merging distributed summaries in the form of a histogram. Finally Chapter 6 summarises the contributions of this research thesis and includes potential future research directions.

CHAPTER 2

# LITERATURE REVIEW

In the current digital era, the massive progress and development of internet and online world technologies such as big and powerful data servers has meant we face huge volumes of information and data daily from many different resources and services which were not available to human kind just a few decades ago. This data comes from different online resources and services which are established to serve the customers. Services and resources such as Sensor Networks, Cloud Storages and Social Networks produce “big data” and there is also a need to manage and reuse that data or (some analytical aspects of it). Although this massive volume of data can be really useful for people and corporations, it could also cause problems. Therefore big data has its own deficiencies as well. They need big storage, and this volume makes operations such as analytical operations, process operations and retrieval operations really difficult and hugely time-consuming. One way to resolve these difficult problems is to have big data summarized so they need less storage and much less time to be processed and retrieved. The summarized data will then be in “compact format” and still be an informative version of the entire data.

## 2.1 Introduction

The aim of this chapter is to provide an overall view of different data summarization techniques [9] found in the literature, including clustering, sampling, compression, histograms, wavelets and micro-cluster with respect to their applications in a variety of fields such as data mining. However the focus here will be the selection of those techniques that are suitable for big data. Some aspects then need careful attention when dealing with big data, so this chapter helps to select relevant techniques. The Volume of the data is the first and obvious important characteristic to deal with when summarizing big data compared to conventional data summarization, as this requires substantial changes in the architecture of storage systems. Another important characteristic of big data is Velocity. This requirement leads to highly demand for on line processing of data where rapid processing is required to deal with data flows. Variety is a third characteristic, where different data types such as text, image and video are produced from various sources such as sensors and mobile phones. These three "V"s (Volume, Velocity and Variety) are the core characteristics of big data [10] which must be taken into account when selecting data summarization techniques.

*Summarization* can be performed in various ways. We have selected here the following summarization techniques that are applicable for big data.

- **Clustering** is an unsupervised summarization technique that aims to gather similar objects into groups or "clusters". The similarity among objects can be determined through different metrics, such as distance. Clustering algorithms can be categorized into different models such as hierarchical, partitioning, density-based and grid-based.

- **Sampling** is another summarization technique that provides a concise and still informative representation of the entire data set. A sample is representative of a larger group (population) which preserves the same characteristics of the population and a study is made of the sample instead of the whole population. There are two main categories of sampling techniques, namely probability-based and nonprobability-based: either provides an efficient way to summarize big data.

- **Compression** is a well-known technique that represents data in a compact way to save time and space. Lossless and lossy are two different compression methods which are broadly used in different areas such as video and image coding. We will

explain a compression method based on a minimum description length principle and its applications for data summarization in this chapter.

- **Wavelets** can be considered as a summarization technique that is mostly used in image and query processing applications. Different wavelet transformations such as (Haar) and dimensional wavelets are used to transform data from one domain to another. We will show in this chapter how wavelet transformations can be applied to summarize data.

- **Histogram** is a method used to represent a large volume of data in a compact manner so that it can be considered as a data reduction or summarization technique. In fact, data distribution can be shown in a synopsis structure through histograms. Accordingly, we will discuss some of the various types of histograms but we will not elaborate on them further.

- **Micro-clustering** is a method to construct a synopsis model of data stream that considers evolving behavior of data streams. In this chapter, we will explain more details about micro-clustering as another summarization technique for data streams.

## 2.2 Chapter Organization

The roadmap of this chapter is as follows. Details about some of the well-known data summarization techniques available in the literature will be discussed in order. Each technique will be explained and then a number of research directions that are conducted on each technique will be briefly reviewed. Finally, Section 2.4 concludes this chapter.

## 2.3 Big Data Summarization Techniques

### 2.3.1 Clustering algorithms

One of the most used techniques having the purpose of data summarization is clustering. This is an unsupervised process of collecting similar data objects in a group, where data objects within the same cluster are more similar to each other than to objects in other clusters. The goal of clustering is to group similar objects together to simplify further processing such as data mining, summarization and analysis.

To cluster data points, the following questions need to be addressed. What are the differences in the cluster data points? What are the similarity metrics that can be used to cluster data points in the same group? How are these similarities measured? What types of data can be used in different clusters? How are the discovered clusters evaluated? Is it possible to cluster all data points of the entire data sets? Do the clusters have the same shapes? How many clusters are required to present data objects? And, last but not least, how large could a data set be to cluster its data points?

In answering these questions, one needs to classify clustering algorithms into different categories. Amongst those that could relate to big data, one can find Hierarchical, Partitional, Density-based and Grid-based algorithms. We also note that in clustering data points, some issues such as requirements of clustering algorithms should be considered. Some examples include scalability, dealing with different type of attributes, finding arbitrary shape clusters, the ability to cope with outliers and noise, considering high dimensional data sets, interpretability and usability.

To better understand clustering algorithms, we will first identify different types of data and then explain a preliminary principle of a clustering algorithm, which is a proximity measure, a common phrase to represent similarity $s(i,j)$ and dissimilarity $d(i,j)$ measure between two data points, two clusters or a data point and a cluster.

- **Data Type** In order to secure universal interpretation of data, data has been categorized into two scales of measurement: qualitative and quantitative. The former includes nominal scales and ordinal scales. The latter includes interval and ratio scales. In addition to these scales, other words describing types of data include categorical, numerical, binary, continuous and discrete.

- **Similarity/Dissimilarity Measures** Based on the type of data, various similarity or dissimilarity measures can be defined. Therefore, some similarity measures are briefly reviewed. One of the most-used similarities metric is distance. There are many distance measures which are mostly used in clustering algorithms, such as Minkowski , Manhattan or City block, Euclidean and Mahalanobis which are described below.

**Minkowski distance** For two numerical points in the form of $x_i = (x_{i1}, x_{i2}, \cdots, x_{il})$ and $x_j = (x_{j1}, x_{j2}, \cdots, x_{j1})$, Minkowski distance or $L_p$ norm is calculated as shown in Equation 3.8.

$$D_{ij} = \left[\Sigma_{l=1}^{d} |x_{il} - x_j l|^{\frac{1}{n}}\right]^n \tag{2.1}$$

An example of application of Minkowski distance can be found in [10].

**Euclidean distance** Euclidean distance or $L_2$ norm is a commonly used measure of distance in clustering algorithms such as [11] to find similar numerical objects and tend to find hyperspherical clusters. It is a particular instance of Minkowski at $n = 2$. It measures the distance between two points of $x_i = (x_{i1}, x_{i2}, \cdots, x_{i1})$ and$x_j = (x_{j1}, x_{j2}, \cdots, x_{j1})$ as shown in Equation 2.2.

$$D_{ij} = \left[\Sigma_{l=1}^{d} |x_{il} - x_j l|^{\frac{1}{2}}\right]^2 \tag{2.2}$$

K-means, CURE and BIRCH algorithms are examples of clustering algorithms that measure similarities between data points based on closeness via the Euclidean distance measure.

**Manhattan / City block distance** Considering Minkowski at $n = 1$ for two numerical points of $x_i = (x_{i1}, x_{i2}, \cdots, x_i 1)$ and $x_j = (x_{j1}, x_{j2}, \cdots, x_{j1})$ gives a Manhattan or City block distance or $L_1$ norm as in [12] as Equation 2.3.

$$D_{ij} = \left[\Sigma_{l=1}^{d} |x_{il} - x_j l|\right] \tag{2.3}$$

Manhattan or Ciwqty block distance normally involves finding hyper-rectangular clusters.

**Mahalanobis distance** Mahalanobis distance considers the correlation between variables or the variance-covariance matrix. Hyper-ellipsoidal clusters can be discovered through applying Mahalanobis distance which is formulated in Equation 2.4.

$$D_{ij} = (x_i - x_j)^T S^{-1} (x_i - x_j) \tag{2.4}$$

where $S$ is the within cluster covariance matrix. Examples of Mahalanobis distance can be found in [13], [14].

There are also different similarity and dissimilarity measures for categorical data. The simple matching distance proposed in [15] is one of the well-known ones that measure dissimilarity between categorical data. Let $x$ and $y$ be two categorical data points. The simple matching distance between $x$ and $y$ is computed as depicted in Equation 2.5.

$$\delta(x,y) = \begin{cases} 0 & if\, x = y \\ 1 & if\, x \neq y \end{cases} \tag{2.5}$$

Equation 2.6 calculates dissimilarity metric based on the simple matching distance for two categorical data points of $x$ and $y$ with $l$ attributes.

$$d_{sim} = \Sigma_{j=1}^{l} \delta(x_i, y_i) \tag{2.6}$$

Note that there are many more similarity/dissimilarity measures in the literature. However, due to the lack of space, we have only explained the most common ones and listed some of them in the following.

Some other dissimilarity measures for numerical data are: Mean character difference [16], index of association [17], Canberra metric and Coefficient of divergence [18], and Czekanowski coefficien [19]. Some matching coefficients measures for nominal data are: Russell and Ra [20], simple matching [21], Jaccord [22], Rogers-Tanimoto [23] and Kulczynski [24]. There are also some similarity measures for binary data such as Jaccord, Dice, Pearson, Sokal-Sneatha/b/c/d, Yule and Ochiai. Some of these are summarized in [25]. The other metric that considers similarity between groups of objects is linkage criterion or connectivity between them.

After presenting some of the concepts related to data clustering, we will consider some of the most prominent clustering algorithms found in the literature. The rest of this section focuses on clusterings of very large data sets; therefore we will only describe those clustering algorithms that can be applied to very large data sets with the aim of summarization.

#### 2.3.1.1 Hierarchical clustering

Hierarchical clustering, also called Connectivity-based clustering, is one of the classical approaches. It creates clusters in the form of a tree in which each cluster is represented as a node. The main idea is that the structure is based on a proximity measure, where the nearby objects are clustered into a group. Therefore, distance between objects plays a pivotal role in the clustering of the data objects.

*Tree-based hierarchical* clustering algorithms can be of two types:

1. Bottom up (Divisive) or

2. Top down (Agglomerative).

In the **Divisive** approach, the clustering process starts from its root in such as way that the entire data set is considered as a large cluster in root; later it iteratively splits data into partitions, where it terminates at leaves level. There are two good examples of divisive clustering algorithms: MONA and DIANA. These are detailed in [26]; some applications of those algorithms are given in [27].

In the **Agglomerative** approach, each data point is considered as a single cluster at the leaf level, and then every two closest clusters based on proximity measures will be merged to achieve a single cluster at the root tree.

Although a broad range of agglomerative hierarchical clustering algorithms are fund in the literature, such as single linkage clustering [28], complete linkage clustering [29], group average clustering [30] and the centroid method [30], only some of them can be applied to very large data sets which is the focus of this chapter. The prominent reason that the above clustering algorithms are not selected to cluster large data sets is their quadratic computational complexity which is a function of number of data points. As there are several deficiencies with hierarchical clustering algorithms, new algorithms were proposed to cover their shortcomings, such as the high degree of sensitivity to noise and outliers, incapability of correcting previous misclassification and unclear termination criteria. It should be noted that highlighting defects of hierarchical clustering algorithms does not contravene their important benefits, such as handling any forms of similarity or distance, covering different types of attribute and not requiring knowing the number

of clusters in advance. Considering all the pros and cons of hierarchical clustering algorithms, this has led to describing some other hierarchical clustering algorithms that could be applied on large data sets, such as BIRCH, CURE and ROCK.

**BIRCH** [31] employs a tree structure which is called CF Tree (Clustering Feature Tree). This tree is a height-balanced one and consists of leaf nodes and intermediate nodes where each of them has certain entries. The number of entries is constrained by two branching factors, noted as $B$ and $L$. The B factor is a maximum number of entries for each intermediate node, and the factor $L$ represents the maximum number of entries for each leaf node. Each entry of intermediate node is in the form of $[CF_i, Child_i]$, in which $CF_i$ is a summary information consisting a 3-tuple $< N, LS, SS >$, where $N$ is the number of data points in a cluster, $LS$ is the linear sum, $SS$ is the square sum of the $N$ data points in a cluster and $Child_i$ is a pointer to its $i^{th}$ child node. Entries of the leaf nodes are also in the form of $[CF_i]$. The number of leaf entries is controlled by a threshold $T$ which is set to 0 by default. The height of tree is also defined by $T$. The larger $T$ leads to the smaller tree.

This algorithm is a local one since it does not scan all the data points once and it starts with sub-clustering of leaf entries via closeness metric. Five alternative metrics are used to measure closeness of clusters: centroid Euclidian distance; centroid Manhattan distance; average inter-cluster distance; average intra-cluster distance, and variance increase distance. It clusters dense area as a single cluster and removes sparse area as an outlier. BIRCH starts by building a CF tree dynamically and incrementally based on available memory and adjustable threshold of T. Each entry is inserted to the CF tree based on the closest child node metric in leaves. If the number of entries of a leaf node does not exceed L, a new entry is inserted to this leaf node; otherwise the leaf node is divided and this division will be continued in ascend trend in the CF tree until a node, whether leaf or intermediate node, is found that has the capacity to add more entries. If this trend presumes up to the root, the root of CF tree will be split and therefore the height of tree will be increased by one.

Since a CF tree is built based on an agglomerative algorithm (which is a bottom up approach as previously explained) a new cluster in an upper level of CF-tree is constructed through merging two sub-clusters in a lower level of the CF Tree. In order to do that, the Additivity theorem is used to merge two clusters. Based on this Additivity theorem, CF vectors of two clusters are computed as below if two clusters are merged.

$$CF_m = CF_1 + CF_2 = (N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2) \tag{2.7}$$

where $CF_m$ in Equation 2.7 is a clustering feature of newly merged cluster. The insertion operation in CF Tree is similar to B+-tree and also an Additivity Theorem is used to build a CF Tree.

BIRCH's main goal is to minimize running time, memory and data scans. It also makes clustering decisions without scanning the whole data; the group dense areas is a single cluster and sparse areas as outliers are handled by removing them. Hence, it can handle outliers. Despite BIRCH's advantage, it has some deficiencies. One of the major problems of BIRCH is that it cannot perform well in the face of non-spherical shape clusters since a boundary of a cluster is controlled by the notion of radius or diameter. As mentioned earlier, BIRCH clusters data points by using clustering features of the original data instead of using the whole data and consequently it causes a reduction in storage space and frequent I/O operations. Further, its computational complexity of $O(N)$ means that for it to be used to cluster very large data sets its time and memory constraint must be explicit. Several extensions of BIRCH were proposed, which we have briefly described here. A clustering algorithm is proposed [32] where its pre-clustering phase is similar to BIRCH. In this way, the whole data set is scanned to find the dense areas which are then clustered by applying a hierarchical clustering algorithm and making a CF tree. In contrast with traditional clustering algorithms (which can deal with one of those attribute types), this algorithm can handle both continuous and categorical attributes. Therefore, the clustering feature CF has been changed to $CF_j = (N_j, S_{Aj}, S^2_{Aj}, N_{Bj})$, where $N_j$ shows the number of data spots in cluster $C_j$ , $S_{Aj}$ is the sum of consecutive features, $S^2_{Aj}$ is the square sum of consecutive features of $N_j$ data spots, and $N_{Bj}$ is a d-dimensional vector representing the value of categorical attributes and distance of pair of clusters is measured by a log-likelihood function.

BIRCH is generalized in [33] into a wider framework, called $BIRCH_*$, in distance spaces. This framework is based on two algorithms named BUBBLE and BUBBLE-FM. The parameters of CF vector in $BIRCH_*$ are a sum of the squared distance of a data point to other data points, the centroid of the cluster which is determined based on minimum squared distance and the radius r of the cluster which are components to build the CF-Tree. BIRCH has been extended in many more studies such as [34], [35], [36] and [37].

**CURE (Clustering Using REpresentatives) [34]** is another hierarchical clustering algorithm. Unlike BIRCH, CURE is robust against outliers and can deal with arbitrary-shape clusters. The handling of arbitrary shape clusters is due to the fact that each cluster is represented by a set of representative points instead of a single centroid or all-points. Therefore, it can find non-convex shape clusters. Furthermore, this set of representatives is shrunk towards a centroid through an adjustable parameter $\propto = [0,1]$ to deal with outliers. Shrinkage causes outliers come closer to the centroid of the cluster to avoid wrong clustering. CURE is designed to apply to large data sets by using random sampling and partitioning. First, a sample of the data set is chosen randomly, and then this sample is partitioned to K equal partitions. To reduce time complexity, these partitions are pre-clustered like the pre-clustering phase of BIRCH, and then an agglomerative hierarchical clustering is applied to each pre-cluster partition. At the end of the process, a label is assigned to each data points based on its distance from representatives. CURE applies two data structures in its algorithm, namely a kd-tree and a heap-tree. CURE stores its representatives in the kd-tree and clusters are stored in the heap-tree. The time complexity of CURE is $O(N^2_{sample})$ which depends on the number of sampling data and the number of partitions.

**ROCK (RObust Clustering using linKs) [35]** is an agglomerative hierarchical clustering algorithm that groups categorical data points through non-metric measures. The two metrics that measure either Euclidian distance in hierarchical clustering algorithms or a criterion function (such as square error) in a partition-clustering algorithm cannot properly measure similarity for categorical data points. Therefore, two similarities metrics were introduced in ROCK to enable accurate merging as well as clustering of data points. These metrics are: $sim(p_i, p_j)$ to consider neighbors of a point and link $(p_i, p_j)$ to define the number of common neighbors between two points $p_i$ and $p_j$. The similarity measure is defined as shown in Equation 2.8.

$$Sim(T_1, T_2) = \frac{|T_1 \cap T_2|}{T_1 \cup T_2} \tag{2.8}$$

where $|T_i|$ is the number of items in the transaction $T_i$ and $Sim(T_1, T_2) \geq \theta$; $0 \leqslant \theta \leqslant 1$. Data points are considered as transactions in a market basket. If no similarity is found between transactions, then $\theta = 0$; meaning that any pair of transactions can be neighbors of each other. If $\theta = 1$, only identical transactions can be considered as neighbors. Thus, it is important to properly define $\theta$, which is a user-specified parameter based on desired

closeness. The number of links between a pair of points also indicates the probability whether or not data points are presented in the same cluster. The larger the link is, the more probable it will be that two points belong to the same cluster. Using links allows ROCK to be robust. It is important to note that ROCK clusters data points in a similar way that CURE does, but the difference is that ROCKS uses links and different similarity measures instead of distance measure. It can also handle outliers. Finally, ROCK uses random sampling and labeling techniques, which makes it a good approach to deal with very large data sets.

Table 2.1 provides a summary of characteristics of these three well-known algorithms, namely BIRCH, CURE and ROCK. Our investigation regarding such clustering approaches for big data can be summarized as follows. BIRCH is suitable for large data sets where finding spherical shape clusters in a linear time is required. CURE can be used to search arbitrary-shape clusters of numeric data in a large data set. CURE is robust against outliers. Furthermore, CURE can fit within available memory since a random sample of a large data set is chosen to perform clustering. ROCK will tackle the presence of categorical data in large data sets.

Table 2.1: *Characteristics of BIRCH, CURE and ROCK clustering algorithms*

| **Algorithm** | **Type of data** | **Cluster shape** | **Time complexity** | **Space complexity** |
|---|---|---|---|---|
| BIRCH | Numerical | Spherical | $O(N)$ | - |
| CURE | Numerical | Arbitrary | $O(N^2_{sample}\log N_{sample})$ | $O(N_{sample})$ |
| ROCK | Categorical | - | $O(N^2_{sample}\log N_{sample}+ N^2_{sample}+kN_{sample})$ | $O(minn^2, nm_m m_a)$ |

#### 2.3.1.2 Partitioning Clustering

In partitioning clustering algorithms, a data set is partitioned into k partitions with n objects within each partition using a predefined objective function. Minimizing square error function is as an objective function which is computed as shown in Equation 2.9.

$$E = \Sigma\Sigma\|p - m_i\|^2 \tag{2.9}$$

where $p$ is a data point in a cluster and $m_i$ is the mean of the cluster. As the centroid-based algorithm considers all possible partitions, this is not practical for large data sets due to its high computational complexity. Hierarchical clustering algorithms cannot undo

in their clustering phases, meaning that if two clusters are merged, it is not possible to obtain the two original clusters (existing before the merge operation) by splitting the merged cluster. Therefore, a few heuristic methods are used to deal with this issue, such as k-means and k-medoids. In partitioning algorithms, it is possible to move an object from one cluster to another to improve the clustering quality conversely hierarchical clustering algorithms. However, if a point is near to the center of another cluster, it maybe cause an overlapping problem. Here we summarize some of the well-known partitioning algorithms and we briefly explain how they deal with very large data sets.

**K-Means [38], [37]** is probably one of the best known partitioning algorithms. It divides data objects into k partitions in such a way that each object is assigned to the nearest cluster center. This operation is continued until all data objects have been visited; then the centroid is recalculated to achieve better clustering. The number of clusters (namely k), cluster initialization and distance metric are user-specified parameters, in which selection of $k$ is the most challengeable task. Therefore, K-means is a heuristic algorithm and is run several times to find better partitions with the smallest squared error since it aims to minimize the within-cluster sum of square. K-means is a greedy algorithm with time complexity of $O(TKN)$, where $N$ is the number of objects, $K$ is the number of clusters and $T$ is the number of iterations. $T$ and K can be ignored since they are negligible in comparison with $N$. Therefore, a K-means algorithm is scalable and suits large data sets because of its linear complexity. However, the numbers of clusters need to be defined in advance: K-means has limitations when dealing with outliers and discovering a non-convex cluster's shape.

K-means is also not suitable for categorical data and usually terminates at a local optimum. K-means utilizes the Euclidean distance so spherical clusters are found in this way. However, it is based on Mahalanobis distance to discover hyper-ellipsoidal clusters [39] with a higher computational cost. Several extensions of K-means were later proposed to deal various aspects, such as cluster size, merge and split operations. ISODATA (Iterative Self-Organizing Data Analysis Technique) and FORGY as proposed in [40] and [36] respectively are some examples which were proposed in the field of pattern recognition. In [40] and then [41] some changes in K-means are made in terms of type of clustering (hard, in which each object belongs to just one cluster, in contrast to soft, in which each object can belong to multiple clusters). This is called Fuzzy c-means. Another approach is proposed in [42] to make Fuzzy c-means and K-means faster through data reduction by replacing group examples with their centroids before clustering. Bisect-

ing K-means [43] is another example, which divides data recursively into two clusters at each phase.

Another interesting extension of K-means is presented in [44], which applies kd-tree to discover the closest cluster centers. In [45], x-means is proposed to define k using Akaike Informatio Criterion (AIC) or Bayesian Information Criterion (BIC). Finally, Kernel K-means [46] and Kmedoid [47] are two other extensions of K-means.

**CLARA (Clustering LARge Applications) [47]** deals with the deficiencies of the above clustering algorithms using PAM (Partitioning Around Medoid) algorithm [47], which has a time complexity of $O(k(n-k)^2)$, where k is the number of medoid objects and n is the number of non-medoid objects. Despite the attempt to fix the limitations of clustering algorithms, PAM is not an appropriate algorithm to be used with large data sets because of its time complexity. PAM is a medoid-based clustering algorithm. Medoid is a data point located roughly in the center of a cluster. PAM starts by finding k medoids randomly as representatives of each cluster and form $k$ clusters. Then through the use of a brute force approach, it finds the best k medoids between all pairs of the entire data set to perfectly cluster $k$ partitions. Obviously this is the reason for its high complexity. CLARA benefits from the PAM algorithm by applying it to a random sample of the data set instead of the whole set. CLARA takes multiple samples from the data set and then applies PAM on each sample to find the best k medoid among the sampled data. After that, CLARA attempts to discover the most similar data points to each k medoid from the entire data set to form k clusters. However, there is no guarantee that CLARA can find the best k medoids during the sampling process and also does not achieve the best clustering. As mentioned, the problem with PAM is that it stores all pair-wise distances between objects: this is space consuming and is not an option to apply to large data sets. However CLARA does not consider the whole dissimilarity matrix through sampling, which leads to achieving linear complexity in terms of time and space. So CLARA can be applied to large data sets.

**CLARANS (Clustering LargeApplications based upon Randomized Search) [48]** is an improved version of CLARA in terms of quality and scalability. This can be applied to large and high dimensional data sets since it uses a randomized search to cluster data points. CLARANS is also suitable to find polygon objects. The clustering process in CLARANS is similar to a search process in a graph. Each node in the graph is a representative of a set of k medoids. Two nodes are neighbors if their set of medoids

differs by one. The algorithm starts with a random node and max-neighbors are checked in a random way to find a better partition. If the neighbors provide a better partition, this process resumes with a new node; otherwise, the search stops by finding a local minimum. This iteration continues to find several local optimums, and the "best" local optimum is considered to be a clustering output. CLARANS and CLARA are similar in terms of sampling. However, there is a difference between them when choosing samples from a data set. Although CLARANS does sampling for a set of neighbors of a node and does not consider all neighbors of a node, it does not restrict a search to a localized area. This means that CLARA draws a sample from the whole data set and then works on the selected sample, while CALARANS draws a sample of neighbors and dynamically changes this sample and so works on all data sets not just on a particular sample of the entire data set. Since CLARANS considers the local area at each step, it can detect outliers more precisely than CLARA and is more resistant to dealing with increasing dimensionality.

Table 2.2: *Summaries of some of the characteristics of K-means, CLARA and CALARANS*

| **Algorithm** | **Type of data** | **Cluster shape** | **Time complexity** | **Space complexity** |
|---|---|---|---|---|
| K-means | Numerical | Spherical | $O(NKd)$ | $O(N+k)$ |
| CLARA | Numerical | Arbitrary | $O(k(40+k)^2+k(N-k))^+$ | - |
| CLARANS | Numerical | Arbitrary | Quadratic | - |

Table 2.2 summarises some features of K-medoids, CLARA and CLARANS in terms of complexity, data type, and cluster shape. These three partitional clustering algorithms can be applied to large numerical data sets. However, K-means is appropriate to find clusters of a spherical shape, while CLARA and CLARANS can find any arbitrary shape clusters. For clustering large data sets, CLARANS demonstrates better quality and efficiency than CLARA in discovering clusters, but fails to enable clustering in a very large data set because of its quadratic time complexity.

#### 2.3.1.3 Density-Based Clustering Algorithms

In Density-based algorithms, clusters are created based on highly dense areas over the remainder areas and the sparse areas are classified as noise or border areas. In this way, they can deal with outliers and non-convex shape clusters. Some of the most-used

density-based algorithms for large and high dimensional data sets are DBSCAN, DBCLASD, GDBSCAN, DENCLUE and OPTICS, which are briefly explained next.

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** [49] defines clusters using the concept of density reachability. Simply, a point q is directly density-reachable from a point p if this is not farther away than a given distance. eps and the minimum number of points (MinPts) are critical factors for DBSCAN to generate a cluster. This algorithm starts with a random or arbitrary point, and if sufficient neighbors are surrounded within the range of eps-neighborhood of a selected node, a cluster is then formed. Otherwise, the point is considered as noise. However, a rejected point (noise) may be reconsidered as a part of a cluster if it meets specific conditions. If a point is found to be in a dense part of a cluster, its eps-neighborhood is also part of that cluster. Hence, all points that are found within the eps-neighborhood are added, as are their own-neighborhoods when they are also dense. This process continues until the density-connected cluster is completely found. Then a new non-visited point is retrieved and processed, leading to the discovery of a further cluster or noise.

**DBCLASD(Distribution Based Clustering of Large Spatial Databases) [50]** is an incremental density-based clustering algorithm that uses a uniform distribution of data points in a cluster. Nearest neighbor distance is a key parameter through which clusters are formed. This algorithm builds clusters incrementally, meaning that it does not require the whole dataset to be loaded into the memory and it processes each data point on time. It is also called online clustering. Arbitrary shape clusters are discovered in this algorithm and it does not need any input parameter. Because of this, it is called an independent user-specified parameters algorithm. However, the problem is that it is an order-dependent algorithm.

**GDBSCAN (Generalized Density Based Spatial Clustering of Applications with Noise)** [51] is a generalized version of DBSCAN. Two definitions are changed in this algorithm. First, it changes the definition of neighborhood by a symmetric and reflexive binary predicate. This means that any binary predicate which is symmetric and reflexive can define a neighborhood such as intersect predicate to identify a neighborhood in a polygon. The second change, to obtain cardinality of a neighborhood, uses other measures such as non-spatial attributes instead of directly enumerating data objects of a neighborhood's object.

**DENCLUE (DENsity-based CLUstEring) [52]** is based on the idea that every data point has an impact within its neighborhood, which is mathematically modeled through influence function. In addition, the sum of influence functions are computed to obtain density attractors that are local maxima of the overall density function to define clusters. DENCLUE is robust against noise and outliers. It can handle arbitrary shape clusters in a high dimensional data set. It is faster than DBSCAN since it uses grid cells and just keeps information of grid cells in a tree-structure access. In spite of all these advantageous, DENCLUE needs to choose the density parameter and noise threshold carefully since they have a remarkable impact on the clustering quality.

**OPTICS (Ordering Points To Identify the Clustering Structure) [53]** is an algorithm for finding density-based clusters in spatial data. The rationale behind OPTICS is similar to DBSCAN, but it addresses one of DBSCAN's major weaknesses: the detection of meaningful clusters in a variable density data set. To do so, the points of the database are (linearly) ordered such that points which are spatially closest become neighbors in the ordering. Additionally, a special distance is stored for each point that represents the density that needs to be accepted for a cluster in order to have both points belong to the same cluster. This is represented as a dendogram.

Table 2.3: *A comparative study of the various clustering algorithms.*

| **Algorithm** | **Type of data** | **Cluster shape** | **Time complexity** |
|---|---|---|---|
| DBSCAN | Numerical | Arbitrary | $O(N \log N)$ |
| DBCLASD | - | Arbitrary | Roughly 3 times of DBSCAN |
| GDBSCAN | - | - | $O(n^* runtime of a neighborhood query)$ |
| DENCLUE | Numerical | Arbitrary | $O(N \log N)$ |
| OPTICS | Numerical | Arbitrary | $O(N \log N)$ |

Table 2.3 summarizes some the features of the above density-based clustering algorithms. As seen, they can be applied to large data sets including numerical data. All of them are capable of finding arbitrary-shaped clusters.

#### 2.3.1.4 Grid-Based Clustering Algorithms

Grid-based clustering algorithms generate a finite number of cells by quantizing data space and making a grid structure to perform the clustering process on it. Since these methods are dependent on the number of cells in each dimension and not on the number of data objects, their processing time is very fast. STING, CLIQUE, GRIDCLUS, Wave Cluster, FC and OptiGrid are examples of well-known grid-based clustering methods applicable in large and high dimensional data sets.

The main idea behind grid-based clustering methods is taken from [54] and [55] and can be summarized as follows:

1. "Creating a grid structure, i.e., partitioning the data space into a finite number of non-overlapping cells;
2. Calculating the cell density for each cell;
3. Sorting the cells according to their densities;
4. Identifying cluster centers, and;
5. Traversing of neighbor cells"

Also mentioned in [56], some of the main characteristics of the grid-based methods are as follows: (1) no distance computations; (2) clustering is performed on summarized data points; (3) shapes are limited to union of grid-cells, and (4) the complexity of the algorithm is usually $O($ number of populated-grid-cells).

- **STING** (Statistical Information Grid-based clustering) [57] decomposes the spatial data into rectangular cells and is represented by a hierarchical tree. Like BIRCH, STING makes data summaries in this way so statistical information such as mean, maximum and minimum values, standard variation and distribution type are stored in each cell. STING is a query-independent method since grid-cells store statistical information as summary information which is independent of the query. Incremental updating and parallelization are suitable for this grid structure. In addition, Time complexity of STING is $O(K)$, where K is the number of grid cells at the lowest level. Despite all advantage of STING, its performance depends on the granularity of the bottom layer of grid structure. Moreover, created clusters are enclosed horizontally or vertically, not diagonally, which affects the quality of clustering.

- **WAVECLUSTER** [58] is originated from signal processing. It transforms spatial data into a frequency domain to find a dense area in the frequency domain. In this way, different clusters with different resolutions and scales are obtained. The computational complexity of wavelet transformation is O (N), where N is the number of objects in the data space. WaveCluster can handle outliers and works very well with high dimensional spatial data. It is able to find arbitrary shape clusters. Moreover, it does not need to know the number of clusters in advance.

- **GRIDCLUS [59]** is used on the space surrounding the data values instead of the data by taking benefits from a multidimensional data grid. A neighbor search algorithm is applied to cluster blocks organizing patterns. This algorithm consists of five main steps: (1) insertion of points into the grid structure, (2) calculation of density indices, (3) sorting the blocks with respect to their density indices ,(4) identification of cluster centers, and (5) traversal of neighbor blocks.

- **FC [60]** is a self-similar clustering algorithm in which self-similarity is measured by applying the concept of fractal dimensions through the Hausdorff dimension. FC incrementally adds points to the cluster and after clustering, there is no radical change in the cluster's fractal dimension. Since the space is partitioned into the cells of a grid, it is counted as a grid-based clustering algorithm. FC scans the data once and it is a suitable clustering algorithm for large data sets and high dimensional ones. It can handle noise and can also discover clusters of arbitrary shapes.

- **OptiGrid [61]** uses a grid clustering algorithm that is applied to high dimensional data sets. It runs in the way that the whole data set is recursively partitioned into different subsets to find optimal grid partitioning. “Optimal” grid partitioning is achieved by finding a good cutting plane for each cluster recursively through a set of contracting projections.

- **CLIQUE** (Clustering In QUEst) is proposed in [62] in which subspaces of k-dimensional data set are defined to find their dense areas to present a cluster in k-dimensional data space. It identifies subspaces of a high dimensional data space to achieve better clustering than the original space. To find dense regions in a subspace, each dimension is divided into equal intervals. A dense area is found when the number of data points in this area exceeds a defined threshold. Also, a cluster in a subspace is a maximal set of connected dense units. Therefore, after

identifying subspaces containing clusters, it finds dense areas and connected dense areas in all subspaces of interest, and then through the MDL principle, clustering is terminated. CLIQUE automatically identifies subspaces; it is not sensitive to the size of input and the number of dimensions and can scale linearly. However, it is a simple method that causes loss accuracy in the clustering (as shown in Table 2.4).

Table 2.4: *Comparisons of time complexity and applicability of clustering algorithms for high dimensional data.*

| **Clustering Algorithms for large data sets** | **Capability to apply in high dimensional data** | **Time complexity** |
|---|---|---|
| Wave Cluster | No | $O(N)$ |
| STING | - | $O(number of cells at the bottom layer)$ |
| FC | Yes | $O(N)$ |
| CLIQUE | Yes | Linear with the number of objects and quadratic with the number of dimensions |
| OptiGrid | Yes | Between $O(Nd)$ and $O(N \log N)$ |

To conclude this section on clustering algorithms, the reader may refer to some other interesting studies [63], [64], [65]. This section, however, has focused on analyzing some specific algorithms that can be used for very large data sets.

### 2.3.2 Sampling

The definition of sampling taken from the Merriam Webster dictionary is, “the act, process or technique of selecting a representative part of a population for the purpose of determining parameters or characteristics of the whole population.” Based on this definition, sampling can be considered as a summarization technique that can reduce time and space by observing only a part of the whole data set but which remains informative, instead of considering the entire data set.

With the advent of digital technology, many data storage systems bear a huge volume of data that need to be processed and analyzed to meet users 'requirements. However, considering this huge amount of data demands a lot of time and cost. So in order to tackle these issues, sampling techniques have been widely applied in many research

areas such as data mining, data management, query optimization, approximate query answering, statistics estimation and data stream processing which meet the purposes of summarization. Therefore, before explaining different sampling techniques, for the reader to better understand sampling techniques, some preliminary descriptions are now provided.

- What is a sample? A sample is representative of a larger group (population) which preserves the same characteristics of the population and a study is conducted on the sample instead of on the whole population.
- What is population? Population is a large group of data from which the sample is taken for study.
- What is a frame? Sampling is performed on a special set of the population: this is called a frame.
- What is the aim of sampling? Generalization of an induction derived from a data collection (sample) to the population is the main goal of sampling.
- What is sampling error? Since statistical characteristics of a population are illustrated by a sample, one expects to encounter a sampling error, which is the difference between the sample and the population or, in the other words, from statistics and optimization study, statistical error is the difference between the observed value (sample) and the unobserved value (population).
- What is sampling bias? Sampling with unequal probability of being selected as individual data points (sample) from a data set (population) is indicated as bias sampling which is a non-random sampling.

With these explanations, let us consider the different sampling methods. There are various sampling techniques that meet different aims of a wide range of applications. However, most use the following steps to take samples from a data set.

- Define the population (N) to be sampled.
- Determine the sample size (n).
- Establish controls for bias and error.
- Select the sample.

There are, some factors that need to be considered when in selecting a sampling technique. These include degree of accuracy, research objectives, resources, time frame, knowledge of population, research scope and statistical analysis requirements.

This section reviews some of the primary sampling techniques and their extensions, looking back at their summarization aspect. To avoid confusion, we will use the terms data set and population interchangeably throughout this section. In general, sampling techniques are categorized into two main groups of probability-based and non-probability-based sampling. Sampling algorithms that give an equal chance of being selected to all data points are considered as probability or unbiased sampling. Biased sampling algorithms, on the other hand, consider data points with different probabilities of the literature on probability and non-probability sampling techniques, the most common methods of probability-based sampling are simple random sampling, systematic sampling, stratified sampling, and clustering sampling. The most common non-probability sampling techniques are accidental sampling, quota sampling,snowball and purposive sampling. All these techniques are described below.

#### 2.3.2.1 Probability Sampling

**Simple random sampling** [66] is one of the basic sampling techniques in which the probability of being chosen for each individual data point as a sample is as equal as other data points in the data set. Simply, every data point has an equal chance to be selected as a sample. The data points are numbered from 1 to n from which a sample including some random number is chosen. Simple random sampling can be performed in two ways: with and without replacement. In the former, every time a data is drawn from the data set, this is replaced to the data set and may be re-selected with the same probability in the next round. In the latter meaning, every data point can be selected only once: after being selected, it is removed from the data set and is not considered any more.

The advantage of random sampling is that it is really easy to perform with minimum insight from the data set in advance. However, it needs to have a list of all the population. There is a broad study of random sampling for various applications in the literature. We briefly review some of them in this section. In the context of large, very large or big data sets, analyzing and processing such large data takes time and sometimes it is not possible to store the whole data set such as a data stream. Therefore, to accelerate performance of these tasks, random sampling (which does not require pre-knowledge of data) can be helpful in this way to efficiently process and analyses data and be performed on a small part of the entire data (sample) which is still informative and accurate. An approximate

answer can be obtained more rapidly instead of considering the massive volume of data. Therefore, sampling can be considered as a good summarization technique for big data.

**Random Sampling with Reservoir [67]** solves the issue of selecting a sample size of m without replacement, randomly from a data set of size N (N elements), where N is not known in advance. It is an extension of random sampling that is one of the classical uniform schemes, and is also an infrastructure for many uniform sampling methods such as concise sampling, dynamic inverse sampling, chain sampling and distinct sampling. In the sampling algorithm, a reservoir maintains a fixed size, uniform and random sample of k that is drawn during a sequential pass through the data set. This means that the first n data points are added to a reservoir. Then, by arriving $n+1^th$ data point, one of the existing data points in the reservoir is randomly chosen to be deleted and therefore makes space for new data points in the reservoir, since the size of the reservoir is fixed and it is required to keep it constant. The unbiased reservoir random sampling is performed with average CPU time of $O(n(1+\log\frac{N}{n}))$.

In [68], the authors also proposed an online algorithm to choose a sequential random sample of n from a data set of size $N$ with minimum memory requirements. A sampling method is described [69] to summarize data traffic in vehicle-to-vehicle (V2V) space. Previous sampling methods (such as sliding Window, Reservoir Sampling and Exponentially-biased reservoir sampling) consider incoming traffic flows that are increasingly ordered based on data arrivals; therefore there was a limitation in this context. In fact, they investigated the case the data traffic in disorder because of transmission delays and multiple sources. They extended the early sampling method and effected some changes to make them compatible with disordered data streams. They also proposed another sampling method, called Polynomially Biased Reservoir Sampling (PBRS), which is applicable for multi-dimensional sampling tasks. In this way, a huge volume of traffic data can be summarized to be considered as a data stream and used this summary to predict upcoming traffic data and its conditions.

The reservoir sampling method is improved within DSS (distance-based sampling) algorithm [70] for transactional data stream to cover the deficiency of low performance of the reservoir in dealing with a small sample. They enhanced accuracy of reservoir sampling by using and comparing Euclidean distance function and re-ranking steps in an arriving new transaction to decide whether or not to include the sample.

**Acceptance/Rejective sampling** is based on the Bernoulli design, where every data point can be included in the sample. It is subjected to an independent Bernoulli trial that has an outcome that could be Success (1) or Failure (0). Every data point can be randomly included in the sample. Therefore, if the probability of success is shown by p, the probability of failure is $q = 1 - p$. Given n independent Bernoulli trials, then the probability of $m$ success is mathematically shown as follows, which is called Binomial distribution.

$$P(m) = \binom{n}{m} P^m q^{(n-m)} \tag{2.10}$$

In Acceptance/Rejective sampling [71], a candidate is obtained where acceptance or rejection of candidate depends on meeting some user-specified conditions. If it meets these conditions, it is accepted for inclusion in the sample; otherwise, it is rejected and the next candidate will be selected in turn for assessment to meet the conditions.

**Chain sampling and priority sampling** are two extensions of reservoir sampling [72]. The problem is how to select a sample from a moving window of recent data. The chain sampling deals with expired data in this way: a constant size sample of $k$ is taken from window size of W. Whenever new data $i$ arrives, its chance to be taken as sample is $\frac{1}{min(n,w)}$ and if so, then an index from domain of $(i+1, \cdots, i+n)$ is a candidate to be swapped with $i$, when $i^th$ data point is expired and this process of finding a substitute for a newly-arrived item is continued like a chain. This approach is applied on sequence-based windows with space complexity of $O(k \log n)$.
They also considered the case where the window size is not constant and is time stamp-based. A priority between 0 and 1 is allocated to newly-arrived data points and the highest priority will be chosen to be included in the sample. The space complexity of a priority algorithm is also not more than $O(k \log n)$ without any prior knowledge of size $n$.

**Biased Reservoir Sampling** Reservoir sampling is an unbiased sampling algorithm, where data points have equal chances of being selected as a sample. This unbiased sampling approach may have some deficiency in coping with evolving data streams. Indeed, after some time, parts of a sample may be less related and therefore become "useless" because of evolution of the data stream. Then, since recent history of evolving data streams are considered more than the rest of the data stream, the probability of their appearance in the sample should be changed and they do not have the same prob-

ability over the other parts of data streams for sampling. A biased reservoir sampling is proposed in [73] and employs a memory-less bias functions to use a replacement algorithm in the occurrence of stream evolution.

**Random Pairing** As mentioned earlier, the Reservoir sampling method cannot deal with expired data. This means that reservoir sampling can handle only updates and insertions, but not deletions. To deal with this issue, a new sampling method is proposed in [74], called Random Pairing (RP), to cope with deletions in a data set with stable size. In this way, they add new data points to the sample to keep the size of sample constant when a deletion occurs in the sample. They also considered growing data sets whose size increase over time and proposed a resizing algorithm to control the samples growth over time.

**Concise and Counting Sampling** These are uniform random sampling algorithms [75]. Concise sampling is similar to the reservoir sampling having this difference: the values which appear frequently in the sample are displayed as a couple of <value, count> to save more space. This approach inserts a new data point to the sample with a probability of $1/T$. If a newly-arrived item has been visited earlier in the sample, then the count increases. By exceeding predefined sample size bound, the new bound $T'$ is defined such that $T' > T$, and the deletion of each data point with $p(T/T')$ and insertion of subsequent data points to the sample with $p(1/T')$ is performed to achieve uniform sampling with lower overhead. Counting sampling [75] is an alteration of concise sampling with different treatment to deal with exceeding the predefined threshold of the sample size. It is more accurate than concise sampling. Later, Counting sampling was extended in [76] through applying a tracking counter as an estimator to count and discover the high frequency item set, sum and average and employing Bernoulli samples over evolving multisets.

**Weighted Random Sampling (WRS)** [77]. Unlike uniform random sampling, where each data point has an equal chance of being selected, data points in WRS do not have the same probability. Therefore, data points are weighted and they are selected based on their assigned weights. For example, WRS can be applied in a data stream that is considered big data, to take a sample from the recent data streams since it is based on weighting different parts of data streams. It is possible to choose a part of data streams which has high weight according to recent data streams. There are various extensions of

WRS in the literature, such as [78] and [79].

**Congressional sampling method** [38] is composed of biased and unbiased sampling techniques which are called senate and house respectively. Data are divided into groups and then a uniform sampling is performed on each group (house) and a biased sampling (senate) is applied on the entire data set and then the two taken samples are combined. They applied their proposed approach on group-by approximate query to enhance accuracy of group by query. Generating fast approximate answers to complex queries in very large data warehouses can be achieved through pre-computed summaries of samples, instead of considering the whole data warehouses that are very large and take too long to find answers.

#### 2.3.2.2 Systematic Sampling

Assume one wants to choose $n$ samples from a data set with $N$ data points. First, systematic sampling [66] computes an interval so that $K = N/n$, where $K$ is the size of the interval. Then a random starting point is selected. Thereafter, the starting point and $K^{th}$ data point from the starting point are picked as the first and second samples. This process continues to pick every $K^{th}$ data point based on the predefined interval until $n$ data points are selected as samples. For example, for a data set with 20 data points and 4 samples, the interval will be 20/4 = 5. Therefore, a starting point will be selected randomly from the first interval [1, 5]. Suppose that the third data point is chosen as the starting point. Thereafter, every $5^{th}$ data point is picked as a sample. So, 3, 8, 13 and 18 are chosen as the sample set. It should be noted that inappropriate selection of intervals may cause that some patterns in the data to remain hidden.

An advantage of systematic sampling method is in the simplicity of sample selection as well as its accuracy in comparison with random sampling. However, the chance of being chosen for all data points is not equal and depends on the starting point and interval. Systematic sampling can prepare enough samples if there is no pattern in the data. It is a good option for web query analysis where fixed interval sampling is performed. Some studies have concentrated their efforts to improve systematic sampling. Linear systematic sampling in [80] and circular systematic sampling in [81] are also considered. In [82], a modified, balanced circular systematic sampling when $N = nk$ is suggested. Another modification of systematic sampling, called FCFSSS (First Come First Served following Systematic Sampling) [83], is proposed. This works based on conventional systematic sampling with the difference being that it takes than one sample each time.

#### 2.3.2.3 Stratified Sampling

Stratified random sampling [66] divides the population into L non-overlapping sub-populations called strata. A sample is then taken from each stratum individually and if random sampling is employed in drawing the sample from each stratum, the method is called stratified random sampling. The size of a sample is usually computed in different ways for each stratum. One way is proportional to the size of the stratum, called optimal allocation, aiming to maximize precision with minimum cost.

$$n_s = n^*[(N_S * \sigma_S / sqrt(c_s)]/[\Sigma(N_i * \sigma_i)/sqrt(c_i)] \tag{2.11}$$

where $n_s$ is the sample size for stratum $s$, $n$ is total sample size, $N_s$ is the population size for stratum $s$, $\sigma_s$ is the standard deviation of stratum $s$, and $c_s$ is the direct cost to sample an individual element from stratum $s$.

Another method is the Neyman allocation, where the size of the sample for each stratum is defined based on the stratum size and its standard deviation with the aim of maximizing precision with a given fixed sample size, which is defined as

$$n_s = n^*(N_S * \sigma_S)/[\Sigma(N_i * \sigma_i)] \tag{2.12}$$

where $n_s$ is the sample size for stratum $s$, $n$ is total sample size, $N_s$ is the population size for stratum $s$, $\sigma_s$ is the standard deviation of stratum $s$.

Although categorizing and identifying proper strata is not easy and analysing results is complicated in stratified sampling, it covers the population better than simple random sampling. Stratifying a sample is easy and helpful to better analyse data for each group with different characteristics. Accuracy of stratified sampling can be regarded. Stratified sampling has drawn many observations and has been extended in many studies. Some of these studies are cited accordingly.

- In a heterogeneous data stream, stratified sampling could be a good option to take a sample from every sub-stream with different statistical properties. In this way, a data stream is clustered as strata and then a random sample is taken from each obtained homogeneous cluster or strata. In [84], an adaptive size reservoir sampling method is proposed to regulate the size of reservoir since (as mentioned earlier) the size of reservoir in conventional reservoir sampling is constant. Therefore, they proposed the method to maintain the constant size of reservoir in some situations

where the size of reservoir varies. They then extended their proposed solution to adaptive multi-reservoir sampling.

- In [85], an adaptive stratified reservoir sampling (ASRS) is offered in which two issues of optimal size determination of sub-samples of each sub-stream and uniformity maintenance of each sub-sample is addressed. They considered the first issue by applying power allocation from [86] and for the second issue, they employed an alteration of their previously-proposed adaptive-size reservoir sampling technique [84].

- In [87], a sampling algorithm is illustrated and called strata which is based on the stratified sampling with the aim of drawing a sample to minimize the workload error and is applicable to approximately answering aggregate queries.

- A recursive stratified sampling method is presented in [88] to apply association rule and differential rule mining on the deep web. They draw a testable sample from the deep web to discover rules. Then, a learning phase is established to find data distribution and their relationship. At the end, the recursive stratified sampling is executed on the deep web. Their approach outperforms simple random sampling in terms of accuracy and cost.

#### 2.3.2.4 Clustering Sampling

In the cluster sampling method, the population is grouped into mutually exclusive and collectively exhaustive clusters, and later some clusters, not individual points, are picked through random sampling. There are two kinds of clustering sampling: single-stage and multi-stage. If clustering sampling is a single-stage, all data points of all selected clusters are seen as samples. In the case of multi-stage kind, a random sampling method is employed to select the data points from each chosen clusters in each stage. If clusters are similar, the sampling error will be reduced. If the clusters are very different, the sampling error gets larger so cluster sampling is not a suitable method in that situation. It is worth noting that in stratified sampling, an individual data point is drawn from each stratum as a sample but in cluster sampling, a cluster is selected and then treated as a sample. Stratified sampling aims to increase accuracy while cluster sampling aims to reduce costs with respect to boosting efficiency of sampling. The prominent advantage of cluster sampling method over other methods is that it is cheap but at the expense of a higher sampling error.

**Other Improvements and Applications of Probability Sampling Techniques** Many studies have been made within the scope of random sampling techniques with different applications, some of which are now briefly reviewed.

In [89], the random sampling method was improved by taking advantage of generating a decision tree from a data set. In fact, the decision tree presents a knowledgeable structure of a data set which may be very large. However, random sampling is a way to summarize the data set but it may not present the general picture of the whole data set very well through the samples taken. Therefore, they took advantage of both methods of random sampling and decision tree to present a better picture of the entire data set in a concise way. A method for online maintenance of an arbitrary sample size is investigated in [90], in which a sample is drawn through random sampling without replacement, by applying a suggested geometric file at the cost of $O(\omega \times \log|B|/|B|)$ random disk head movements for the newly sampled record. In [91], a sampling method is proposed to take a representative sample from a relational database considering data correlation. They named their new sampling method as CoDs (Chains of Dependencies-based sampling) through which a link of dependencies between data is extracted by considering foreign key constraints. They employed histograms in order to simply depict these relationships in distributed data, then they analyzed these discovered dependencies to take the sampling. In [92], a sampling framework for parallel data mining was suggested. They aimed to mine useful information from a large data base by finding frequent item sets and sequential patterns. To achieve their purpose, they employed a pattern-growth algorithm which is categorized as a divide-and-conquer algorithm since it projects and segments a data base based on discovered patterns. Then they tried to balance distribution of work load of mining tasks across processors. For parallel data mining, they needed to estimate time mining of different tasks to achieve load balance, which they addressed by proposing a selective sampling. They tested their parallel mining algorithm on a selective sample to estimate the required time for each task and also to identify large items.Their proposed selective sampling takes a sample from frequent items set by casting off a fraction of the most and the least frequent items and not considering the last m ending items of each sequence and also infrequent ones. There are some studies on sampling for approximate query answering applications such as [93]. Other research considers maintenance of dynamic data streams such as [94]. Discovery of association rules through sampling has been investigated in many studies by [95], [96], [97] in which sampling approaches for database files are reviewed.

#### 2.3.2.5 Non-Probabilistic Sampling

A few approaches relating to non-probabilistic sampling are listed as follows.

- **Accidental sampling** [98], also called convenience or opportunity sampling, takes those data points from a data set as a sample that are more available or close to hand. Therefore, the sample taken through this approach might not be a good representative of the entire data set.
- **Quota sampling** [92] is a non-random sampling in which a data set or population is divided into mutually exclusive groups. Then, through a judgment, samples are drawn from each group satisfying a pre-determined proportion. In fact, quota sampling is the non-probability case of stratified sampling. Quota sampling can be evoked in some cases including when the time factor is more important than accuracy, when budget is limited or when a sampling frame is not accessible.
- In **purposive sampling** [66], samples are taken from a specified population. It means that the research focuses on sampling from a particular population. [99] is an example of applying purposive sampling in social networks for the purpose of recruitment.
- In **snowball sampling**, a data point or group of data points which are sampled provide more data points to be sampled. Snowball sampling is useful in data mining social networks, such as [100].

### 2.3.3 Compressive Summarization

Data compression represents data using fewer bits than are in the raw data itself, through which resource usage such as storage space or transmission capacity is reduced. This is categorized into lossless and lossy techniques. In the former, compression is achieved by removing statistical redundancy and original data is retrieved after decompression, while in lossy compression, compression is obtained by excluding inessential data and recovering original data is impossible. Since much redundant data and similar patterns in data can be extracted in compression, this is considered to be a summarization technique that can reduce the size of data and present a compact version that is still informative and accurate. In other words, compression reduces the data size to save space and communication costs, and to provide fast data transfer. Meanwhile, summarization tries to give a compact version of the entire data to be analyzed. Therefore, this

compact version can be obtained through compression techniques. Since compression is considered to be as a data reduction technique in the context of data mining, much research has focused on applying compression as a summarization technique in big data sets. Although there is a broad range of studies on data compression such as video coding, image coding, audio coding and text coding, we briefly consider the compression of event sequences based on Minimum Description Length (MDL) considering summarization aspects.

First, we consider the definition of MDL and then we review some studies focusing on compression of event sequences based on MDL.

**Minimum Description Length** (MDL) traces back to the Occam's Razor principle. This principle declares that among competing hypotheses, the hypothesis with the least number of assumptions should be picked, from which it follows that simplicity generally causes correctness. MDL [101] formalizes this principle principle so the best hypothesis for a given set of data is the one that can achieve the best compressed data. Rissanen [102] stated that the rationale behind MDL is finding regularities in the visited data and the prosperity of detecting these regularities is evaluated through the length with which the data can be explained. The MDL principle is "a relatively recent method for inductive inference. The fundamental idea behind the MDL principle is that any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols to describe the data literally." [103]. sequences are massively generated through monitoring systems and user activities

We shall next describe an event sequence and then explore some studies that have designed a compression method based on MDL for the event sequences. Compression of event sequences based on MDL is discussed in this chapter because event sequences are massively generated through monitoring systems and user activities such as network traffic and logging systems. To easily analyse these event sequences, a summarized version of this large volume of data can be considered.

**Event sequences** are produced by monitoring user/system activities such as logging systems and network traffic data. In order to handle and have a general picture of the entire system behavior, some research attempts focus on finding comprehensive and short summaries of the entire event sequence. Studies have presented a local picture of a system's behavior. In general, there are two points of view:

- local structure, and
- global structure.

It is also noted that in data analyses, event summaries have some properties such as brevity and accuracy, global statement of data, local pattern recognition, and parameter free.

A good solution to find short and accurate summaries based on the MDL principle is described in [104]. This meets the aforementioned properties. The authors proposed a summarization method, in which findings are an optimal segmentation and local models are derived from MDL principles, as an optimization problem. Each sequence is segmented into n intervals where events with similar frequencies are grouped together. Also, the probability of occurrence of different event types at each timestamp is independent of the probability of occurrence of other event types and their segment. This means that different event types can appear simultaneously. The authors tried to segment an interval of event sequence into contiguous and non-overlay intervals. After segmentation, a local model is computed as the one that can best describe data with fewer bits in each segment. Two dynamic programming solutions were applied to find the minimum total cost through a greedy algorithm, and the proposed segmentation method reduces the compression ratio and also achieves the minimum overall description length in polynomial time.

An extension of [104] is proposed in [105] by considering overlapping segments, segments separated by gaps and presenting an event summarizer tool. However this extended approach has some limitations. Firstly, the segmentation approach does not consider the relationship between different models, and this is not a good option for predicting future patterns. Moreover, it stores the same copies of models in an occurrence of long event sequences: having many duplicated models leads to a low compression ratio.

An event summarization method using MDL and the Hidden Markov Model (HMM) is given in [106]: this captures both the global and local view of a system. An HMM is learnt to portray the global relationships among the segments. An event sequence is divided into disjoint segments based on the frequency changes of the events and then modeled each segment in a way that overall description has being the minimum length. Two types of models were considered: independent ($M_{ind}$) and dependent ($M_{dep}$). In the former, each segment is isolated from other segments. Conversely in the latter, segments are correlated. Two different costs were considered to compute the final cost

of encoding an event sequence: (1) the cost of encoding segmentation, and (2) the cost of encoding event occurrences. The quality of summarization is evaluated through an objective function. The problem of this method is that only the intra-correlation among event types of adjacent segments is considered and the temporal information between event types in a segment is not taken into account. They thus benefitted from the concept of a machine state with the knowledge that system behavior within each state was stable.

To address the limitations of the methods suggested in [106], [107], Natural Event Summarization (NES) was designed where inter-arrival histograms are used to exploit pairwise temporal correlations among events. By applying disjoint histograms and using MDL to encode histograms, event sequences are summarized. The temporal patterns of the events, which can be periodic or correlation, are discovered through histograms and then by employing multi-resolution characteristics of wavelet transformation, the size of discovered histograms is shortened and the summary is visualized by an Event Relationship Network (ERN). However, there are different possibilities of drawing histograms to present correlation or periodic patterns between events, and employing MDL paves the way to choose the most suitable histograms to explain an event sequence. Inter-arrival histograms are applied to find correlations amongst event types. These histograms facilitate discovery of periodic and correlation patterns to describe temporal dynamics of event sequences. Therefore, in this way they depict a histogram graph presenting a relationship among event types. Then the histogram graph is encoded through finding the shortest path from it. The Dijkstra algorithm is used to find the shortest path in a polynomial time $O(|D|^2)$. To speed up the summarization process, the histogram graph is pruned by employing a wavelet transformation relying on multi-resolution analysis (MRA).

Contrasting with previous work, inter-arrival histograms are used as they can define various boundaries of the segmentations of different event types. Thus pattern discovery becomes more efficient and produces better summarization. After discovering patterns in a sequence, the summarization results are provided through ERN.

Pattern mining has attracted much attention in the field of data mining. Finding a small set of patterns is of particular concern as the characteristics of a large data base could be presented in a small set of patterns instead of the whole discovered patterns. This becomes an aspect of summarization-compression.

Two kinds of long pattern mining approaches are considered in the literature: maximal item sets [108], [109] and closed item sets [110]. The former is considered to be a lossy compression, while the latter is seen as a lossless compression. In [111], an

algorithm called $Clo\_episode$ is offered to pick a closed episode effectively through pruning methods and minimal occurrence. Furthermore, some studies benefit from the MDL principle to find short, beneficial and high quality sets of patterns to summarize and compress data. These are now briefly reviewed. A two-phase MDL-based code table mining approach is investigated in [112], called as KRIMP, in which a set of frequent items is discovered. Then a pattern is selected from this set to improve the compression ratio. In [113], a new version of KRIMP considered classification, where the issue of having large frequent items set is verified at a low threshold by using specific heuristic methods. A set of item sets, which compresses the database in a lossless and good manner, is mined. The value of compression is determined by employing the MDL principle. They achieved a set of frequent item sets with four orders of magnitude shorter than the entire frequent item sets. Experimental results are extended in [114] for evaluating other methods, and they proposed STREAMKRIMP in [115] as an extension of KRIMP to discover changes in data streams. An alternative pattern mining of [112] is proposed in [116] as one-pass approach, called SLIM. In contrast to KRIMP that finds the pattern set from a chosen set of candidates, SLIM finds the best pattern set from data. In [117], an event summarization algorithm is proposed in which serial episodes are discovered, and a set of patterns is mined instead of individual patterns. They used MDL to choose the best set of patterns that can describe a short and accurate summary of a database of events. Sequential data are encoded through a set of patterns and employ two algorithms: SQS-Candidates search (to choose an appropriate set of patterns from a set of candidates) and SQS-Search (to find the appropriate set of patterns from a database). They showed that event sequences can be summarized through finding a set of patterns which are short and non-redundant. Finally, GoKRIMP [118] improves KRIMP by using two heuristic methods to compress sequential patterns. There are other studies on mining compressing sequential patterns such as [119].

### 2.3.4 Wavelets

Generally speaking, these transformations are mathematical functions (that project Set X to Set Y) to make the “work” easier with the transformed set instead of the original one. There are different, well-known transformations, such as Fourier transform and wavelet transform that can be applied to a set of large data. In the context of summarization, a wavelet transformation is mostly used to transform data and then make it possible to construct a compact representation of the data in a transformed domain. As a wavelet transformation can truncate the wavelet transformed data through saving the strongest

wavelet coefficient and setting the other coefficients to zero, a compact version of data can be achieved. This section provides the required background on transformation techniques and then describes the wavelet transformations as a tool to build a summarized version of massive data sets to achieve fast approximate answers.

From signal processing, the purpose of transformation is access to information in signal that is not easily attainable from the original signal [120]. Many studies have focused on the application of various transformations in different fields. One of the most famous transformations is Fourier Transform (FT) [121]. FT can plot a frequency-amplitude curve, meaning that spectrum frequency of a signal can be observed in a signal by FT. However, FT is unable to present time when the spectrum frequency becomes visible in the signal. Therefore, FT is a good choice in applications where "time" is not an important factor, such as a stationary signal. A stationary signal is one whose frequency does not change over time. Conversely, a non-stationary signal is one in which the frequency is varied over time. Hence, other transformations through which time resolution of frequency is observable were required.

Wavelet transformation allows the time-frequency of a signal to be represented simultaneously. We first explain briefly what wavelet transformation is and then review some studies that have applied wavelet transformations in their works, considering the aspect of summarization for big data.

The definition of wavelet is a small wave and the wavelet transform is the process of converting a signal into a series of wavelets through which signals can be stored more efficiently than FT. As mentioned, FT only considers the frequency of signals, namely frequency content and its amount are shown through FT, whereas time-frequency of signals can be represented simultaneously by wavelet transforms. Therefore, a wavelet decomposition is considered to be another compression technique that can be used to create a summary of large data sets. In fact, wavelet decomposition is a mathematical tool that is widely used in compression fields especially for image compression.

The rationale behind a wavelet is that a data vector V is transformed to a numerically different vector of wavelet coefficients. The higher coefficients which have most compact energy will be retained and the others will be set to zero, thus achieving compressed data. In other words, wavelet transform provides a time-frequency representation of signal by decomposing a signal into a set of basis functions (wavelets) which are orthonormal. Wavelets are produced from a mother wavelet by dilation and shifting [120].

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\Psi(\frac{t-b}{a}) \tag{2.13}$$

where “a” denotes scaling parameter and “b” denotes shifting parameter.

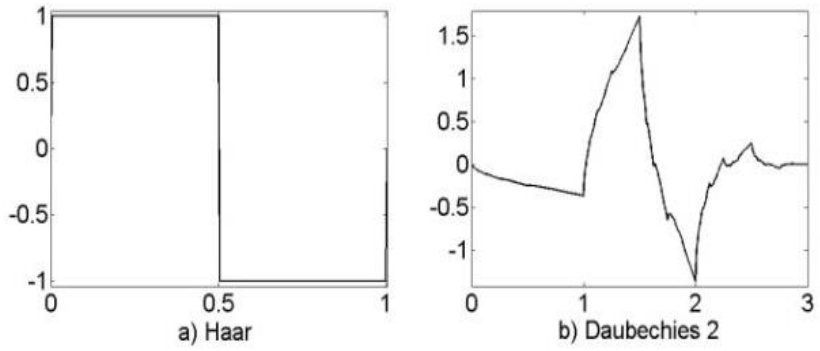



Figure 2.1: *Examples of Haar and Daubechies Wavelets (source [1])*

DWT is a wavelet transform which is mostly used in data mining applications. The properties of wavelet help data mining to present data efficiently. These properties could be considered such as hierarchical decomposition, multiresolution decomposition, vanishing moment, linear complexity, and decorrelated coefficients. The Haar wavelet [122] (1D and 2D), Daubechies [123] and multi-resolution transform are the most popular transforms derived from DWT as seen in Figure 2.1. DWT decomposes a signal with length of $L$ into high and low frequency parts by applying low and high pass filters and down and up sampling.

Simply speaking, a Haar wavelet also works in this manner that the first signal is halved and an average of each pair of samples is computed. Then the difference between the average and the sample is calculated and the first half is replaced with the average and the second half is replaced with the difference as detail coefficients. This process continues till full decomposition is achieved. For example suppose that we have a 1D signal as [9 7 3 5]. The Haar wavelet transform is calculated as follows.

The first half is (9 + 7)/2 = 8 and the second half is (3 + 5)/2 = 4 so that we have [8 4]. Then, (8 + 4)/2 = 6 and (8 - 4)/2 = 2 so we get [6 2]. On the other hand, we have (9 - 7)/2 = 1 and (3 - 5)/2 = -1. Therefore, the Haar wavelet decomposition of signal [9 7 3 5] is [6 2 1 -1], where 1 and -1 are detail coefficients in order to reconstruct the original signal.

Having explained wavelet transform, we now provide an overview of the major studies that take advantage of wavelets in data mining applications with the aim of summarization and making a data synopsis. We will not explain each method, but a

general view is given about the various techniques and their applications with the purpose of data reduction and data summarization.

In large scale decision support systems (DSS), query processing plays an important role. Sometimes, answers to queries do not need to be exact but approximate and rapid answers will satisfy a user 's requirements. Therefore, many studies have focused on proposing some data reduction mechanism to achieve compact sets of data to give approximate answers to the queries from these synopsis sets which results in achieving fast, approximate answers. Some of the proposed methods rely on wavelet based methods to attain these compact sets.

In [124], probabilistic wavelet decomposition is proposed to find precise approximate answers to queries. Since, approximate answers provided by wavelet decomposition differ widely, there is no guarantee that the obtained answer is accurate. Therefore, in contrast to conventional wavelet transform, each coefficient is allocated a probability that shows its importance to preserve it for reconstruction. In [125], a Haar-wavelet based histogram creates a synopsis of data to obtain accurate selectivity estimations for query optimization. In [126], optimality of the heuristic method in [125] is also demonstrated. A synopsis OLAP data cube is proposed in [127]: it applies multi-resolution wavelet decomposition. They retained a compact set of wavelet coefficients over a data cube for the approximate range sum queries considering space limitations. An extension of that work regarding approximate query answering through wavelet can be found in [128]. In [129], a general wavelet technique is presented to calculate a small space representation for data streams. In [130], a new method to create wavelet synopses is proposed, called hierarchically compressed wavelet synopses (HCWS). To build optimal HCWS, a dynamic programming algorithm is presented to minimize the sum squared error considering space limitations, and consequently increasing accuracy of the created synopses.

The Haar wavelet decomposition can be used to minimize mean squared error and other metrics such as relative errors in data value reconstruction. However, the main purpose of the Haar wavelet is to minimize mean square error. In [131], [132] the authors showed that these wavelet based synopsis approaches of different measures may reduce the accuracy of approximate answering. Thus, they presented an idea of extended wavelet coefficient and proposed new algorithms for creating extended wavelet considering storage limitations and multi measure data (sum square and relative error norms). An extension of this work can also be found in [133]. The study in [134] presents some algorithms to create unrestricted wavelet synopses to achieve an “optimal” solution.

A dynamic maintenance of wavelet-based histograms for data streams is considered in [135] because if underlying data distribution is changed then maintaining accuracy of histogram is not easy. Sampling and probabilistic counting are used in this approach.

In [136], the authors presented the first known streaming algorithms based on Group-Count Sketch (GCS) wavelet synopsis for both one and multi-dimensional data, satisfying polylogarithmic space usage, logarithmic update times and polylogarithmic query times for computing the top wavelet coefficients from the GCS. In [137], wavelet synopses are built for static and streaming massive data by using a greedy algorithm for maximum error metrics. U-HWT algorithm is suggested in [138] to deal with uncertain data streams through applying Haar wavelet decomposition. The accuracy of the proposed algorithm was demonstrated via experiments and it was shown that a compact uncertain data stream can approximate the raw data stream. There is another study about compact representation of uncertain time series through hierarchical wavelet decomposition in [139]. Also in [140], the authors considered the issue of constructing data summaries through wavelet histogram in Map-Reduce. Haar wavelet-based synopses on probabilistic data are investigated in [141] through applying dynamic programming. There are many more studies about constructing synopses through wavelet decomposition, and the reader can find some important ones in [142], [141].

### 2.3.5 Histograms

Histograms are a method used to represent a large volume of data in a compact manner so they can be considered to be a data reduction or summarization technique. In fact, data distribution can be shown in a synopsis structure through histograms. Mathematically speaking, a histogram is a function $x_i$ that represents how much data are within the disjoint ranges (bins/buckets) and represents the frequencies of data falling whitin these ranges. This function can be shown graphically. The function $x_i$ satisfies the following condition,

$$Y = \Sigma_{i=1}^{n} X_i \tag{2.14}$$

where $Y$ is the total data and $n$ is the total number of buckets or bins. Depending on the type of data attribute, histograms can be depicted. If an attribute is nominal, then a pole or vertical bar is displayed for each value of data. If the attribute is numerical, then data is divided into buckets in which they are disjoint subsets of data. In other words, data is divided into successive disjoint sub-ranges. For instance, a data attribute value

within a range of 5 - 45 can be partitioned into 8 equal sub-ranges, as shown in Figure 2.2.

Each sub-range is plotted with a bucket or bin in which the width of the bucket is the size of sub-range and the height of the bucket indicates the frequency of observed item within the sub-range.

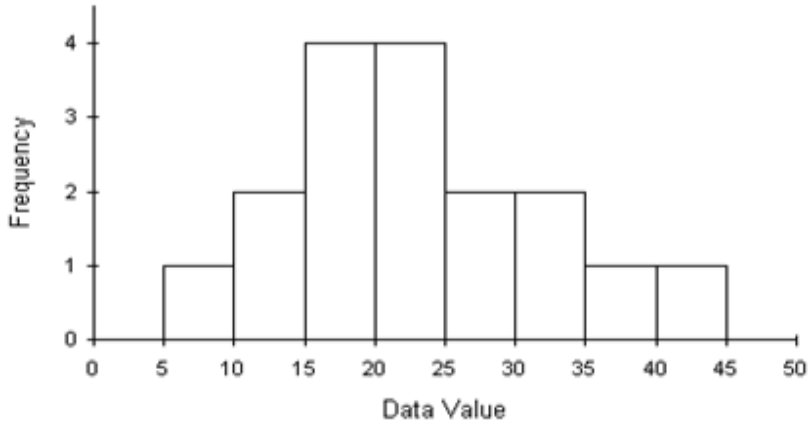



Figure 2.2: *An example of a Histogram based on frequency and data value*

There are different types of histograms. Some of the popular are categorized as follows.

- **Equi-sum** [138], also known as Equal-width histogram, categorizes continuous ranges of attribute values into $N$ equal intervals (buckets). The width of intervals is calculated based on the maximum (Max) and minimum (Min) values of the attribute as follows: $W = (Max - Min)/N$. Equal-width histograms have been employed in many commercial systems. However, they are not suitable for skewed data.

- **Equal-depth (frequency) histogram** , also known as an Equi-height histogram, is similar to Equal-width but with equal frequency in each bucket. In other words, the range is divided into $N$ intervals with approximately constant frequency for each bucket, This provides which is a good option for range queries with low skew data distribution but is not a proper option for commercial systems since bucket boundaries computation is expensive [143], [144].

- **V-optimal histogram** categorizes the continuous set of frequencies into a set of buckets to achieve minimum variance of the entire frequency approximation.

Simply speaking, V-optimal considers all types of histogram for a given number of buckets and picks the one with the least variance [145].

- **V-Optimal-End-Biased** histogram [145] groups the highest and lowest frequencies into individual buckets while the other frequencies are located in a single bucket. The advantage of V-optimal over Equi-depth and Equi-width is that it can give a better approximation of original data with fewer errors. However, updating a V-optimal histogram is not as easy as the others: sometimes it is necessary to change the whole histogram and rebuild it.
- In **MaxDiff histogram** [146], data are first sorted and then the margin of each bucket is computed considering adjacent values. The margins of buckets are determined where the difference between neighbor values is Maximum.
- **Spline histogram** [147] groups attribute values into contiguous buckets in which the width of the bucket can be varied. Data distribution in buckets is not uniform and is presented as a spline function instead of a flat value.

Note that these histogram methods are considered to be one-dimensional summarization techniques. There are also some multi-dimensional histograms, of which a few keep a one-dimensional histogram for each dimension based on the attribute value independence assumption (AVI) [148]. In others, the data is divided into d-dimensional buckets such as **GENHIST** [149]. Many studies have been conducted to apply histograms with the aim of fast approximate query answering, of which an example can be found in [150].

### 2.3.6 Micro-Clustering

Mining data streams has attracted much attention in recent years. Specific characteristics of data streams such as being infinite and in real time lead them to be processed as they arrive from different sources such as sensor networks and mobile devices. The clustering of data streams is studied in this separate section of summarization methods because micro-clustering techniques deal with real time summarization of data. One of the early works in this area is described in [151]. Many studies considered one-pass clustering over an entire data stream as not therefore being on user-defined time slices. Also, since a data stream is infinite, it is impossible to store the whole data streams because of memory limitations, so it would be beneficial to store a compact representation of data streams. Therefore, a two-phase micro-clustering algorithm was investigated in [151] for infinite and evolving data streams. The algorithm has two phases relating to online

and offline situations. The summary statistics of data are collected in an online phase and a clustering algorithm is then performed on these summary data. The proposed algorithm, called CluStream, enables micro-clusters to store summary statistics in a pyramidal time frame. The summary statistics are obtained as a temporal extension of the cluster feature vector of BIRCH [31]. They added timestamps to the feature vector as $(\overrightarrow{CF2^x}, \overrightarrow{CF1^x}, CF2^t, CF1^t, n)$,where $\overrightarrow{CF2^x}$ and $\overrightarrow{CF1^x}$ are the same as $SS$ and $LS$ in $CF$ in BIRCH and $CF2^t$ and $CF1^t$ are the sum of squares of timestamps $T_{i1} \cdots T_{in}$ and the sum of timestamps $T_{i_1} \cdots T_{i_n}$, respectively.

The pyramidal time frame is used to store micro-clusters that are captured at specific instants and are called snapshots in order to answer the queries of user over different time horizons. K-means is used to perform clustering in the offline mode.

After CluStream, other micro-clustering algorithms over data streams were proposed. Some of these studies considered micro-clustering frameworks based on a density feature. A density-based micro-clustering algorithm for a data stream, called DenStream, is proposed [152]. Like CluStream, this has two online and offline components. It made some changes in the concept of density that was used in DBSCAN by weighting areas of points in the neighborhood. The proposed algorithm can find arbitrary-shaped clusters and outliers by using p-micro-cluster, core-micro-cluster and outlier micro-cluster. They also applied a pruning strategy with the purpose of emerging new clusters.

Another instance of two phase components is investigated in [153] and they proposed D-Stream. A grid is built for each input data point in the online component. Then arbitrary-shaped clusters are formed based on the grid density in the offline component. In [154], SDStream was another online-offline framework based on CluStream. Since the framework focuses on the most recent data, so sliding windows model [155] is used. SDStream finds arbitrary-shape clusters as does Denstream. Therefore, they modified and used the core micro-cluster and outlier micro-cluster which are recorded as an Exponential Histogram of Cluster Feature (EHCF) in main memory. Micro-clusters are discovered and removed through the value of t in Temporal Cluster Feature (TCF). Clustering of discovered potential micro-clusters through DBSCAN in online mode is performed in offline mode. rDenStream is suggested in [156], considering the concept of outlier retrospect. It is a developed version of Denstream with three phases. rDenStream is a good option for applications with large numbers of outliers since it stores rejected outliers in an outside temporary memory in order to allow them to be included in the clustering process with the aim of increasing accuracy of clustering. This phase is called retrospect as a third phase of this algorithm. The other two phases are the same as

DenStream. It is obvious that by adding the third phase to process the historical buffer, the time complexity and memory usage will be increased in comparison with DenStream which are also demonstrated through experimental results. However, its performance is better than DenStream. In [157], C-DenStream is studied as a density-based clustering algorithm for a data stream based on DenStream. They suggested their algorithm based on the concept of static semi-supervised through and domain information in order to achieve highly satisfactory results. Still more studies related to clustering data streams are mentioned in the literature, such as AclueStream [158], OPClueStream [159] and ClusTree [160]. Two extensive surveys on clustering data stream can also be found in [161] and [162]. More detailed micro-clustering of data stream is available in [140].

## 2.4 Summary

In this chapter, we have described the concept of summarization. We also presented some of the important applications of summarization techniques to illustrate the urgent need for big data summarization in future. We provided an overview of some of the well-known summarization techniques that could be useful for big data. Specifically, clustering, sampling, compression, wavelets, histograms and micro-cluster were discussed in details.

In the next chapter, we investigate how compression-summarization techniques can be used in e-medical applications to extract patterns and summaries across a stream of images, where many similarities exist, to reduce storage and communication costs.

CHAPTER 3

# SUMMARIZATION OF TWO-DIMENSIONAL ARRAYS

Every day a large number of two-Dimensional arrays (i.e. matrices) is produced by graphical software such as Adobe photo-shop, video games, medical imaging, robotics and automation, economics and geology, to represent their correlated/uncorrelated data in a tabular format. Among all these applications, medical applications such as tele-medicine and tele-radiology are among the most significant sources of generating voluminous two-Dimensional arrays (e.g. digital images), and many challenges are associated with storage, retrieval and transmission of this amount of data due to the practical limitations in communication bandwidth and constraints on time and space. One way to tackle these limitations is to summarize data with the help of dimensionality reduction and compression techniques.

In this chapter, we investigate and propose solutions to summarize and compact these voluminous digital images, particularly x-ray images, to lessen the aforementioned limitations. We have proposed two new compression-summarization frameworks to more efficiently compact large medical images. Many similarities exist among a stream of medical images. We extract these similarities using some machine-learning techniques to improve performance of compression techniques as one of the methods of summarization. More importantly, in computer-assisted diagnoses, the loss of any part of the information contained in the data can be detrimental. Therefore, lossless image compression is the method of choice for our approaches.

## 3.1 Introduction

As medical imaging facilities move towards film-less imaging technology, conventional storage and transmission of large-scale raw medical image datasets can be very expensive and time-consuming. Lossless compression schemes play a crucial role in numerous medical applications especially with the growth of e-health services provided online. Among the most important applications are storage, retrieval and transmission of large volumes of medical data which should be handled fast, without distortion, and over band-limited channels. In this chapter, we start by showing that signicant amounts of correlation and redundancy exist across different medical images. Such inter-image correlations can be utilized to achieve better compression-summarization, and consequently less storage and less communication overhead on the network. We propose here a novel memory-assisted compression-summarization technique which can be used to complement any existing algorithm to further eliminate redundancies across images. We introduce the concept of learning-based coding by using source statistics to reduce redundancy of universal coders. In particular, we show the power of the proposed framework by combining simple Principal Component Analysis (PCA), as a learning stage, with existing lossless compression techniques. The proposed memory-assisted compression-summarization allows each image to be processed independently from other images, and hence allows individual image access and transmission. We targeted our work to medical digital images for the following reasons.

1. The vital and crucial role of e-health applications in life today.
2. A publicly available X-ray image data set.
3. A voluminous number of X-ray images as 2dimensional-arrays.

The goal of this chapter is, firstly, to develop practical machine learning algorithms that can first exploit both intra- and inter-image redundancies for a dictionary of medical images. Secondly, we require that the algorithms allow individual (random) access to images. In other words, when a retrieval request for an individual image is received, the proposed algorithms should be able to retrieve the requested image without decompressing the whole database. We argue that the compression problem discussed above can effectively be solved using the recently-developed “memory-assisted compression” technique developed by the authors [163]. As such we discuss the adaptation of well-known compression algorithms in the literature to this concept of memory-assisted compression-summarization framework, which can be formulated as a two

phase compression-summarization scheme. The first phase is learning, in which the algorithm runs over a subset of images in the database, and extracts the commonalities shared among all the images. Such information is stored and used in the next phase; the memory-assisted compression. We note that, in several applications, the subset of the images in the first phase is readily available. As one example, in telemedicine applications, where medical images are taken every day and transmitted over the network, the images from the previous day can serve as the memory for the next day.

Our focus here is mostly on the second phase, i.e. the memory-assisted compression. In the second phase, for compression of every image in the database, the common information stored in the memory as data summaries is used to eliminate the inter-image redundancy and only the residue is fed to traditional lossless compression algorithms. We reiterate here that the proposed two-phase structures enable individual access to all the images without the need to decompress the whole database and at the same time, all the dependencies present among the images are used for efficient compression. Indeed, we propose a novel memory-assisted compression technique, as a learning-based universal coding, which can be used to complement any existing algorithm to further eliminate redundancies across images. The approach is motivated by the fact that, in medical applications, massive amounts of correlated images from the same family are often available as training data for learning the dependencies and deriving appropriate reference models. Such models can then be used for compression of any new image from the same family.

We applied dimensionality reduction techniques, particularly Principal Component Analysis (PCA) and Non-negative Matrix Factorization (NMF) on a set of images from training data to form the required reference models. The proposed memory-assisted compression algorithms allow each image to be processed independently of other images, and hence allow individual image access and transmission.

## 3.2 Chapter Organization

The rest of this chapter is organized as follows. In Section 3.3, we briefly review some of the related works relevant to our approach. Sections 3.4 describes required preliminaries of our proposed algorithms. Section 3.5 describes the problem and introduces the basic extraction algorithms using two well-known dimensionality reduction algorithms from machine learning techniques, PCA and NMF. Section 3.6 discusses the proposed single-

level and multi-level algorithms. Then, Section 3.7 discusses our experimental results. Finally, Section 3.8 has some concluding remarks.

## 3.3 Related Work

Most of the literature on medical image compression focuses on reducing redundancy within a single image [164]. However, only a limited body of research has considered the possibility of extracting commonalities (similarities and correlation) among a set of images [165]. Indeed, some studies have shown that extracting cross-image redundancy can significantly improve performance of traditional lossless compression techniques [166]. In this section, we first review some of published research on lossless image compression based on individual medical images. Then we briefly review a few studies that focus on extracting redundancies across a set of images.

### 3.3.1 Redundancy extraction within a single image

MacMahon et al. [167] proposed a form of adaptive blockCosine Transform coding, in which considerable compression of digital radiographs with minimal degradation of image quality is allowed. Their results obtained for chest radiological images showed a compression ratio as high as 25:1. Ekstrand [168] presented a lossless compression algorithm based on Context Tree Weighting (CTW). The algorithm performs optimally in terms of redundancy for a wide range of data sources including medical gray scale images. The results show enhanced performance compared to JPEG, JBIG, and CALIC. Asraf et al. [169] proposed a novel hybrid lossy and lossless compression method using neural networks, vector quantization, and Huffman coding. The method was tested on CT (Computerized Tomography) images achieving a compression ratio of 5 to 10. A lossless medical image compression method was presented by Kil et al. [170]. Their method was based on redundancy analysis and segmentation of image into Variable Block Size (VBS) in order to extract similarities and smoothness of blocks. It was reported that the technique outperforms Huffman, JPEG-LL and lossless JPEG2000 by 10- 40%. Ghrare et al. [171] introduced a lossless image coding algorithm based on pixel redundancy reduction and using 2 matrices of gray-scale and binary. The algorithm achieved a maximum compression ratio of 4. Miaou et al. [172] proposed an image compression technique which combines JPEG-LS and interframe coding with motion vectors showing

a better outcome than JPEG-LS alone. They tested their algorithm with six capsule endoscope image sequences and improved the average compression by 13.3% and 26.3% over JPEG-LS and JPEG-2000, respectively.

### 3.3.2 Redundancy extraction across a stream of images

The compression algorithms that exploit correlation within a set of similar images are named Set Redundancy Compression (SRC) [166], and are categorized into four types:

1. Min-Max differential method (MMD)
2. Min-Max predictive method (MMP)
3. Centroid method (CM)
4. Multilevel centroid method (MCM)

In the MMD algorithm, two images are generated from a set of similar images to extract redundancy. One image is called maximum and the other called minimum. The former is created by searching the maximum pixel values among pixels of similar images in the same position. Correspondingly, the latter is created by searching for the minimum pixel values in the same position across the set of similar images. Then the original image is subtracted from min and max images and MMD scans the image in a raster order and saves the smallest difference value for each pixel position.

In the MMP method, max and min images are used for discretization of the original image. Min and max images define the smallest and largest values for all images in the set for every pixel in position $i$ as lower bound and upper bound. Then the interval between these two bounds is divided into $N$ levels. Every pixel in position $i$ of each image will be represented as a level $L_i$. MMP uses neighboring pixel levels to predict the value of a pixel $i$ since levels of neighboring pixels are mostly with high probability and are close to each other.

The CM method takes the average of all similar images in a set to calculate the difference image. The histogram of difference image should be like a Laplacian distribution where all pixel values are close to zero. The MCM method uses the same technique as CM but multiple times. At each level it calculates the average image of difference images from the previous level.

In summary, all SRC algorithms extract “interimage redundancy” [173] from a set of images, then compress the residues from the same set of images. However, our proposed

MAC techniques, in contrast, learn the common pattern (a.k.a. prototype or model images) from the training set, then use these to compress unseen images from the testing set. Therefore, the ability to memorize the commonalities and exploit these to compress unseen images is a unique feature of the proposed MAC techniques [164].

## 3.4 Preliminaries

In this section, we explain two machine-learning algorithms as techniques of dimensionality reduction, PCA and NMF, that we used in our framework to extract commonalities across a set of similar images.

### 3.4.1 Principal Component Analysis (PCA)

PCA [174] is a statistical approach used to find an orthogonal transformation to decorrelate random variables. The PCA technique has been extensively used in diverse signal and image processing applications. It was originally introduced as a dimension reduction technique. The technique starts with a set of observation vectors of dimension N. For images, the columns are concatenated into a large vector of size N (number of pixels). Let M be the number of observations in the training set:

$$\mathbf{X_i} = [\mathbf{p_1}, \cdots, \mathbf{p_N}]^{\mathbf{T}}, \mathbf{i} = \mathbf{1}, \cdots, \mathbf{M} \tag{3.1}$$

From these observations, the mean vector and covariance matrix are estimated:

$$\mathbf{m} = \frac{\mathbf{1}}{\mathbf{M}} \Sigma_{\mathbf{i}}^{\mathbf{M}} (\mathbf{x_i}) \tag{3.2}$$

$$\mathbf{C} = \frac{\mathbf{1}}{\mathbf{M}} \Sigma_{\mathbf{i}}^{\mathbf{M}} (\mathbf{x_i} - \mathbf{m})(\mathbf{x_i} - \mathbf{m})^{\mathbf{T}} \tag{3.3}$$

Let the mean-centered observations be represented by:

$$\mathbf{w_i} = \mathbf{x_i} - \mathbf{m}, i = 1, 2, \cdots, M. \tag{3.4}$$

The goal is to find a subspace whose basis vectors correspond to the maximum-variance directions in the orthogonal space.

Let W represent this linear transformation that maps the original N-dimensional space onto a K-dimensional feature subspace, where normally $K \ll N$. The columns of W are the eigenvectors, $e_i$, of the covariance matrix C. The eigenvectors are obtained using eigen-decomposition of C, $\lambda_i e_i = C e_i$ and $\lambda_i$ is the eigenvalue associated with $e_i$. For a given observation vector $x_i$, the transformation results in a new vector $y_i$ given by: $y_i = W^T(x_i - m)$. As the first few eigenvalues represent most energy in the data, we usually select K to be much smaller than N. The original observation vectors can then be reconstructed using the inverse transform. $\hat{x_i} = W_T + m$.

### 3.4.2 Non-negative Matrix Factorization (NMF)

Technically, NMF [175] is an unsupervised dimensionality reduction technique which identifies a set of non-negative components of an object and converts a data matrix into the product of two smaller matrices. In its simplest form, NMF works as follows.

Assuming that an image database is represented by matrix $\mathbf{V}_{n\times m}$, where each column is a vectorised image containing $n$ non-negative elements (i.e. pixel values), and $m$ is the number of images in the set. NMF factorizes $\mathbf{V}$ into two matrices $\mathbf{W}_{n\times r}$ and $\mathbf{H}_{r\times m}$, where $r$ is usually smaller than both $n$ and $m$. More formally:

$$\mathbf{V}_{ij} \approx (\mathbf{WH})_{ij} = \sum_{k=1}^{r} \mathbf{W}_{ik}\mathbf{H}_{kj}, \text{ subject to } \mathbf{W}, \mathbf{H} \geq 0 \tag{3.5}$$

where the columns of $\mathbf{W}$ are the basis images of size $n$, and each column of $\mathbf{H}$ is a coefficient vector representing one of the $m$ images.

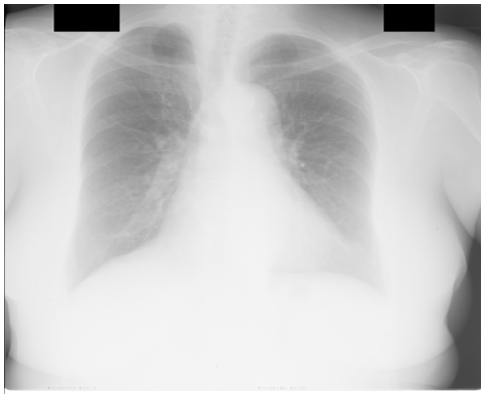
To calculate the basis and coefficient matrices, both $\mathbf{W}$ and $\mathbf{H}$ are traditionally initialized by random positive numbers. Then their elements are iteratively fine-tuned according to the following assignments:

$$\mathbf{W}_{ik} \leftarrow \mathbf{W}_{ik} \frac{(\mathbf{VH}^T)_{ik}}{(\mathbf{VHH}^T)_{ik}} \tag{3.6}$$

$$\mathbf{H}_{kj} \leftarrow \mathbf{H}_{kj} \frac{(\mathbf{W}^T\mathbf{V})_{kj}}{(\mathbf{W}^T\mathbf{WH})_{kj}} \tag{3.7}$$

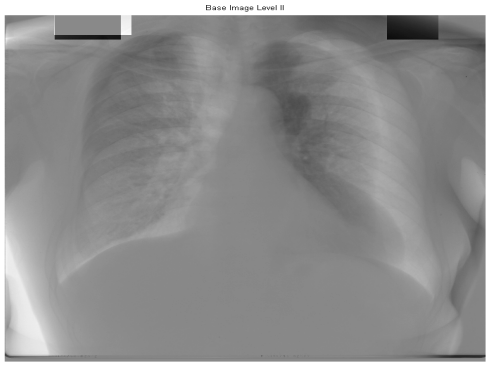
Note that any column of $\mathbf{W}$ can serve as a basis image. In this study, we only use the first column as the template image reflecting the similarities of all images in the training set. Actually, similarly to PCA, the energy contained in the first basis image represents a large percentage of the total energy. Assuming that the images of training and testing sets are strongly similar, as in Figures 3.1a and 3.1b, we expect that this first basis image can also capture most of the redundancy in the testing set (see Figures 3.1e and 3.1f).

(a) *Sample Image* $\mathbf{T}_1$



(b) *Sample Image* $\mathbf{T}_2$


(c) *PCA-driven* $\mathbf{M}^0$




(d) *PCA-driven* $\mathbf{M}^1$


(e) *NMF-driven* $\mathbf{M}^0$


(f) *NMF-driven* $\mathbf{M}^1$

Figure 3.1: *Random sample X-ray images (a, b), PCA-driven template images (c, d) and NMF-driven (e, f) template images.*

## 3.5 Problem Setup

The main rationale behind memory-assisted compression is learning source statistics at some intermediate entities, then leveraging the memorized context to reduce redundancy of the universal compression of finite length sequences. To the best of our knowledge, this is the first attempt to use cross-image correlation in medical image compression. Indeed, we propose to apply the concept of memorization with medical image sequences. The basic problem setup in a telemedicine application is displayed in Figure 3.2. The source contains a set of correlated medical images (e.g. Chest X-ray images) at the server node S (e.g. the central hospital) that need to be encoded and transmitted to the destination node (e.g. remote hospital) D through the network. We further assume that Hospital D has already memorized a database of previously transmitted images in its database which is also shared with S. In the absence of memorization, traditional compression techniques can still be applied to transmit the sequence of images from S to D. Here, we argue that the communication overhead to send the images from S to D can be substantially reduced if memorized context is available to the encoder and the decoder.

We present two different scenarios to compare results of traditional lossless image compression algorithms with our proposed memory-assisted compression algorithms.

1. First, we apply traditional but state-of-the-art logarithms to a set of medical images. In this scenario,redundancy is only considered within a single image for encoding and decoding it. The problem is that every single image is encoded without considering other images in the same set leading to a low compression ratio and additional overheads.

2. Second, we apply our memory-assisted compression algorithm using either PCA or NMF within the same set of images. In this case, the encoder and decoder have access to the memorized context for compression of new unknown images. We show that we can obtain a significant improvement in compression ratio for lossless medical images over state-of-the-art algorithms used in the previous studies.

## 3.6 The Memory-Assisted Compression Frameworks

In this section we discuss and describe our proposed single level and multiple level compression frameworks in further detail.

### 3.6.1 Single level memory-assisted framework

Consider a basic scenario in which a set of X-ray images is available at node S, and node D requests one of the images, as shown in Figure 3.2. This scenario can be an abstraction of a transmission problem or a storage and retrieval problem. Our benchmark is the case in which each image is compressed individually and sent to D. Then, at node D, each compressed image is decompressed independently. In the proposed method, the outcome of the learning phase, called M, of memory-assisted compression is available at both S and D. Then, using M, just the residuals of other test images are encoded at node S and decoded at node D. The proposed memory-assisted lossless compression method consists of two main phases :

1. Learning(memorization), and

2. Memory-assisted Compression (testing).

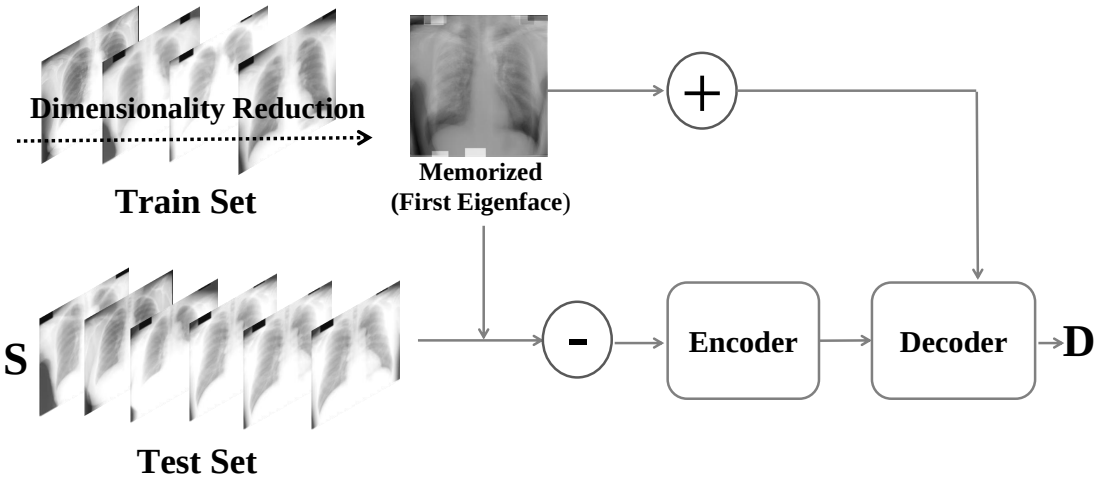



Figure 3.2: *The proposed single-level memory-assisted compression algorithm.*

The main question now is how can we consider and model the memorization concept from a set of gray-scale medical images?

The simple answer comes from the Karhunen Loeve transform (KLT) [176]. KLT is shown to be the optimal orthogonal transform through which the energy (information) contained in the signal is compacted. Indeed, with KLT, most energy is redistributed over a small number of components, simply called eigenimages. These eigenimages are obtained from the decomposition of the estimated covariance matrix. For our experiments,

we simply use the PCA transformation matrix as described in section 3.4.1 to decorrelate the images and remove inter-image redundancy.

For our experimental setup, the PCA procedure discussed above was first applied to the training set of images. Once the training stage is completed, we move to the coding stage. In this stage, test images are first projected over the PCA space. Using the reduced PCA space, the test images are reconstructed. These reconstructed images are close approximations of the original images. An error image is obtained by subtracting the reconstruction image from the original test image. It is simple to show that the pixel values of such an error image are uniformly distributed. As such, we can compress error images in an optimized way using the traditional CTW, JP-LS, CALIC and bzip2 algorithms. This means that we only need to send the feature vectors as well as the new compressed error images to the receiver. At the receiver, we first reconstruct the image projection then add to it the decompressed error image. To evaluate the performance of the proposed approach, we considered two scenarios and four generic compression techniques. The compression techniques considered are: the CTW, JPEG-LS, CALIC and bzip2 algorithms. The two scenarios are explained below.

- Scenario 1 (Comp): It denotes the case of using the CTW, bzip2, JPEG-LS, and CALIC algorithms directly on the test set.

- Scenario 2 (PCAComp): Here, PCA is applied to a train set of images. Then, for testing, the images are first projected and reconstructed using the PCA. Second, the residual images are encoded using the CTW, bzip2, JPEG-LS, and CALIC algorithms. At the receiver, the images are reconstructed using the decoded residuals added to the PCA-reconstructed images.

In the next section, we discuss how we extended our MAC framework [164] by improving its template extraction algorithm and adding an extra learning level. More specifically, we substituted Principal Component Analysis (PCA) with Non-negative Matrix Factorization (NMF). Although there are more constraints on NMF, the algorithm better captures the similarities and hence enhances the compression rate. We also extended our single level MAC framework to a multilevel framework. At the expense of a linear increase in computational complexity, the multilevel framework outperforms substantially its single level counterpart. We studied the sensitivity of the proposed technique to the sizes of the training and testing datasets as well.

### 3.6.2 Multilevel memory-assisted framework

In our previous MAC framework [164], we used a simple PCA algorithm [177] for determining the projection basis. Technically, PCA transforms the data from the original coordinate system into a new one such that the first coordinate (a.k.a. first component) contains the largest variance of the data, the second coordinate contains the second highest variation, etc. [177]. Despite its simplicity and effectiveness, a PCA application is sometimes limited by a number of assumptions, including:

1. Linearity in the combination of basis vectors;
2. Importance of directions with the largest variance,
3. Orthogonality of principal components

In addition, the eigenvalue decomposition of the covariance (or correlation) matrix for large size datasets (especially with large images) is usually a tedious task. Moreover, the performance of the decomposition may drop when the images are noisy or sparse (as in different biomedical imaging modalities).

To lessen some of the aforementioned difficulties, we propose a new MAC framework as shown in Figure 3.3 in which Non-negative Matrix Factorization (NMF) [165] is used for the template extraction phase. Technically, NMF is an unsupervised dimensionality reduction technique which identifies a set of non-negative components of an object and converts a data matrix into the product of two smaller matrices. Similarly to PCA, the energy contained in the first basis image represents a large percentage of the total energy. Assuming that the images of training and testing sets are strongly similar,as shown in Figures 3.1a and 3.1b, we expect that this first basis image can also capture most of the redundancy in the testing set (see Figures 3.1e and 3.1f).

A bold advantage of NMF over PCA is that it can be simply implemented using iterative algorithms. This means a balance (or a compromise) between accuracy of the approximation and speed of the algorithm can easily be achieved. Another advantage that makes NMF more applicable to our scenarios is that NMF can be seen as a parts-based representation. Unlike PCA, NMF representation is based on a positive-based combination of the basis images which can be seen as “true” template images.

The second framework introduces the concept of multilevel feature extraction to the MAC framework. Before explaining the details of the proposed multilevel framework, let us briefly review the existing (i.e. single-level) MAC algorithm, originally proposed in [164].

In a basic sender/receiver scenario, we assume that a set of "similar" images $\mathscr{T} = \{\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3, \cdots, \mathbf{T}_{|\mathscr{T}|}\}$ is available to the sender. This set of images can be used as the training set. The similarities within the training set are captured using an eigenvalue eigenvector decomposition of the estimated covariance matrix. Either the eigenimages of PCA (see [164]) or basis images of NMF (see Section 3.4.2) can be used to represent the whole training datasets.

Before any compression, the reconstructed image from PCA or NMF is subtracted from the images that need to be compressed (so-called testing images). More formally:

$$\mathbf{R}_i = \mathbf{I}_i - \mathbf{M} \tag{3.8}$$

where $\mathbf{R}_i$, $\mathbf{I}_i$ and $\mathbf{M}$ are the $i$-th residue image, the $i$-th image to be compressed, and the reconstructed template image, respectively. Now, the sender compresses the residue $\mathbf{R}_i$ (instead of the original image $\mathbf{I}_i$), using any arbitrary lossless compression tool. Then, it sends the compressed residue $\acute{\mathbf{R}}_i$ to the receiver. At the other end, the receiver needs to decompress each $\acute{\mathbf{R}}_i$ to retrieve corresponding $\mathbf{R}_i$, using the same lossless algorithm. Then the template image $\mathbf{M}$ is added to each $\mathbf{R}_i$ to reconstruct all the original images $\mathbf{I}_i$'s. Clearly, the more images to compress, the higher the improvement this method can achieve.

The main steps of a multilevel MAC framework are very similar to the components of the single-level MAC framework. The key difference between these two models is that in the multilevel framework, the residue images from the previous levels are treated as the next level "'original" images. In other words, assume that $\mathbf{M}^0$ is the reconstructed image learned from $\mathscr{T}$ (similar to $\mathbf{M}$ in single-level MAC). This means $\mathbf{V}$ in Equation 3.5 should be constructed based on $\mathscr{T}$. Then, the first set of residue images is defined as follows:

$$\mathbf{R}_i^1 = \mathbf{I}_i - \mathbf{M}^0 \tag{3.9}$$

At every level, the basis image $\mathbf{M}^j$ ($\forall j > 0$) is obtained from $\mathscr{R}^j = \{\mathbf{R}_1^j, \mathbf{R}_2^j, \cdots, \mathbf{R}_{|\mathscr{I}|}^j\}$, which is simply the set of all residue images computed at that level. This means for the calculation of $\mathbf{M}^j$, the $\mathscr{R}^j$ should be used to form $\mathbf{V}$ in Equation 3.5. The residue images at the $j$-th level ($j > 0$) can simply be computed as:

$$\mathbf{R}_i^j = \mathbf{R}_i^{j-1} - \mathbf{M}^{j-1} \tag{3.10}$$

In the multilevel MAC framework, the sender only compresses the residue images from the last level (i.e. $\mathbf{R}^j$). Then it sends these along with all template images (i.e., $\mathbf{M}^j$s). Figure 3.3 illustrates the different steps of the proposed multilevel MAC framework.

Note that adding any extra level yields to two contradicting objectives: On one hand, each extra level reduces the energy of the last residue images and improves the compression ratio. On the other hand, each level produces one more template image which should be transferred (once per set). Consequently, to achieve the best result, the number of levels should be carefully chosen.

In brief, the proposed NMF-based multilevel MAC encoder works as follows.

1. Learn similarities across a set of training images $\mathscr{T}$ by applying NMF. Store the resulting template into $\mathbf{M}^0$ (see Section 3.4.2).

2. Get the input test image set $\mathscr{I} = \{\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \cdots, \mathbf{I}_{|\mathscr{I}|}\}$.

3. Obtain residue-set $\mathscr{R}^1$ by subtracting $\mathbf{M}^1$ from $\mathscr{I}$ (see Equation 3.9).

4. Apply NMF on the residue-set from the previous level ($\mathscr{R}^{j-1}$) to obtain $\mathbf{M}^j$.

5. Calculate $\mathscr{R}^j$ according to Equation 3.10.

6. Repeat Steps 4 and 5 until final residue-set ($\mathscr{R}^l$) is computed. Then go to the next step.

7. Store $\mathscr{M} = \{\mathbf{M}^1, \mathbf{M}^2, \mathbf{M}^3, \cdots, \mathbf{M}^l\}$.

8. Apply any lossless compression algorithm on $\mathscr{R}^l$ to obtain $\acute{\mathscr{R}}$.

In order to reconstruct the original image set ($\mathscr{I}$), the following steps must be taken.

1. Apply a decompression algorithm, which matches the compression technique that is applied in Step (8), on $\acute{\mathscr{R}}$ to obtain $\mathscr{R}^l$.

2. Add $\mathbf{M}^j$ to each image of $\mathscr{R}^j$ to yield $\mathscr{R}^{j-1}$.

3. Repeat step 2 until $\mathscr{R}^1$ is calculated. Then go to the next step.

4. Add $\mathbf{M}^0$ to each image of $\mathscr{R}^1$ to obtain $\mathscr{I}$.

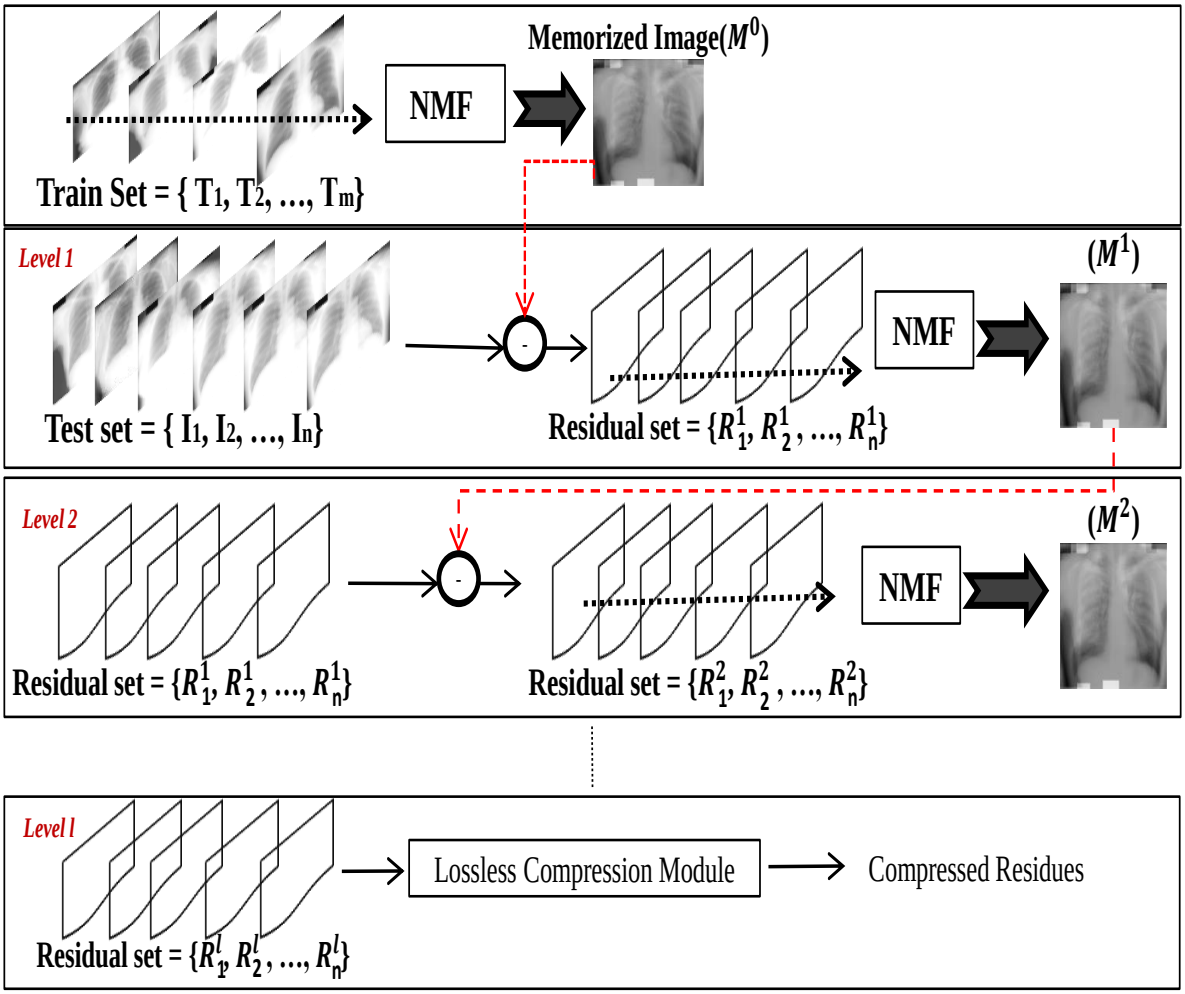



Figure 3.3: *The proposed Multilevel memory-assisted compression algorithm.*

## 3.7 Experimental Results

In this section, we will discuss our experimental results of the single level and multiple level frameworks respectively. Due to limited computation resources, we present results only for one and two levels (i.e. for multilevel frameworks we adopt only 2 levels) For lossy compression, we usually use the RMSE and a measure of performance, but for lossless coding, compression efficiency is usually measured using a compression ratio. Computational complexity is another factor that determines the efficiency of the method. This can be the number of CPU cycles, number of hardware gates, or memory bandwidth, etc. These are usually application dependent.

In our work, we focused mainly on the compression ratio defined as the compressed image size $S_{comp}$ over the original image size $S_o$, i.e.

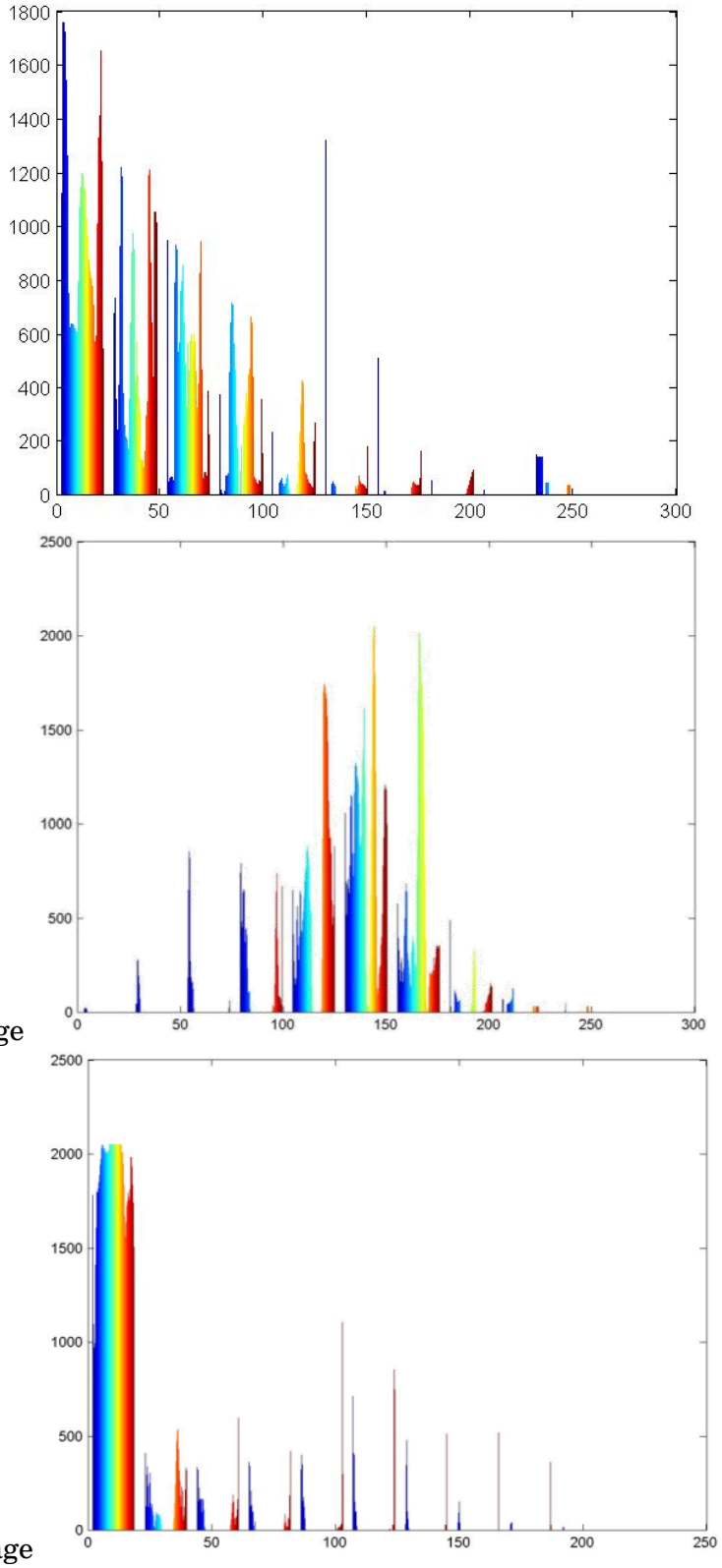
$$\mathbf{CR} = \frac{\mathbf{s_{comp}}}{\mathbf{S_o}} \tag{3.11}$$

The CR represents the number of bits the scheme uses to represent each bit in the uncompressed image. We calculated the CR for various compression schemes discussed in Section 3.6.1. We also examined the performance of single and multilevel PCA-based and NMF-based MAC methods. We also included the well-known Context Tree Weighting (CTW) [178] technique as the non-memory-assisted baseline in the experiments on multiple level frameworks. Since the superiority of the MAC technique (i.e. single level framework) over traditional lossless compression algorithms has been shown in the first set of our experiments in Figure 3.5, there was no need to compare the new MAC framework (i.e. multiple level framework) to the traditional methods. Note that in all MAC techniques, the same implementation of CTW is used to provide a fair comparison.

To study the sensitivity of the algorithms to the training and testing dataset sizes, three different sizes (i.e. 10, 20, 30) and (i.e. 5, 10, 15) were examined for both training and testing sets for single level framework, and further comparison of single and multiple level frameworks respectively. Therefore, each method is run nine times, in total. Note that for all cases, training and testing sets are completely exclusive.

The image database used is the JRST database which can be downloaded from [179]. It contains 154 nodule and non-nodule 8-bit Chest X-ray images with matrix size of 2048x2048 pixels. We selected a subset containing 20 Chest X-ray images. 10 images were selected as our training set while the other 10 images were used for testing the algorithm. The experiments were repeated 10 times by randomly changing the 20 images.

In the single level framework, the CTW, CALIC, bzip2 and JPEG-LS algorithms were applied on the test set as our benchmark lossless image compression algorithms. We display in Figure 3.4 the histograms of two typical images before and after applying PCA. The entropy for the original image (Figure 3.4 (a)) was 4.83 and decreased to 2.35 for the image after PCA decorrelation. We also show the histograms of the first eigen image and residue image (i.e. residue of raw image after performing PCA) as shown in Figures (3.4 (b)) and (3.4 (c)) respectively, which clearly proves the maximum variance that such an image can model.




(a) Raw image

(b)1st eigenimage

(c) Residual image

Figure 3.4: *Histograms of a raw image, first eigenimage, and single image after PCA*

Figure 3.5 displays a compression ratio for different lossless image compression algorithms using both the Comp and the PCAComp scenarios. The compression ratios (CR) for CALIC, CTW, bzip2, JPEG-LS were 0.18, 0.19, 0.24, 0.27, respectively. After applying our memory-assisted algorithm, the compression ratio was improved by 17.79%, 27.67%, 14.90% and 14.84%, respectively. As can be seen, similar trends are observed on the compression ratios of the CTW, CALIC and JPEG-LS algorithms. The CALIC, in particular, achieved a slightly better CR than the others.

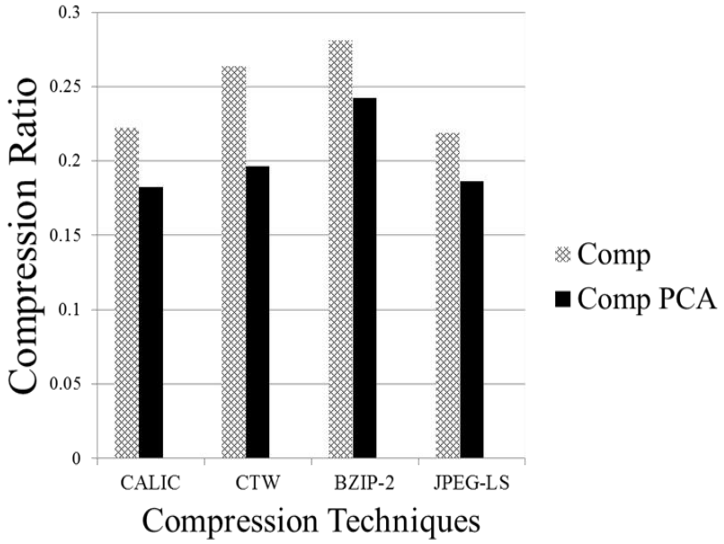



Figure 3.5: *Compression ratio (CR) for traditional and memory-assisted algorithms with different compression techniques.*

We also considered a number of other experiments on different sets of training images with various sizes for a single level framework. Figure (3.6) presents the average compression ratio on three different sets of training images including 10, 20 and 30 chest X-ray images. As can be seen, there is only a minor improvement in compression ratio when we increase the size of the training set (number of images) from 10 to 30 images.

The highest compression efficiency is achieved for the CALIC and the JPLS algorithms when used with the proposed PCAComp algorithm. By applying PCA on top of CALIC and JPEG-LS, we obtained gains of 16.97% and 14.55%, respectively. These reported gains are for training sets of 10 images. The same trend can be observed for the image set of 20 and 30.

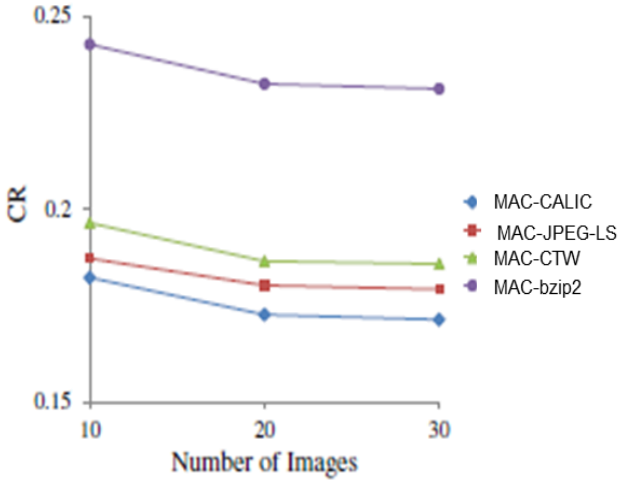



Figure 3.6: *Average compression ratio (CR) of memory-assisted compression algorithms for different training set sizes of 10, 20, and 30.*

To further analyze the energy compaction property of PCA, we display in Figure (3.7) the compression ratios (CR) of memory-assisted techniques for CTW and CALIC using a different number of eigenimages. As expected, few PCA components are indeed important in the reconstruction. Actually, our experiments showed that more than 97% of the total energy is contained in the first 5 eigenvalues (i.e. $\frac{\Sigma_{i=1}^{5}}{\Sigma_{i}^{M}\lambda_i} > 0.95$).

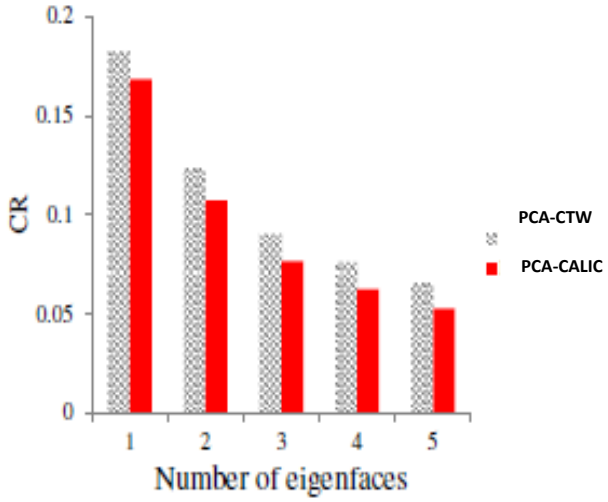
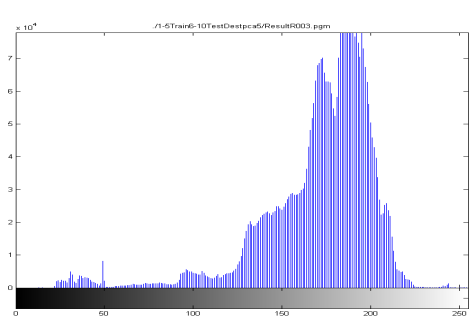
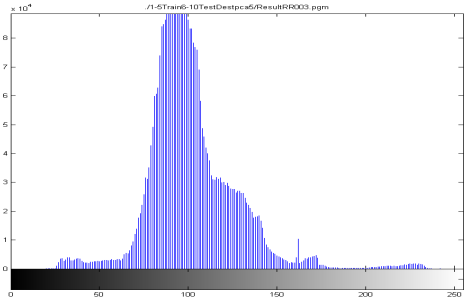


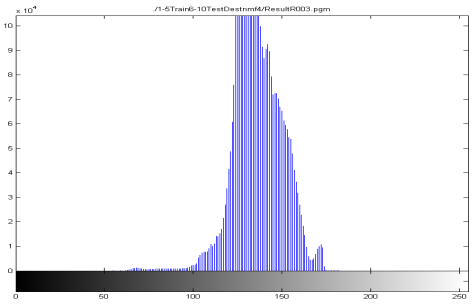

Figure 3.7: *Average compression ratio (CR) of memory-assisted compression algorithms for different number of eigenfaces.*



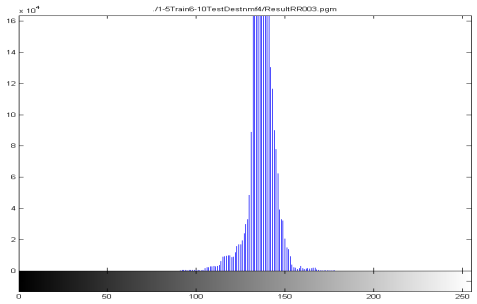
(a) *PCA-driven* $\mathbf{R}_1^1$



(b) *PCA-driven* $\mathbf{R}_1^2$



(c) *NMF-driven* $\mathbf{R}_1^1$



(d) *NMF-driven* $\mathbf{R}_1^2$

Figure 3.8: *Histograms of residue images of a randomly chosen X-ray image. Template images are computed either by single-level/two-level PCA (a,b) or by single-level/two-level NMF-based template extraction (c,d).*

We ran further experiments on the same set of data set to compare the results of a multi-level framework over a single level framework.

Figure 3.8 illustrates the histograms of the residue images of a randomly selected X-ray image. The residues in Figures 3.8a and 3.8b are computed using PCA, whilst Figures 3.8c and 3.8d are produced using NMF-driven templates. A comparison between the histograms reveals that both modifications (NMF and multilevel) narrow the distribution of pixel values and increase the number of pixels with zero value.

Table 3.1 shows the entropies of raw image, and image after PCA and NMF reconstruction with different levels. The table demonstrates a decreasing entropy trend of raw image after decorrelation by PCA, NMF for different levels. It can be clearly seen that the entropy of raw image is substantially decreased by NMF compared to PCA. This means the resulting residue images can be compressed much more efficiently.

Table 3.1: *Entropy Results*

| Entropy | PCA | NMF |
|---|---|---|
| Raw image | 6.7569 | 6.7569 |
| $\mathbf{M}^1$ | 6.2984 | 7.1897 |
| $\mathbf{M}^2$ | 6.1086 | 6.2888 |
| $\mathscr{R}^l$ | 6.1118 | 5.0064 |
| $\mathscr{R}^2$ | 5.6630 | 5.0050 |

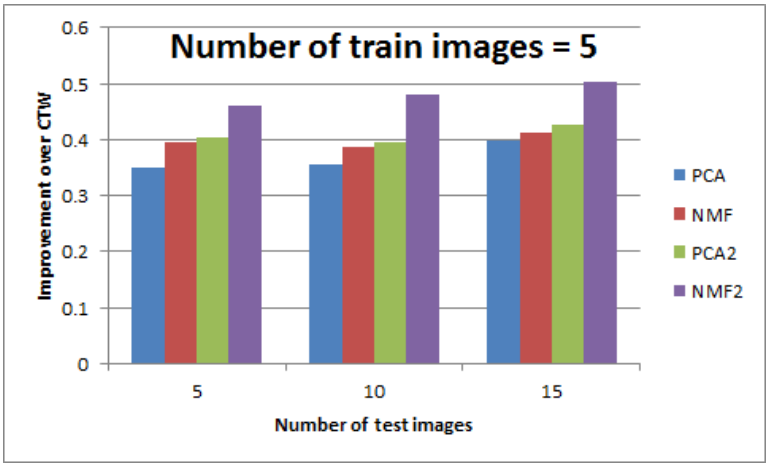
The experimental results of all algorithms with nine different training/testing set sizes are depicted in Figure 3.9. The bar charts compare the algorithms according to the compression ratio improvements over the memory-less CTW [178]. Each sub-figure depicts the relative improvement of the algorithms when applied on a fixed number of training images. In contrast, different groups of bars within a sub-figure show the performance of the algorithms when the number of test images varies. Note that in all figures, larger values for compression ratio improvement indicate better performance.

As Figure 3.9 confirms, all variants of MAC framework improve the no-memory-assisted baseline by at least 39%. Among all the variations, single-level PCA and multi-layer NMF (i.e. NMF 2) demonstrate the least and the most improvements, respectively. Indeed, the best performance among all variations is the multi-layer NMF-based algorithm with 15 train and 15 test images (62.25% improvement).

By comparing NMF variations with similar PCA ones, we see that NMF-based algorithms are the preferred algorithms. On the other hand, comparisons between single and two-level methods reveals that the multi-level learning significantly improves its single-level parent.

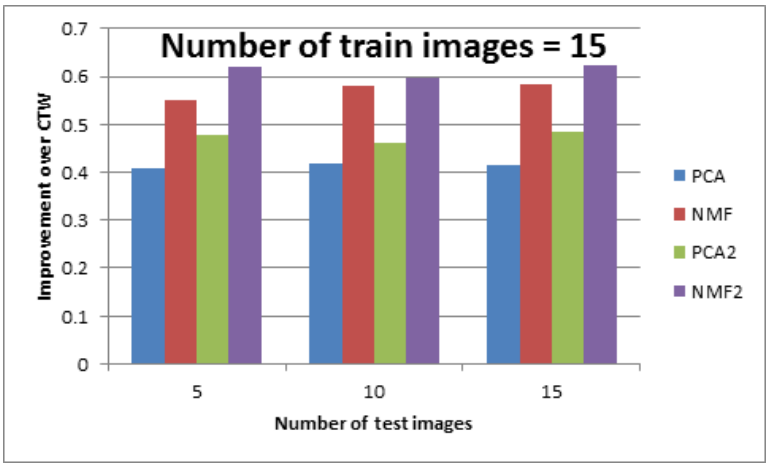
Another observation worth mentioning here is the fact that is when the number of training images is fixed, any increment in the number of testing images results in improvement without exception. This effect is more visible when the number of training images is moderate (see Figure 3.9.b). For small and large training sets (e.g., Figures 3.9.a and 3.9.c), the effect of the testing set size is superficial. In general, growth in the size of training set enhances the compression ratio. However, there is one exception. Comparing PCA-based methods (both one and two-level) in Sub-figures 3.9.a and 3.9.b shows that when the number of images in the testing set is not large enough, increasing the number of training samples from 5 to 10 has an adverse effect on the compression ratio. This phenomenon is unique to PCA-based MAC and is not seen in NMF-based techniques. Comparing Figure 3.9.b with Figure 3.9.c confirms that this surprising phenomenon is

just an exception and does not recur when the number of training samples increases from 10 to 15 images.




(a) $|\mathcal{T}| = 5$


(b) $|\mathcal{T}| = 10$




(c) $|\mathcal{T}| = 15$

Figure 3.9: *Comparing PCA, NMF, two-level PCA, and two-level NMF MACs to CTW.*

## 3.8 Summary

We discussed new methods for lossless compression using the concept of memory-assisted universal coding. The proposed approaches are well suited to compress large datasets of medical images especially for recurrent usage. The algorithms consist of a learning phase followed by a testing phase. In the learning phase, PCA or NMF is performed on training images to extract a set of eigenimages which are used to reconstruct the different test images. The reconstructed images are simply represented (coded) by low dimensional feature vectors. The error (or residual) images are then compressed using traditional lossless compression algorithms such as the CALIC, JPEG-LS, bzip2 and CTW algorithms. Our experimental results using the JRST database showed that the performance of traditional lossless algorithms can be improved by an average of 20% using the proposed algorithms. The proposed concept of using memory to enhance the performance of universal coders is expected to have a major impact in areas where images exhibit high correlation such as satellite images.

CHAPTER 4

# MULTIPLE DATA STREAM SUMMARIZATION

Advanced technologies, such as sensor networks, social networks and medical applications produce data streams. New data streams are continually being received and these can mean huge storage requirements with limited processing time. It means that as data is produced, it needs to be mined immediately in a single pass, to be able to answer client queries with minimum response times [180]. In addition, memory for storing arriving data is limited; hence data should be represented as a summary [181]. Even ignoring memory constraints, accessing and maintaining compact data is still a challenge. Having an appropriate data structure is a precondition to being able to accelerate the process of mining streaming data due to memory/disk limitations. For example, Figure 4.1 illustrates that maintaining data summaries in a tree data structure will minimize average and worst case times required for mining operations (e.g. searching for a proper cluster to insert a new data object). Moreover, data structures that can support dynamic updates as data arrives (insertions and deletions, e.g. insertion of arriving data objects from stream into the data structure) are preferred [2].

In the previous chapter, our focus was on covering "Data **V**olume" in big data by summarizing large medical images. In this chapter, the main focus is on "Data **V**elocity" as another characteristic of big data in both centralized and decentralized models by proposing summarization frameworks to deal with speed of multiple data streams.

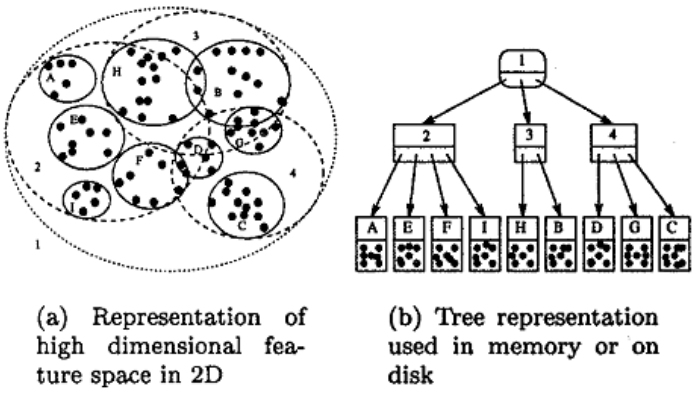



Figure 4.1: *The R-tree Family structure (Source: [2])*

## 4.1 Introduction

With advances in data collection and generation technologies, such as sensor networks, we now have environments that are equipped with distributed computing nodes that generate multiple streams rather than a single stream. Mining a single stream data is challenging, so mining multiple data streams becomes even more challenging.

Clustering is a well-known data mining and summarization technique and is a fundamental problem in many fields such as data mining, data bases and machine learning. It is a powerful tool to get insight into data and discover the structure of data by grouping a set of similar objects with each other. Some studies have focused on clustering multiple streams in a centralized fashion, while others have focused on clustering multiple streams in a distributed model [182], [183] as shown in Figure 4.2.

In many cases, collecting voluminous data from different nodes to a central location is impractical because it creates heavy traffic over limited bandwidth channels. Even ignoring bandwidth limitations and scalability, privacy issues may forbid gathering of the entire data set in a centralized third party. Environmental sensor networks are an example of this distributed setting in which centrally tracking data distribution over all sites is desirable.

Although many parallel and distributed clustering algorithms have been introduced for knowledge discovery in very large data bases [184], [185], scalability of data stream mining algorithms has reached its limitations. Therefore, development of more parallel (concurrent) and distributed mining algorithms is needed.

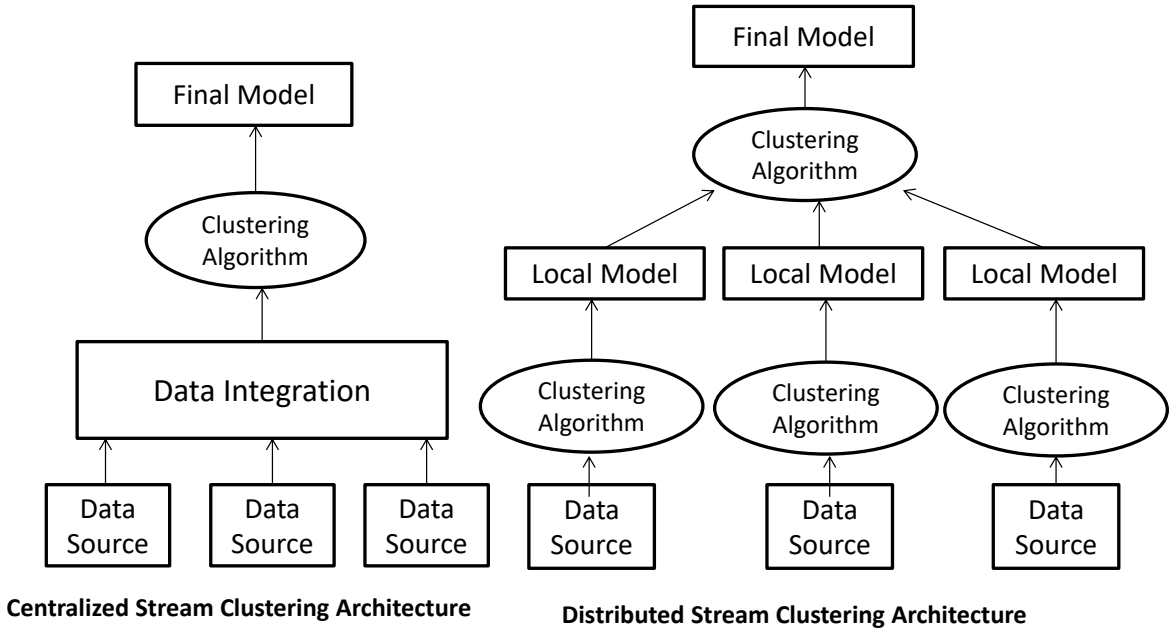



Figure 4.2: *Centralized architecture vs Distributed Architecture.*

Moreover, many classical clustering algorithms are designed for static data sets while ongoing clustering of massive data streams is highly desirable in modern distributed and continuous data acquisition systems. Thus, the dynamic nature gives rise to a new challenge; data should be able to find its path in the clustering structure and update clustering without restarting the process. If the distribution of new data eventually invalidates the previously computed clusters, the structure should reorganize. In the case of distributed clustering, updating master clustering and its reorganization is even more challenging.

There have not been enough studies on clustering multiple streams either in distributed or in centralized models with the focus on speeding up the process of clustering using appropriate data structure. Therefore, we have proposed two new frameworks using an index data structure from the R-tree family [186] to cluster multiple streams.

- Centralized multiple streams clustering framework, and
- Distributed multiple streams clustering framework.

In both frameworks, we have extended a clustering algorithm from single stream clustering to multiple stream clustering.

1. **Centralized multiple stream clustering**: We have extended the ClusTree algorithm [4] by replacing its data structure and access method from single access to multiple access and removing the buffer used as the anytime clustering feature. More specifically, this framework reports on our project's contribution to multiple data stream clustering in a centralized fashion.

    - Firstly, we substituted single access method with a multiple (concurrent) access method within the maintenance index data structure. Although there are some constraints on concurrent access to the index data structure, we believe that the concurrent clustering speeds up the process of clustering multiple streams, and creates more clusters at any given time.
    - Secondly, we reduced the space complexity of the ClusTree algorithm by removing its buffers at each entry.
    - Finally, we introduced a new framework to cluster multiple streams (distributed streams) which can also be used for very fast single streaming data.

    Through our improvements, we insert data objects to the proper micro-clusters in near real time, through concurrent access. Indeed, the anytime property of the ClusTree allows for interruption of the insertion process when a new data object arrives. The buffered data object should wait until a new data object arrives, then ride down the same subtree where the data object is buffered. Then, the new and the buffered data objects can descend the tree. Additionally, waiting at the buffer's entry may cause the buffered data object to become obsolete which consequently affects the quality of clustering. We also extract intra-correlation aspects from multiple streams through concurrent access to achieve high quality clusters.

2. **Distributed multiple streams clustering**: The focus is to reduce the limitations of monitoring dynamic distributed stream clustering as described above by introducing a new algorithm guaranteeing clustering almost as good as the best centralized clustering at any time with the main objective of reducing communication costs. More specifically, we keep track of the continuously evolving distribution

over all sites. Continuously tracking the underlying distribution of data leads to better understanding the nature of data over time. In essence we study the problem of tracking distributed clusters in evolving multi-dimensional data streams. Our algorithm monitors changes in the local sites and sends updates to a central site so that communication is only triggered between the central site and local sites whenever quality of central clustering is degrading. In this way, we provide guarantees on the continued quality of clustering. Although there have been few attempts to cluster continuous distributed data streams, none of them have considered the effect of using a proper data structure to store local summaries and as a factor of accelerating clustering of distributed streams and simplifying their maintenance (i.e. update of clustering). To the best of our knowledge, this is the first attempt to introduce a distributed multidimensional stream clustering algorithm using local index data structures to capture summaries over the entire data and reflects data distribution continuously with an adaptive precision error in the central site.
The manifold objectives of this study are to minimize the communication cost of clustering distributed data streams considering speed of processing data stream. Indeed, data streams cannot be stored or processed in their entirety in a central location and only a summarized form is stored. Data is summarized by means of micro-clustering techniques; such techniques allow local sites to capture as much information as their local storage permits in the form of cluster feature vectors and to store them in a local tree index structure.

In summary, our main contributions in the second study are as follows.

- We extend ClusTree into a distributed framework DistClusTree.
- We propose adaptable solutions to select local micro-cluster summaries to be sent to the central site. The central site then refines the quality of clustering according to real-time network traffic and the degree of privacy required.
- We model the monitoring and maintenance of distributed clustering as online tracking and extend a 1-to-1 multidimensional online tracking scheme into $m$-to-1 ($m$ local sites with one central site).
- We demonstrate through extensive experiments the performance of our framework in balancing communication costs and clustering quality.

## 4.2 Chapter Organization

The rest of this chapter is organized as follows. Section 4.3 describes related work. Section 4.4 describes required preliminaries of our proposed algorithms. Sections 4.5 and 4.6 introduce our proposed centralized and decentralized multiple streams clustering frameworks, respectively. Section 4.7 discusses our experimental results. We conclude the chapter in Section 4.8 with a summary of key discussions.

## 4.3 Related Work

We do not aim to review all stream clustering methods but focus on those that are relevant to our research. We divide relevant works on stream clustering into centralized and distributed clustering. We start with a brief introduction to centralized single stream clustering, continue with a few related studies on single stream clustering and finish with reference to a few distributed clustering algorithms.

### 4.3.1 Centralized Stream Clustering

Stream clustering approaches involve two phases: online and offline. The infinite properties of data stream and restricted memory make it impossible to store all incoming data. Therefore, summary statistics of data are collected at the online phase, and then a clustering algorithm is performed on obtained summaries at the offline phase. At the online phase, micro-clusters are created to group and store summary information with similar data locality, and these micro-clusters are accessed and maintained through a proper data structure. Micro-clusters are stored on a disk at a given time for a snapshot of data following a pyramidal time frame to be able to recall summary statistics from various time horizons. This provides further insights into the data through offline clustering.

Most of the literature on clustering single data stream focuses on extending and developing algorithms without considering the requirements of using a proper data structure to speed up the process of assigning continuously arriving data points to clusters.

BIRCH was introduced by [187] as a hierarchal clustering algorithm to handle very large static data sets. It introduced Cluster Feature Vector (*CFV*) as a compact representation of data which maintain in cluster feature tree (CFT). CFT allows the data set to be scanned once and so incrementally and dynamically cluster incoming multidimensional metric data points. BIRCH is used for clustering static data sets (whole

data set) rather than evolving data. It considered the importance of having a proper data structure to support efficient clustering of the data. ClueStream [188] introduced a micro-clustering technique and added some additional information to the BIRCH algorithm in order to adapt to continuously arriving data. This additional information is about time stamps of arriving data streams. DenStream [189] was proposed on a density based two-phase stream clustering algorithm. ClusTree [4] has been developed as the first, anytime single stream clustering algorithm in which a data structure from the R-tree family has been applied to maintain data summaries. That means it also considered the importance of having a proper data structure as BIRCH algorithm to support efficient clustering of the data. It was shown that the ClusTree performs better than other stream clustering algorithms [190], [191], both in terms of speed and in purity of clustering.

The main characteristic of ClusTree over other micro-clustering algorithms is its adaptability to the speed of streams in an online model. That is the reason we extended the idea of ClusTree, making it applicable to multiple streams in both centralized and decentralized models. Many of these two-phase clustering algorithms are reviewed in [192] in terms of number of parameters, cluster shape, data structure, window model and outlier detection.

### 4.3.2 Distributed Stream Clustering

All the aforementioned algorithms were designed and developed to cluster a single data stream in a centralized fashion. However, with the new generation of distributed data collection and multiple streams acquisition, it is desirable to introduce more parallel and distributed stream mining algorithms to tackle scalability and efficiency limitations of stream mining algorithms. Although many studies have been conducted on distributed clustering algorithms in very large, static data sets [193], [194], few studies have been reported on parallel and distributed stream clustering [183], [195], [196], [182], [197], [198], [199], [200].

The general idea of distributed clustering is that each local site clusters its own data. The local models are sent to a central site in either synchronous or asynchronous models; this central site keeps track of up-to-date clustering over entire local models. In [201], two types of continuous distributed clustering algorithms are proposed: local and global. The algorithm is formulated based on $k$-center clustering algorithm. The main objective in $k$-center is to minimize the maximum radius/diameter of the clusters. In their local solutions, each local site builds and keeps its local model, and only updates are sent to the central site. In their global algorithm, a global model is created iteratively in a central

site by message passing between local sites and the global site. Then the global model is sent to all local sites and in this way all sites have the same view of the clustering model. Local sites insert their data points into the global clustering and continuously send their updates to the central site. The central site decides whether it needs to recompute the global model and sends the new clusters to the local sites. As in most of the distributed clustering settings, distributed clustering results are compared with their centralized counterpart. A distributed extension of the Expectation Maximization (EM) algorithm called *CluDistream* has been proposed in [202]. The authors introduce a framework for clustering distributed data streams in the presence of noisy and incomplete data. The underlying distribution of data has been learnt by maximizing the likelihood of the data clusters. Local sites monitor the current model until they can not describe a new chunk of data. Then a clustering is performed to account for the changing data distribution. In this way, they reduce communication costs by just sending updates from local sites to the central site. In [203], a centralized density based clustering algorithm (DBSCAN) [204] was extended to the distributed model. Each local site performs a clustering on its own data and sends its representatives to the central site. Local representatives are grouped with each other to represent the final clustering in the global site. The work shows that the results of clustering in centralized and distributed model were significantly close to each other. A few more studies have been carried out on global or local distributed clustering algorithm, such as [205], [203]. Despite their underlying assumption to periodically trigger a communication in response to one-shot mining, they cannot satisfy the requirements of continuous stream mining [203]. Some other works have addressed clustering distributed data streams such as [205], [206] and a most recent survey on distributed clustering of ubiquitous data streams is presented in [176] that summarizes many of these methods. In [207], they approximated k-means clustering of distributed data streams by summarizing local clusters and merged summaries in a tree-topology way with bounded-error approximation.

None of the above algorithms enable either anytime concurrent clustering of multiple streams to speed up the process of clustering or extract intra-inter correlations of multiple streams to achieve high quality clusters. Nor do they consider the key role of using appropriate data structure to maintain summaries in distributed model. Therefore, we have proposed two new frameworks for clustering multiple data streams in centralized and decentralized models to lessen these research gaps. We review ClusTree in more detail in the next section since our proposed algorithms have been established based on the idea of ClusTree.

## 4.4 Preliminaries

In this section we explain in more detail ClusTree algorithm which is used in our frameworks to cluster multiple data streams and R-tree data structure in which local summaries are stored and maintained.

### 4.4.1 The ClusTree Algorithm

ClusTree [4] is an anytime stream clustering algorithm which groups similar data into the same cluster based on the micro-clustering technique.
Figure 4.3 presents an overview of the micro-clustering techniques in which, firstly, summaries of data are captured and maintained as micro-clusters and then representatives of micro-clusters are obtained to present broader views of clusters in data as *macro-clusters*.

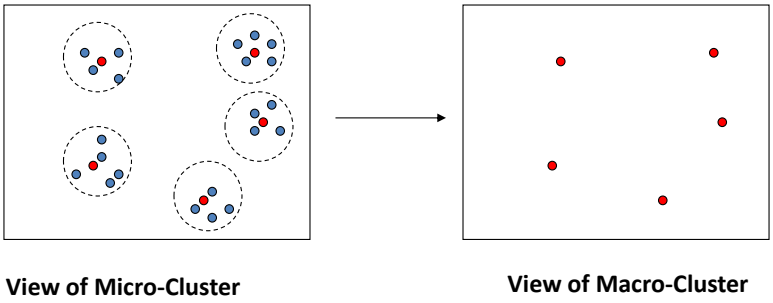



Figure 4.3: *Overview of micro-macro Clustering technique ( [3] ).*

ClusTree uses two phases:online and offline. In the online phase, it stores $N$; number of data objects, $LS$; their linear sum, and $SS$; their square sum in a cluster feature tuple $CF(N, LS, SS)$ as summary statistics of data. Storing these summaries instead of raw data helps to save space which is one of the challenges of stream processing algorithms. Clustree considers the age of the objects in order to give greater weighting to more recent data. The $CF$ tuples are enough to calculate mean, variance and other required parameters for clustering. Then an index data structure from the R-tree family is created to maintain $CF$s to speed up the process of accessing, inserting and updating micro-clusters. The idea is to build a hierarchy of micro-clusters at different levels of granularity.

Figure 4.4 illustrates at a high level the operation of the ClusTree. As shown, from time to time micro-clusters are transfered to a disk to be kept for further offline processing.

For example a data partition clustering like k-means or DBSCAN is performed on micro-clusters to form final clustering in an offline phase.The outputs of the offline phase are macro-clusters whose size is relatively small compared to the entire data stream.

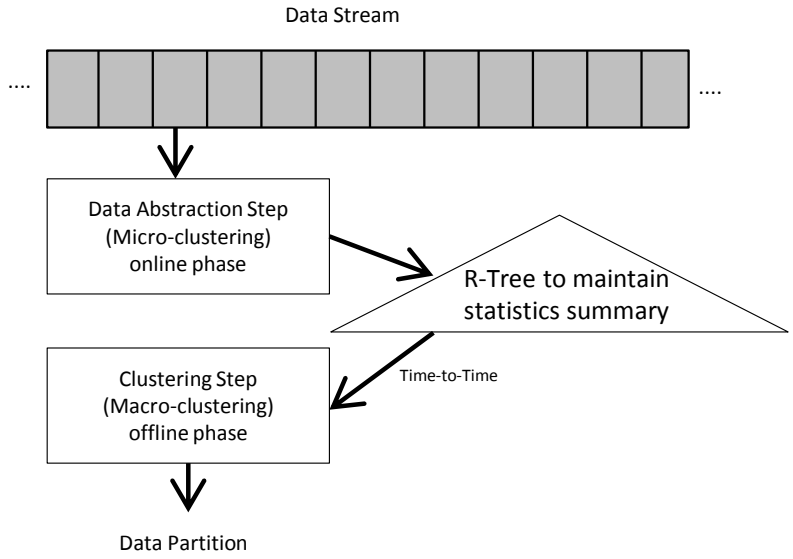



Figure 4.4: *A general schema of ClusTree algorithm*

In this way for an arriving data object, ClusTree descends the tree based on minimum distance between *CF*s and the arrived data object, to insert the data object into the closest micro-cluster at the leaf level within the given time. If a new data object arrives while the current data object has yet to reach the leaf level to be inserted to a proper micro-cluster within the given time, then its insertion process is interrupted. The interrupted object is left in the buffer of an inner node; the tree is descended to find a path for the new object. The buffered object has a chance to continue descending the hierarchy if it has not been outdated up until a new data object arrives, when its path to descend the tree is the same as the buffered object. Therefore, the buffered object descends the tree along with the new object as a hitchhiker to be inserted into the most similar micro-cluster at the leaf level. Using the buffer allows ClusTree to adapt with the speed of data stream to insert data objects into the micro-clusters at any given time. Moreover, ClusTree deals with high speed data streams by aggregating data objects at the top level of the tree, then inserting aggregated objects into the proper micro-clusters.

Figure 4.5 shows the inner entry and leaf entry in a ClusTree. Each entry in an inner node stores *CF* of objects and has a buffer to store *CF*s of interrupted objects which may

be empty. Additionally, each entry keeps a pointer to its child. Entries to each leaf node only store a *CF* of the object(s) it represents [4].

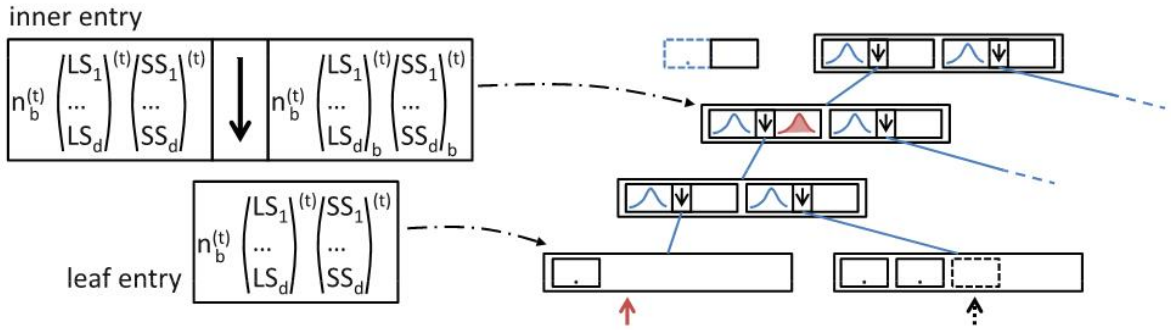



Figure 4.5: *Inner node and leaf node structure in ClusTree (Source: [4])*

Figure 4.6 shows the overall algorithmic scheme of the ClusTree algorithm. The micro-clusters are stored at particular moments in the stream, which are referred to as snapshots. The offline macro-clustering algorithm will use these finer level micro-clusters to create higher level clusters over specific time horizons.

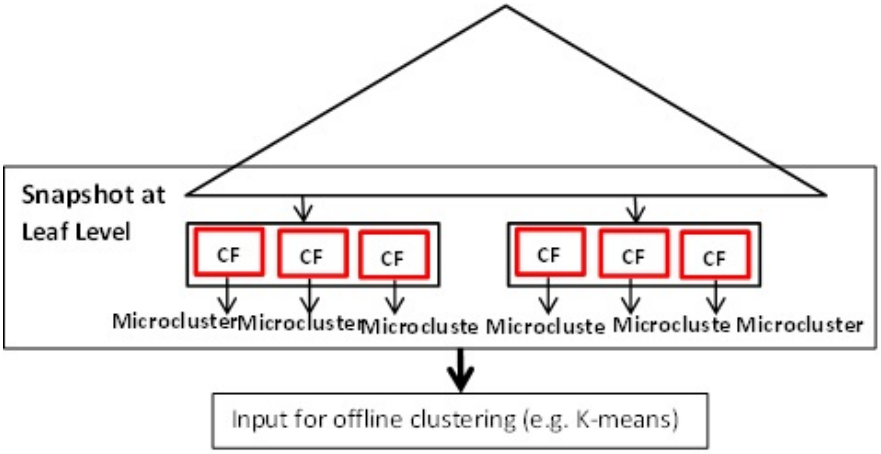



Figure 4.6: *A snapshot of micro-clusters in the ClusTree*

The ClusTree algorithm is proposed to cluster a single data stream with varying inter-arrival times. We have proposed two new algorithms based on the ClusTree to cluster multiple data streams concurrently and in a distributed manner. We explain our proposed algorithms in detail in the next section.

### 4.4.2 R-tree

R-tree is a tree data structure used for spatial data access. R-tree clusters multidimensional data based on their proximity. It represents nearby data objects with their minimum bounding boxes in different levels of the tree. The main goal of this data structure is to group adjacent data objects and represent them with their Minimum Bounding Rectangle (MBR) in the next higher level of the tree as shown in Figure 4.7. Since all data objects fall within this MBR, a query can be answered if it intersects this bounding rectangle. Aggregation of objects occurs at the higher levels of the tree while the root represents aggregation of all data objects.

From a clustering point of view, descending the tree reduces within-cluster sum of squares error. The within-cluster sum of squares measures the variability of the data points within each cluster. Generally, a cluster that has a small sum of squares is more compact than a cluster that has a large sum of squares. This translates to an increased purity of clustering at the leaf level where data objects are indexed. This can also be seen as an increasingly coarse approximation of the data set as we move to the top level of the tree.

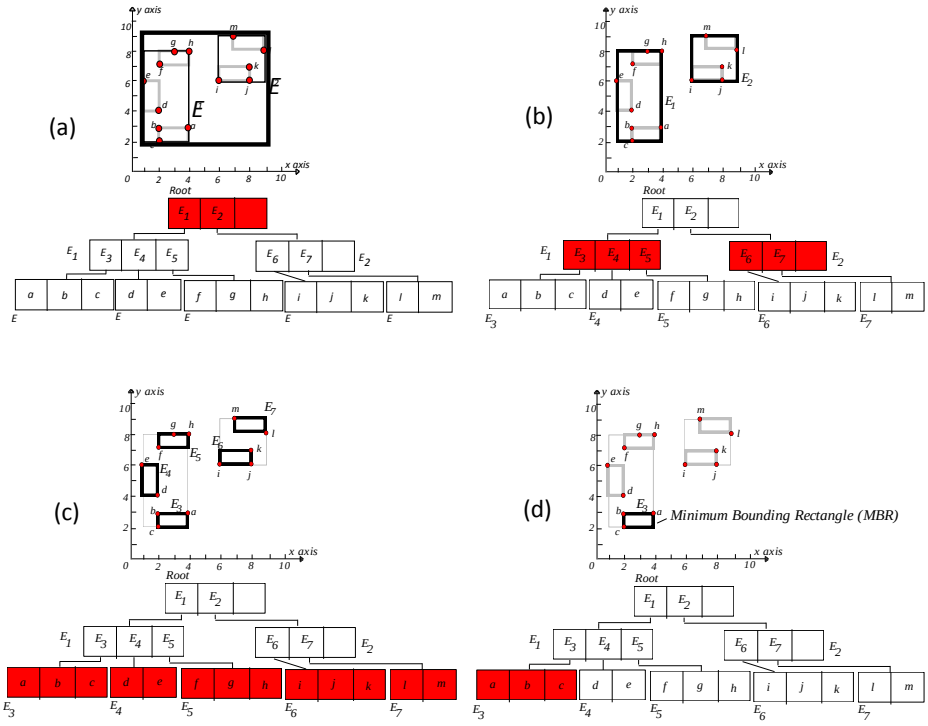



Figure 4.7: *R-tree: Data space partitioning by adjacency (source [5]).*

## 4.5 Centralized Multiple Streams Clustering

In this section we explain our proposed multi-stream clustering algorithm. First, we focus on the centralized multi-stream clustering problem and then we discuss the proposed concurrent clustering framework.

### 4.5.1 Anytime Concurrent Multi-Stream Clustering Framework

Data streams are continuously produced and need to be analyzed online. Moreover, multi-stream applications demand higher anytime requirements due to streams arriving at any time and with varying speeds. This continuously arriving data means huge storage requirements. Therefore, online multi-stream clustering is a twofold problem in terms of time and space complexity. Regarding space complexity, many studies (see Section 4.3.1) have been conducted to represent distribution of data in a compact way. The main idea is that instead of storing all incoming objects, summary statistics of data objects will be stored to reduce the storage problem. Many techniques are proposed in the literature to achieve summary of data. One of these techniques is called cluster feature vector [208] which we use in our proposed algorithm to obtain summaries of data objects. The other issue relates to accessing these summary statistics, which is crucial in terms of time complexity. Therefore, choosing a proper data structure plays an important role in maintaining and updating these summary statistics in memory. In fact, these summaries are generated and maintained in a proper data structure in real time, and then are stored on a disk for further and future analysis called offline processing. Hence, to achieve "extreme" anytime clustering, we propose to extend the ClusTree structure to a concurrent hierarchical tree structure to index more granular micro-clusters.

Figure 4.8 shows a general view of our proposed framework in which each stream is assigned to one processor. All the processors have equal access to a shared memory. The processors will create micro-clusters in memory in a parallel way through concurrency control. We expect to create more accurate micro-clusters with high granularity, in contrast to serialized clustering of a single stream using the ClusTree. Granularity is considered in terms of the number of micro-clusters. Intuitively, highly accurate clusters will be created by extracting correlations among different data streams through concurrent clustering. Similar data objects from different data streams have more chances to be grouped into the same cluster compared with local clustering of individual streams with a decentralized model.

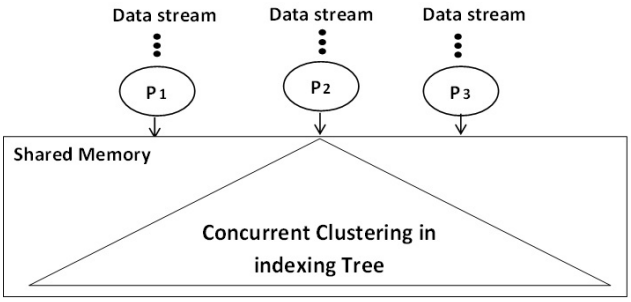



Figure 4.8: *Proposed concurrent clustering framework*

Like many optimization problems using a search tree to obtain an optimal solution, a tree of micro-clusters is created in which clustering data objects are started from the root. The children of the root are obtained by splitting the root into small clusters. The leaves of a tree represent micro-clusters in a given time interval. The goal of this paper is to insert data objects concurrently into their closest micro-clusters; optimal leaves, by using an index search tree. The cost of searching the tree and adding a data object to the closest cluster is $O(log(n))$, where $n$ is the number of elements in the tree. Using a parallel algorithm with concurrency control seems to increase the level of granularity and reduce the execution time of creating micro-clusters. To achieve this, each processor can explore a set of subtrees to reach proper micro-clusters. However, a tree is created during the exploration which means that subtrees are not assigned to each processor in advance. Each processor will get the unexplored nodes from a Global Data Structure (GDS) [209].

We propose to use a search tree that allows concurrent access to the GDS in the context of a parallel machine with shared memory in order to create and maintain high granularity micro-clusters. Each processor will process clustering operations on the GDS, stored in the shared memory. The main difficulty is to keep the GDS consistent, and to allow the maximum level of concurrency. In the shared memory model, the GDS is stored in the shared memory which can be accessed by each processor. The higher the access concurrency, the higher the granularity of clustering will be. The main issue is the contention access to the data structure. Mutual exclusion is performed to provide data consistency. We suggest using a concurrent index structure from the R-tree family to create and maintain more micro-clusters with high accuracy from multiple streams.

The idea of creating micro-clusters at the leaf level means that the algorithm can take a snapshot and send the results to any offline clustering as with the ClusTree algorithm. However, it should be emphasized that ClusTree is applied on a single stream

in a serialized model while our proposed algorithm is applied on multiple streams in a parallel model. Figure 4.9 compares our proposed concurrent clustering of multiple streams with the ClusTree algorithm.

As described in Section 4.4.1, ClusTree uses a buffer for each entry of each node to do anytime clustering. As an example of anytime clustering of ClusTree, suppose that data object 1 arrives at time stamp $t$. Meanwhile data object 1 is descending the tree to find the proper micro-cluster. Data object 2 arrives at time stamp t+1. The insertion of data object 1 is interrupted in the middle of its path in the tree, for example at level i which is not the leaf level. Data object 1 is added to the buffer 's entry of node on level i. Then data object 2 descends the tree. Data object 1 is waiting at the buffer to be picked up by a new arriving data object. Data object 1 can be successfully inserted to an appropriate micro-cluster, provided that data object 1 and the new arriving data object belong to the same subtrees. Otherwise, data object 1 might be obsolete and deleted.

In our proposed concurrent clustering, as can be seen in Fig 4.9, arriving new data objects do not interrupt the insertion process of the current data object, except when they need to modify the same leaf node. In this situation, the leaf node will be write-locked and just one of the data objects has access to this part of the shared memory. Therefore, intuitively, data objects from multiple streams can descend the tree through different subtrees. In this way, data objects have more opportunity to be added to the closest micro-clusters in near real time.

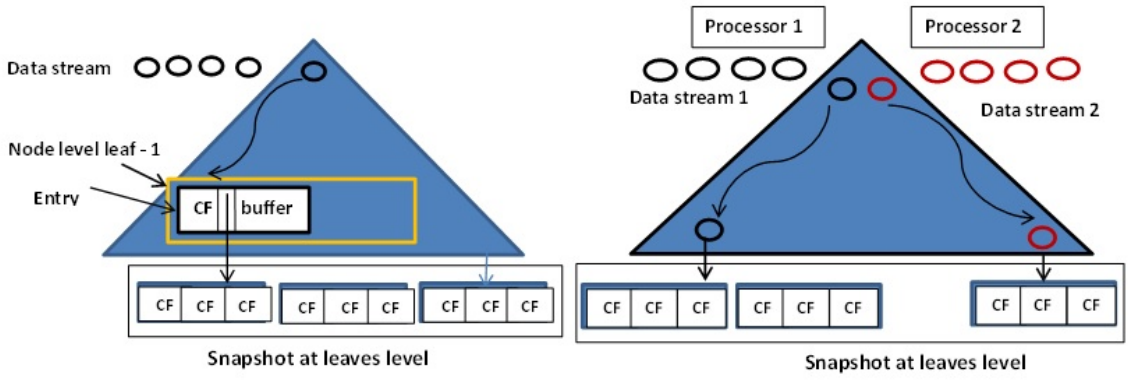



Figure 4.9: *Comparison of ClusTree (Left) and Proposed Concurrent Clustering (Right)*

### 4.5.2 The Anytime Concurrent Clustering Algorithm

Our proposed clustering algorithm is based on using micro-clusters to present data distribution in a compact way. Micro-clusters are broadly used in stream clustering to create

and maintain a summary of the current clustering. A micro-cluster stores summary statistics of data objects as a cluster feature tuple $CF$ instead of storing all incoming objects. A cluster feature tuple ($N$, $LS$, $SS$) has three components: $N$ is the number of represented objects, $LS$ is the linear sum and $SS$ is the squared sum of data objects. Maintaining these summary statistics is enough to calculate mean, variance and other parameters such as centroid and radius, as follows.

$$Centroid\text{: } \overrightarrow{x_0} = \frac{\sum_{i=1}^{N} \overrightarrow{x_i}}{N}$$

$$Radius\text{: } R = \left( \frac{\sum_{i=1}^{N} \sum_{j=1}^{N} (\overrightarrow{x_i} - \overrightarrow{x_j})}{N} \right)^{\frac{1}{2}}$$

Each cluster feature represents a micro-cluster of similar data objects with the following properties.

**Additivity Property of *CF*:** CF has the property of additivity which means if $CF_1$= ($N_1$,$LS_1$,$SS_1$) and $CF_2$=($N_2$,$LS_2$,$SS_2$) then CF=$CF_1$ + $CF_2$=($N_1$+$N_2$,$LS_1$+$LS_2$,$SS_1$+$SS_2$).

**Subtractive property of *CF*:** CF has the subtractive property which means that if CF=($N$,$LS$,$SS$) and $CF_1$=($N_1$,$LS_1$,$SS_1$) then $CF$ -$CF_1$=($N$-$N_1$,$LS$-$LS_1$,$SS$-$SS_1$).

These properties of cluster features are used when a cluster feature tuple requires an update. For example, when two micro-clusters are merged, the cluster feature of the merger is calculated using additivity property.

We extend the ClusTree algorithm into a parallel model in order to cluster multiple streams concurrently. We propose the use of a concurrent index structure from the R-tree family to maintain cluster features in a hierarchical structure. As in all such tree structures, internal nodes hold a set of entries between m and M (fanout) while the leaf nodes similarly store a number of entries between l and L. Figure 4.10 shows the details of internal node's entries and leaf node's entries of our proposed tree structure. An entry of an internal node contains [$CF$ (N, LS, SS), Child-ptr, $LSN$], where $CF$ is a cluster feature of data object (s), Child-ptr is a pointer to its child node and $LSN$ is a logical sequence number. $CF$ is calculated for each dimension of the data object. For a d-dimensional data

object, the linear square and sum square are calculated for all d-dimensions. An entry in a leaf contains a cluster feature of data object(s), and $LSN$.

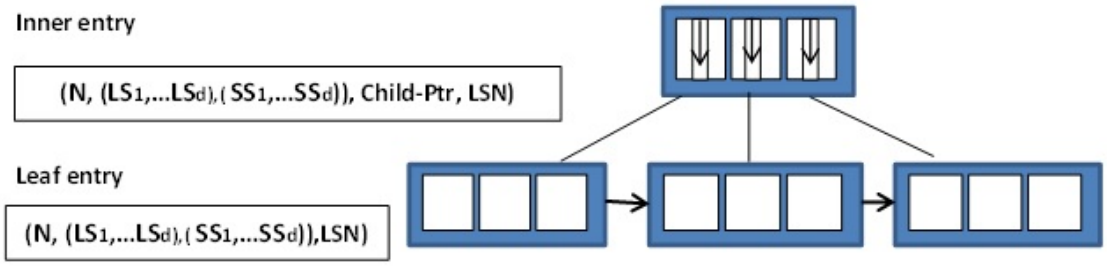



Figure 4.10: *Internal node and leaf node structure of proposed concurrent clustering*

The hierarchy of our concurrent clustering scheme is created like an R-tree except that cluster features are stored instead of bounding rectangles. Incoming data objects are clustered accordingly. First, we have to find the proper micro-cluster to insert an arriving data object into. To achieve this, a data object descends the tree by starting from the root. At each node, the distance between *CF* of the data object and *CF* of the node's entries are calculated. The entry with the closest distance is selected. The selected entry has a pointer to its child, so the data object descends the tree using the pointer. The data object descends the tree towards the leaf level for a proper micro-cluster. When descending the tree, the time stamp of the visiting node is updated.

As illustrated in Figure 4.10, both the node and its entries have certain capacities. This means that before a data object is inserted to the closest entry at the leaf level, the capacity of the closest entry is checked. Different scenarios occur. Firstly, the closest entry (proper micro-cluster) has enough space for the data object. After an insertion, the cluster feature of the entry will be updated through the additivity property of *CF*. Secondly, the closest entry does not have enough space to insert the data object. In this situation, the capacity of the node containing the closest entry is checked. If the node has enough space, a new entry is created to insert the data object into. Then, a new entry at the parent of the node should be created to point to the created new entry at the node. Finally, if neither the closest entry nor its node has enough space for inserting a data object, the node will be split. Splitting a node means a new node is created which needs a parent to point to it. This splitting to create a parent entry could be continued at upper levels of the tree until the root. If the root is split, then the height of the tree will be increased by one.

In our concurrent clustering, in order to recognize node splitting, we use a right-link approach similar to the concurrent R-tree [186]. Suppose that the Data Object 1 from Data Stream 1 and Data Object 2 from Data Stream 2 are concurrently descending the tree to be inserted into their closest micro-clusters. Data Object 1 reaches leaf level and is inserted into a closest entry of leaf node, but this insertion causes a split. Data Object 2 reaches the same leaf node and wants to be inserted into the split node. If the leaf node has been split and Data Object 2 does not recognize this split, to be able to traverse this dynamic tree correctly, likewise R-link-Tree, we modify the ClusTree into the concurrent version by adding extra features.

First, Logical Sequence Number ($LSN$) (as shown in Fig 4.10) is assigned to each node to recognize the split. Second, we link all nodes at each level of the tree using a link list. Using $LSN$ allows the split to be recognized and helps to decide how to traverse the tree. Also, linking all nodes at each level of a tree enables movement to the right of a split node.

Figure 4.11 presents an example where a node is split and the right-link along with $LSN$ is used to chain the split. One of the properties of the R-link-tree data structure is order insensitivity. As can be seen in Fig 4.11, it is possible that node $P_1$ is ordered before node $P_2$ (from left to right at each level) but because of a split, the child of $P_1$, $C_4$, is visited after child of $P_2$, $C_1$.

Using a global counter, each node has its unique $LSN$. Every entry of each node and its child's entries have the same $LSN$. In the occurrence of a split, a new right sibling node will be created for the split node. The $LSN$ of a split node is given to the new right sibling and a new $LSN$ is assigned to the split node. A data object descending the tree recognizes the split by comparing $LSN$ of a visiting (parent) node and its child node. If $LSN$ of the parent and its child is equal, no split has occurred; otherwise if $LSN$ of the child node is greater than its parent node, it means there is a split and the clustering process moves to the right of the child node until it visits a node with the same $LSN$ of the parent node, showing the furthest right node split off the old node. The possibility of moving right to the split node is provided by using a link-list of nodes at each level of the hierarchy.

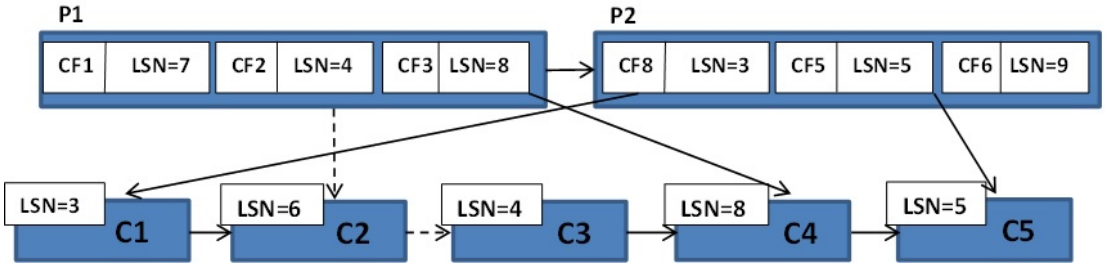



Figure 4.11: *Node split recognition using LSN and right-link*

**Algorithm 1** Clustering Algorithm

**INPUT:** D-dimensional data objects $O_i, O_j, ...$
**OUTPUT:** Inserting data objects $O_i, O_j, ...$ into the closest micro-clusters

```
1: for all processors P_i, P_j, ... do
2:    ClosestMicrocluster = searchLeaf(root, O, root-lsn)
3:    insert O on ClosestMicroCluster at leaf
4:    if leaf was split then
5:       expandParent(leaf,CF(leaf),LSN(leaf),right-sibling,CF(right-sibling), LSN(right-sibling))
6:    else if CF of leaf changed then
7:       updateParent(leaf, CF(leaf))
8:    else
9:       w-unlock(leaf)
10:   end if
11: end for
```

For concurrency control, we use a lock-coupling technique in such a way that during the process of traversing the tree, nodes are read-locked. Hence, data objects from different streams can access the tree and descend the tree in parallel. The main issue is at the time of inserting a data object into a micro-cluster, then updating the *CF* of its parent at the upper level of the tree. To solve this problem like in a R-link-tree, we use the write-lock; when a data object is being inserted into a leaf node, the leaf node is locked. After an insertion, the *CF* of the node's parent should be updated. Therefore, the parent is locked and the leaf node is unlocked.

Our main proposed concurrent clustering algorithm (as shown in Algorithm 1) consists of a search process to find the closest micro-cluster, updating the *CF* of the parents after clustering, expanding the parents in the case of split child and installing a new entry for this split at upper levels of the tree. The algorithm is the same as the concurrent R-tree algorithm [186] except that our purpose is to manage the micro-clusters. We

explain each function in details as follows.

**searchLeaf:** The searchLeaf function is called at the beginning of the clustering algorithm (1) to find the closest micro-cluster for a given data object at the leaf level. The searchLeaf function starts the process of a search from the root. During the process of searching the tree, if a visiting node is not a leaf, it is read-locked. Otherwise, it is write-locked. For each node, the *LSN* of the visiting node is compared with the *LSN* of its parent. If the *LSN* of the parent is smaller than the *LSN* of the visiting node, a split has occurred. Therefore, the tree is traversed to the right of the visiting node (split node) till finding a node with the *LSN* equals to the *LSN* of the parent guarantee to find the closest entry even after a split. If the split node is at the leaf level, then the searchLeaf function returns the closest entry as the closest micro-cluster to the clustering algorithm. Otherwise, the process of search keeps descending the tree recursively from the child of the closest entry and the visited node is read-unlocked.

**expandParent:** After finding the closest micro-cluster through the searchLeaf function, the data object is inserted. If the insertion of the data object causes a split, then the expandParent function is called. The expandParent function either installs a new entry as the parent of the new created leaf (because of the split) at the top level of the split leaf or finds an entry for the new created leaf in the parent of the split leaf node or its right sibling. The former is a new split at the parent level of the split node. Therefore, the expandParent function is recursively called up until the root is split or no further split happens. During the process of expanding a parent, the child nod is write-locked until the parent is accessed. Then, the child node is write-unlocked and the parent is write-locked.

**updateParent:** Whenever a data object is inserted into the leaf node and its CFs are updated, or CFs of a parent are updated because of a split, the updateParent function is called to propagate these updates up to the parent's levels.

We propose to optimize the process of clustering by finding top-k closest micro-clusters. This means that descending the tree by finding the closest entry among all entries of visiting nodes does not guarantee arrival at the closest micro-cluster among all other micro-clusters at leaf level. Therefore, to find a global optimum; the closest micro-cluster among all micro-clusters, we use a stack data structure to keep track of the top-k closest

entries to a data object. In order to maintain up-to-date clustering, we use a buffer in each node, whenever a new data object arrives and descends the tree, the time stamp of the visiting node is updated like ClusTree.

## 4.6 Distributed Streams Clustering-DistClusTree

In this section, we first describe the problem of continuous distributed clustering and then we discuss our proposed solution.

Assume $m$ sites are distributed in a network and each of them receives a stream of data. Every site clusters its own unlimited data continuously. A central site $q$ monitors and clusters data over the union of all $m$ local sites. Our interest is in devising a method so that the local sites communicate with the central site in an efficient way, finding summarized data but is still informative to be sent to the central site for global clustering instead of sending the actual data points in order to reduce communication costs and to preserve privacy. Naively, a communication can be triggered between local sites and the central site every time new data objects arrive at each local site so as to update clusters maintained at the central site. This poses an expensive communication cost of $O(n)$, where $n$ is the number of data points at local sites.

In the DistClusTree framework, $m$ local sites are distributed in a network and each receives and incrementally clusters a continuous stream of data, possibly with an infinite length. A master/central site instead clusters and maintains the union of the local site data to produce the final global clustering result. We are particularly interested in devising mechanisms that allow local sites to communicate with the central site efficiently. Leveraging is a key approach to reduce communication costs and to preserve privacy. Instead of sending the actual massive data, It summarizes the key points to be sent to the central site for global clustering but keeps it informative. Moreover, a communication can be triggered between local sites and the central site every time a new data object arrives at each local site so as to update clusters maintained at the central site. This poses an expensive communication rounds of $O(n)$, where $n$ is the total number of arrived data points. Therefore, a plausible way to balance the communicate cost and clustering quality is to trigger communication periodically and only send the selected updated summaries/representatives to the central site. Our studies shows the choice of proper local representatives has a significant impact on communication costs and central clustering results. The representatives form a summary of local models at a given time

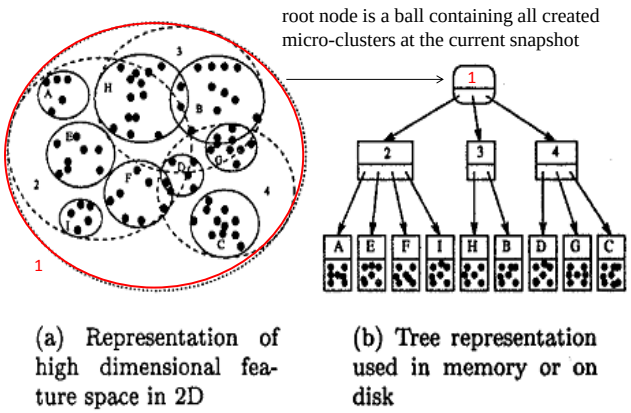
snapshot. The summary is sent to the central site and continues to be locally updated as new data points (stream snapshots) arrive.

In essence, DistClusTree consists of four stages, described in detail in the following sections:

1. Continuous local clustering;
2. Extracting local representatives;
3. Distributed microcluster tracking; and
4. Maintaining global clusters.

### 4.6.1 Continuous local clustering

Every local site clusters its data incrementally with the ClusTree approach. Summaries of data are collected as $CFV$s and maintained dynamically in an R tree. In this way, micro-clusters are maintained in various levels of the tree and in different resolutions (i.e. coarser micro-clusters are located in higher levels of the tree). Therefore, the root node in the ClusTree contains the broadest view of all micro-clusters at the current snapshot, while the leaf nodes include all the fine-grained micro-clusters. Such a hierarchically-summarized organization is shown in Fig 4.12.




(a) Representation of high dimensional feature space in 2D

(b) Tree representation used in memory or on disk

Figure 4.12: *The local ClusTree summary in DistClusTree (source [6])*

### 4.6.2 Extracting local representatives

To extract local representatives, we propose two simple but effective and adaptable approaches:

1. Naive-DistClust
2. DistClust.

**Naive-DistClust**- Local representatives from different levels of the tree are extracted regularly (i.e. at every $\Delta T$ time period) to be communicated to the central/global site. This approach is adjustable based on the network traffic (i.e. the frequency of data arrivals), degree of privacy, and required quality of central clustering. Depending on traffic and required quality of clustering, local sites can send created micro-clusters at different levels of the tree to the central site. For maximum quality, local sites should send all created micro-clusters at their leaf level. While in a heavily loaded network (e.g. at peak hours) more compressed trees (i.e. at most the root level) with some sacrifice of clustering quality can be sent to the central site. This translates to reducing the overall communication cost by sending more coarse local micro-clusters from higher levels of the tree to the central site.

**DistClust**- A further way to reduce communication costs is for every tree node to send only statistical summaries of its contained micro-clusters to the central site. For example, in an R-tree only with 3-fan outs (i.e. the number of entries in each node where each entry represents one summarized micro-cluster) at Level 1 (considering Level 0 as the root level), we have 3 subtrees each containing 3 micro-clusters. This means we have 9 micro-clusters in total at Level 1. Instead of sending all nine entries to the central site, we could choose to send only the median of the entries from each node, thereby reducing communication cost to one-third. Next, we discuss in detail how local representatives (i.e. the selected micro-clusters) are tracked and sent with an on-line tracking algorithm.

### 4.6.3 Distributed micro-cluster tracking

We first provide a brief introduction on online tracking and then illustrate how we formulate the global clustering in DistClusTree as an online tracking problem. In conventional online tracking, a pair of observer and tracker communicates with each other. Observer observes values of a function $f$ over time and keeps the tracker informed of these values

from time to time as shown in Figure 4.13. However, the determination of a strategy that minimizes communication costs is the main issue in online tracking problems. A naive solution is that the observer sends every observed value to the tracker. This leads to a heavy communication. To minimize the communication cost, an error threshold is generally introduced. This means observers only communicate with the tracker whenever a value of $f(t_{now})$ exceeds a predefined error threshold $\Delta$ from the last communicated value $f(t_{last})$.

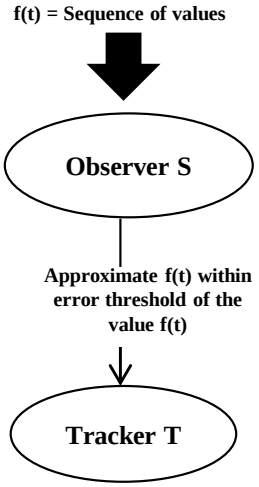



Figure 4.13: *One-to-one online tracking.*

We extend the multidimensional one-to-one online tracking framework presented in [210] that only works when there is an observer and a tracker. It is not designed for the distributed $m$-to-one communication where there are multiple observers and a central tracker. The one-to-one online tracking algorithm divides the entire tracking period into rounds and denotes $A_{opt}$ as the offline optimal algorithm.

A round is started by initializing a set $S = S_0$ which contains all possible points that might be sent by $A_{opt}$ in its last communication. In a while loop, a median of $S$ is calculated and sent to the tracker. If $\|f(t_{now}) - f(t_{last})\| \geq \beta\Delta$, where $\beta = 1/(1+\epsilon)$ and $\Delta$ represents the threshold error , then $S$ is updated as $S \leftarrow S \cap Ball(f(t_{now})), \Delta)$, where $f(t_{now})$ is the center of Ball and $\Delta$ is its radius in $d$-dimensional space. A round is terminated when $S$ becomes empty. The online tracking algorithm is represented in Algorithm 2.

We model our clustering algorithm based on Algorithm 2. First, we show how we can keep track of micro-clusters assuming there are one local site and one central site, and then we extend our algorithm from one local site to multiple local sites as shown in

Figure 4.16 (i.e. distributed) that they communicate with a central site in a synchronous mode. We explain with the Definition 2 and Lemma 4 from [210] below to illustrate our tracking model.

**Algorithm 2** One round of d-dimensional tracking via volume-cutting (source [210])

1: Let $P = Ball(f(t_{now}), \beta\Delta)$
2: **while** $(\omega_{max}(p)) \geq \epsilon\Delta$ **do**
3: Let $g(t_{now})$ be the centroid of $P$
4: send $g(t_{now})$ to tracker
5: Wait until$\|f(t_{now}) - g(t_{last})\| \geq \beta\Delta$
6: $S \leftarrow S \cap Ball(f(t_{now}), \Delta)$
7: **end while**

**Definition 2 (Directional Width).** For a set $P$ of points in $R^d$, and a unit direction $\mu$, the directional widths of $P$ in direction $\mu$ is $\omega_\mu(P) = max_{p\in P} \langle \mu, p \rangle - min_{p\in P} \langle \mu, p \rangle$, where $\langle \mu, p \rangle$ is the standard inner product.

For simplicity, suppose a given set of points form a convex set $P$, and the centroid of $P$ is the intersection of hyperplanes that divide $P$ into two equal parts. This convex set has minimum and maximum directional width as $\omega_{min}(p)$, $\omega_{max}(p)$ , respectively.

**Lemma 4**. For any convex set $P$, if $H$ is any supporting hyperplane of $P$ at $p \in \partial P$, that is, $H$ contains $p$ and $P$ is contained in one of the two halfspaces bounded by $H$. Then there is a ball $B$ with radius $\beta\Delta$ such that $H$ is tangent to $B$ at $p$ and $B$ contains $P$ as shown in Figure4.14.

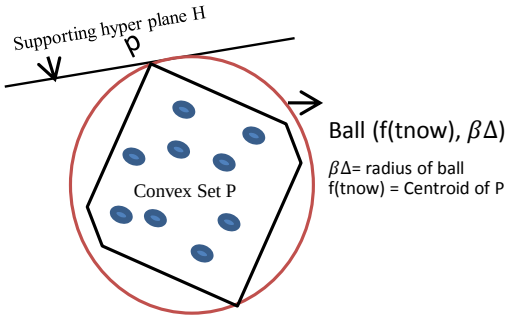



Figure 4.14: *Convex set $P$ is covered by $Ball(f(t_{now}, \beta\Delta)$.*

A local site acts as an observer that sends an approximation of created micro-clusters to a central site at different time snapshots. Assume that error threshold $\Delta$ is determined based on the maximum distance between centroids of two clusters at two consecutive snapshots of $t_i$ and $t_{i+1}$, where $C_{t_i} = f(t_{last})$ and $C_{t_{i+1}} = f(t_{now})$ are the centroids of the previous and current root nodes.

According to [210], a convex set $P$ in our local ClusTree is a set of micro-clusters taken at snapshot $t_{now}$. Based on Lemma 4, a ball containing $P$ is the root node at snapshot $t_{now}$. Following this, we initialize $P$ with the root node, and then centroid of the current root (i.e. as a representative of all microclusters) is computed as $g(t_{now})$ and sent to the tracker. In the next snapshot, if the absolute euclidean distance between the centroid of the previous root node and the current root node is within the predefined error threshold, then there is no communication. Otherwise, the intersection of two roots (two balls) is computed. If the maximal directional width of this intersection is greater than $\beta\Delta$, then a communication between local site and the central site is triggered and the centroid of the intersection is sent to the central site. Otherwise, this round is finished and the next new round is triggered. Different scenarios that may happen between the last communicated root node and the new root node at two different snapshots of $t_1$ and $t_2$ are presented in Figure4.15 (a-c).

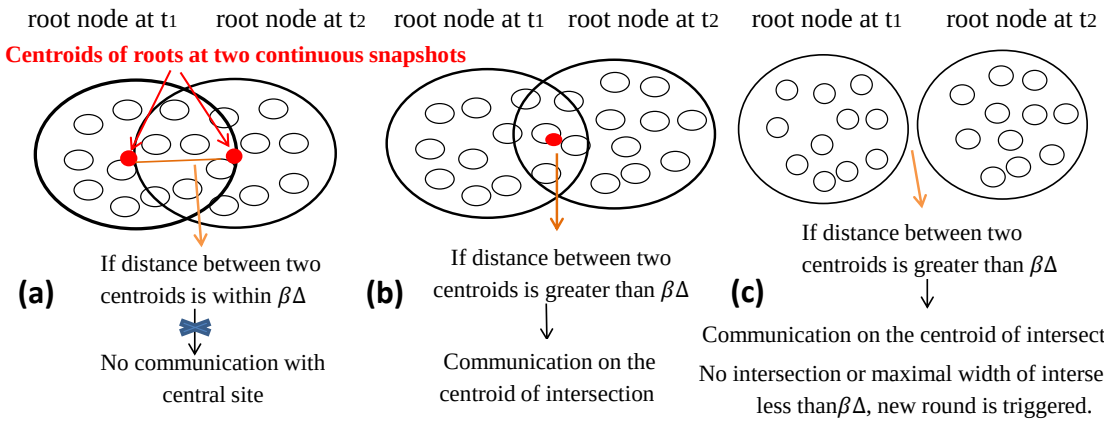



Figure 4.15: *Different scenarios to trigger a communication between a local site and a central site to update the global clustering.*

### 4.6.4 Maintaining global clusters

For simplicity, we only enhance the tracking framework from 1-to-1 to $m$-to-1 sites in a synchronous manner. Each local site keeps track of its local representatives in periodical

time snapshots and if any threshold breaks at a local site, then this site simply sends its updates to the central site along with all other updates from other local sites. As depicted in Figure 4.16, local sites communicate with a central site if $f_1(t_1) - f_1(t_2)$ are within some error threshold. After sending updates to the central site, central site does a global clustering using $k$-means over the union of all received micro-clusters from local sites. Local sites incrementally send their updates to the central site to keep global clusters updated. By receiving regular updates, the central site incrementally keeps global clusters updated.

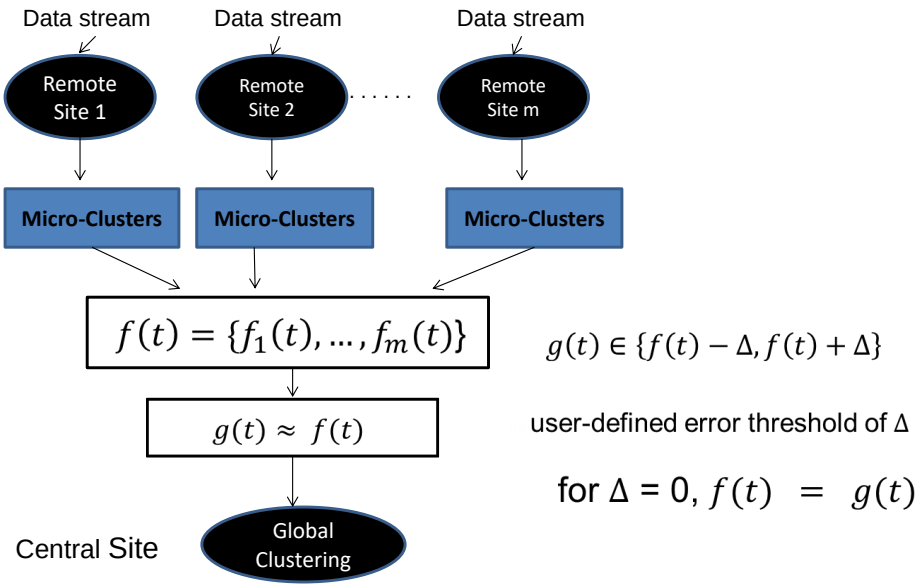



Figure 4.16: *The global DistClusTree framework.*

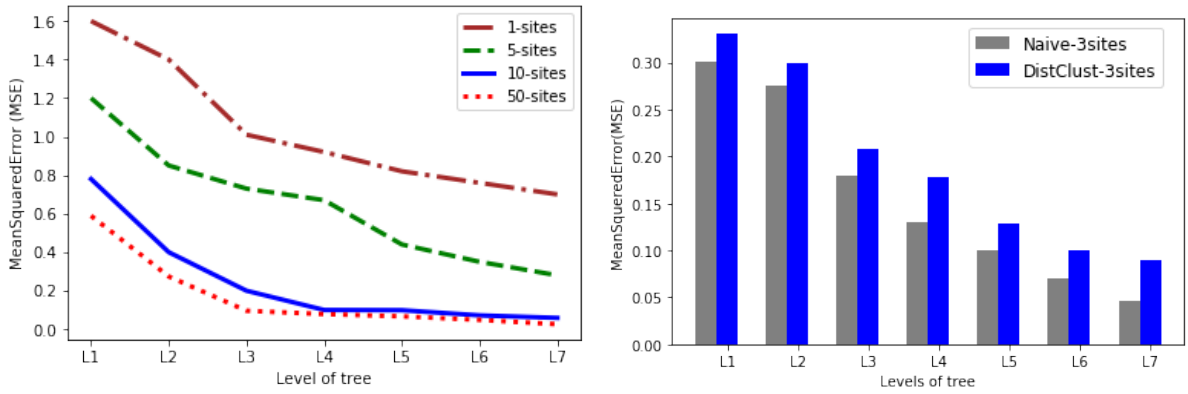
## 4.7 Experimental Results

We implemented DistClusTree under Massive Online Analysis (MOA) [211] and evaluated the distributed algorithms based on a synthetic dataset. The dataset was generated using Gaussian distribution with varying number of attributes and classes. Data points were randomly and equally divided among sites and for the central clustering, we used the union of the local points. Our experiments focused on clustering quality and the communication costs of distributed clustering considering their dependency on different parameters such as the number of sites, the accuracy $\epsilon$, granularity of local representatives and runtime of distributed clustering in comparison with centralized clustering. To

assess our framework, we ran ClusTree on each local site and then collected all representatives of local sites w.r.t the demanded granularity of local clusterings (i.e. different levels of local trees) and then performed a global clustering on these representatives. We executed all the experiments on the same machine and reported all the results as average of 10 runs of our algorithms. We compared clustering quality of our distributed clustering (i.e. DistClusTree) against its centralized counterpart, i.e. ClusTree. As mentioned in [203], different studies have evaluated their algorithms based on characteristics of their distributed clustering algorithm in a variety of ways and most studies compare their proposed distributed clustering algorithm against their centralized counterpart [176]. Therefore, we compared the result of our distributed clustering algorithms to a central clustering of the $n$ data points when all $n$ data points are clustered using ClusTree in local sites and applying $k$-means on top of the micro-clusters created at the leaf level of the tree.

**Clustering quality**- We measured the quality of clustering by defining Mean Squared Error (MSE), and also using within-cluster sum of squares error. As the baseline, firstly we sent all micro-clusters created at the leaf level to the central site and applied $k$-means to calculate cluster centroids. Secondly, we sent micro-clusters of each level of the local trees and find cluster centroids for each level using $k$-means. To calculate MSE, we take the average of euclidean distances between cluster centroids obtained from the last level of the tree in ClusTree (i.e. centralized model) and every level of the local trees (DistClust/Naive-DistClust). As can be seen in Figure 4.17b, MSE is reduced by descending tree since micro-clusters with smaller within-cluster sum squared error are located at the lower level of the tree which impacts on the quality of clustering in the central site. We compared MSE of $k = 5$ centroids at different levels of the tree for both DistClust and Naive-DistClust. In the latter, we sent all created micro-clusters from different levels of trees while in DistClust we only sent a mean of micro-clusters of each node of tree. That is why MSE between ClusTree and NaiveClust is less than MSE between ClusTree and DistClust. The MSE difference between both distributed algorithms gets higher at the lower levels of tree since granularity increases at the bottom levels, and sending fewer micro-clusters impacts on calculating right centroids and consequently on clustering quality.

We ran the experiments with three fan-outs at each node of R-tree as referring to [212], 3 fan-outs is the best number of entries (number of micro-clusters at each node) in terms of space and distance computation.




(a) *Clustering quality of different sites.* (b) *Clustering quality on different levels*

Figure 4.17: *Clustering quality comparison based on MSE. (a) Comparison of clustering quality for increasing number of sites,1, 5, 10, 50. (b) Comparison of clustering quality of Naive-DistClust and DisClust with ClusTree on different levels of tree, when number of sites = 6 and number of entries = 3.*

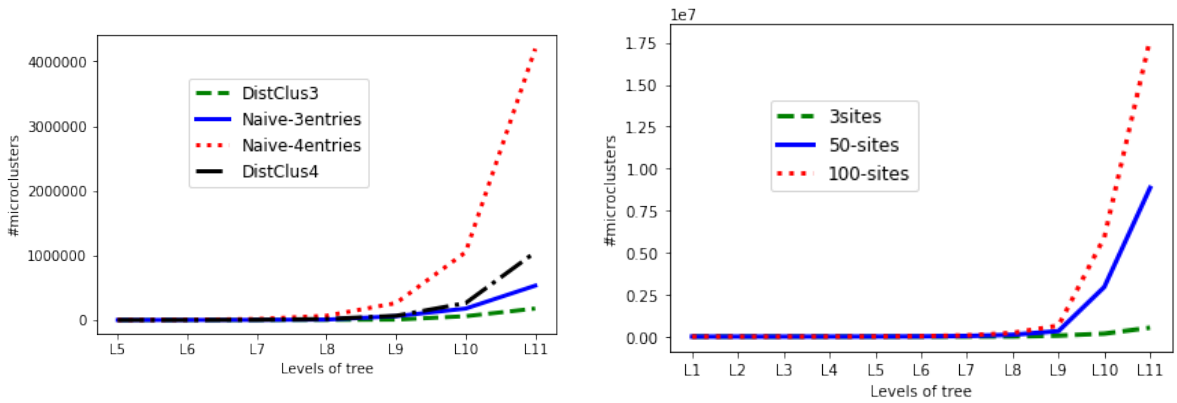
On the other hand, as can be seen in Figure 4.17b, MSE is reduced by descending the tree. The reason is that purity of micro-clusters at lower levels of the tree is increased which causes MSE to reduce between cluster centroids obtained from upper and lower levels of the tree. We tested our framework for different numbers of sites:1, 5, 10, and 50. Although in all plots in Figure 4.17a MSE was reduced by descending the tree, reducing the number of sites also reduces the quality of clustering because we sent fewer micro-clusters: this impacts on the final quality of clustering at the central site.

**Communication costs**- We calculated communication costs in terms of the number of transferred micro-clusters from each level of the tree as shown in Equation 4.1.

$$\text{communication ratio} = \frac{\text{compressed tree}}{\text{uncompressed tree}} \times \text{number of sites} \tag{4.1}$$

In our formula, communication ratio is calculated as the ratio of compressed tree to uncompressed tree. An uncompressed tree is a full multi-way tree with a maximum number of levels. The maximum number of levels is predefined by the user or local memory limits. A compressed tree is a full multi-way tree with fewer levels compared to the uncompressed one. The lowest communication ratio is the ratio of the minimum number of levels (maximum compression, i.e. root level of tree) to the maximum number of levels. The communication costs also depend on the number of entries in each node. For instance, in a 3-way full tree, Level 0 which is the root level has 1 node, Level 1 has 3 nodes, Level 2 has 9 nodes and in general for n-multi-way tree, the number of nodes in Level $n$ is calculated as $m^n$ , where $m$ represents number of ways in the tree.

We compared the communication costs of different levels of the tree for a different number of entries at each node of the R-tree. In Figure 4.18a, we compare the communication cost for two different number of entries, 3 and 4 in our two proposed distributed clustering algorithms. By sending the median of micro-clusters at each node in DistClust, we reduced communication costs to $1/k$, where $k$ is the number of entries. We reduce communication costs to one third with Distclust for the choice of three entries in all levels of the tree. This reduction is obvious in the lower levels of the tree where more granular micro-clusters are required.




(a) (b)

Figure 4.18: *(a) Effect of the number of entries on communication costs; (b) Communication costs of DistClustTree for a different number of sites when the number of entries equals 3*

Figure 4.18b represents the communication costs in terms of the number of micro-clusters for 3, 50 and 100 sites. The number of entries in all three experiments has

been set to 3 and the height of the tree is 11. It can be clearly seen that communication costs depends on the number of sites and different levels of the tree. To have more granular clusters we need to send micro-clusters at the lower level of the tree, causing communication costs to increase exponentially. However, by sending representatives from upper levels of trees we reduced communication costs significantly and still obtained good quality clustering as demonstrated in the above experiments.

**Effect of error threshold** $\Delta$- We evaluated the effect of varying the error threshold on communication costs. Error threshold is the difference between the centroid of the new micro-cluster at snapshot $t_i$ and that previously transmitted at snapshot $t_{i-1}$. As the error threshold is increased, the communication cost is decreased since we send fewer updates to the central site by increasing euclidean difference between centroids of previous and current snapshots. The communication cost at the lower levels of the tree is greater than the upper levels of tree as shown in Figure 4.19 for L1 as root level and Level 6. However, increasing the error threshold reduces the quality of clustering.

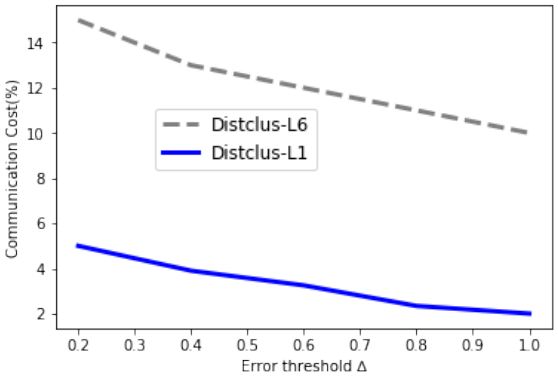



Figure 4.19: *Effect of different* $\Delta$ *values on communication cost, levels 1 and 6, for 10 sites.*

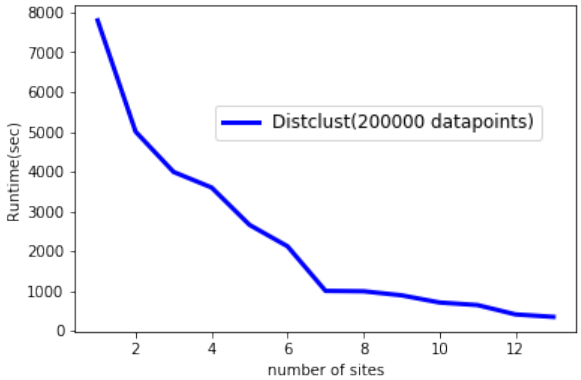



Figure 4.20: *Runtime for central and distributed clustering with varying number of sites.*

**Runtime**- In Figure 4.20, the runtime of DistClusTree is shown. As the number of sites increases, the distributed approach performs much better than a single clustering algorithm applied to the complete data set of 200k data points.

## 4.8 Summary

In this chapter, we proposed two multiple stream clustering algorithms using an indexing tree in centralized and distributed models. Our proposed algorithms are based on one of the well-known micro-clustering techniques, ClusTree [4].

In the former, we proposed a new, anytime, concurrent, multiple stream clustering algorithm. We captured the summary statistics of multiple data streams concurrently in the online phase. We proposed to maintain statistical information of the data locality in micro-clusters at a dynamic, multiple access index data structure for further offline clustering. In the online phase, the index data structure maintains summaries of data in the format of cluster feature tuples ($CF$) instead of storing all incoming objects. Then, the data structure was traversed through an index to insert new data objects concurrently into their closest micro-clusters.

We also extended ClusTree into DistClusTree, a comprehensive distributed framework for stream clustering. The framework leverages both spatial index summaries and online tracking for balancing communication cost and clustering quality. We demonstrated in experiments that DistClusTree efficiently produces clusters as good as its centralized version. DistClusTree is able to reduce communication cost significantly and it is easily configurable in practice according to the requested clustering quality.

In the next chapter, we will study the problem of merging summaries in distributed settings. We will use index data structures to maintain histogram(s) (i.e. another summarization technique) of distributed summaries. We will discuss how we resolve the problem of non-mergeable summaries using cutting head techniques, i.e. deep generative models.

CHAPTER 5

# MERGEABLE SUMMARIES

Histograms provide a compact and effective approach to summarize large data sets. They represent underlying data distribution by partitioning data into a number of blocks or buckets, in a concise manner for the aim of data visualization and analysis. The importance of histograms is highlighted by their wide applicability in a variety of areas from image processing (e.g. in image enhancement), statistical and scientific data analyses to database areas including query optimization, approximate query answering and range query answering in spatial data bases.

In this chapter, we study the problem of merging and continuously maintaining a centralized histogram over the union of distributed summaries in multidimensional spaces. We introduce a new, practical framework to build and maintain a multidimensional histogram over continuous distributed data streams. At the center of our framework, we use a dynamic index data structure to maintain a succinct approximation of the data distribution of the underlying continuous data streams. Our framework is simple, and easy to implement. Meanwhile, it is an efficient summarization algorithm which outputs an approximate but up-to-date histogram over the aggregation of distributed multidimensional data summaries on demand from the index data structure. Our framework is capable of simultaneously representing a centralized histogram and local histograms. Experimental results provide evidence of the performance of the proposed method when it has been utilized in parallel and distributed settings in terms of communication cost, error rate and the practicality of our algorithm.

## 5.1 Introduction

Multidimensional index data structures organize data based on the principle of recursive decomposition. They are good representatives of underlying data distributions. In the context of spatial databases, index structures have been used to accommodate spatial objects to answer different geometric queries such as point, range and nearest-neighbor. In data mining, they have also been extended for tracking clusters. Despite the enormous application of index structures in centralized applications and domains, research has been rather limited in leveraging multi-dimensional indexing in distributed environments.

In a typical distributed scenario, we have $m$ remote local sites and one central site. Each local site stores its multidimensional data points with a multidimensional index tree data structure capturing some distribution property such as clusters or medians/histograms. The central site needs to build a global index over the entire distributed data: this creates a global distribution property. There are commonly two ways to achieve this: i) local sites simply send all data points to the central site for global index construction; ii) local sites only send indices as data summaries and the global index is approximately built and maintained purely from local indices. Clearly, for i) communication cost is often unacceptable; for ii) as to the case of $kd$-tree index or similar, merging local indices is often ineffective.

For tracking data clusters in both centralized and distributed settings, $R$-tree has been extended in [212] and [213] and merge of local $R$-trees at a central site has been straightforward [214]. However, for tracking median information in the form of $kd$-trees [215], their merge and maintenance seem impossible as median instead describes the order statistics. A main application of $kd$-tree is maintaining quantiles and intervals of multi-dimensional equi-height histograms [216]. Quantiles are similar to equal-height histograms where ranges are divided in a way that all buckets contain the same number of values. As $kd$-tree eventually divides the space into partitions having equal numbers of multidimensional data points.

In this chapter, we propose a framework called iDMS which synchronously constructs and maintains global $kd$-tree from locally distributed summaries of streaming data. In the proposed framework, we keep multidimensional histograms of local data in an index tree structure. We use kd-tree as our index structure to maintain histograms from all distributed sites in a central site. We merge summaries of local histograms (index structures) to discover equal-height histograms or quantiles, over the entire approxi-

mated distributed data at a central site. In fact, the proposed framework constructs a centralized, approximated kd-tree over the union of distributed summaries. The proposed method incrementally updates the centralized histogram (central kd-tree). Indeed, we keep track of a multidimensional equal-height histogram, known as a quantile, in a central site.

For this purpose and to overcome the above mentioned challenges from i) and ii), we use two different probabilistic and deep generative models of Gaussian Mixture Model (GMM) and Generative Adversarial Networks (GAN) to construct local summaries. We leverage local GMMs [217] or GANs for building approximate global $kd$-tree that significantly reduce communication costs and at the same time enable effective construction and maintenance.

To the best of our knowledge, our work is the first attempt to introduce a novel distributed framework that maintains the global $kd$-tree at the central site from the locally received data summaries. As the way of exactly merging $kd$-trees is deemed to be impossible due to their recursive structures along different dimensions, the prominent characteristics of our iDMS framework are the efficient and effective implementation and the extensibility to other multi-dimensional index structures and non-mergeable summaries. The main contributions of this chapter are as follows.

1. We address the fundamental problem of maintaining a global $kd$-tree via a novel distributed framework.
2. Within the framework, we propose GMM/GAN-based distributed and maintenance algorithms that are succinct, effective and efficient.
3. We demonstrate the effectiveness and efficiency of the framework (in terms of $kd$-tree approximation error and communication cost) through experiments on two real data sets and against the baseline setting (i.e. all arrived local data points are sent to the central site).

## 5.2 Chapter Organization

The rest of this paper is organized as follows. Section 5.3 reviews related work on mergeable summaries and distributed online tracking. Section 5.4 describes required preliminaries of our proposed algorithms. Our proposed framework are explained in details in Section 5.5. Section 5.6 presents experimental results. Section 5.7 concludes the chapter.

## 5.3 Related Work

The aim of this work is to develop a new framework that aggregates kd-trees/histograms into a single kd-tree/histogram. We review some of the works that carried out studies in this context either as distributed functional monitoring or mergeable summaries.

Histograms have been widely used in many applications such as similarity searching [218] and [219], approximate querying [220], [221] and [222] and data mining [223]. Most of the literature on histograms and quantiles merely considers summarizing uni-variate data in a centralized model and a few of them considers uni-variate distributed model [224], [225]. In addition, most studies focus on constructing histograms over static data sets [226] and [227] where the entire data is available. There are fewer works on constructing dynamic multidimensional histograms over continuous data streams [228] and [216], especially in distributed environments where multiple streams are generated by local sites. In many distributed and parallel application scenarios, the main issue on building a multidimensional histogram over the union of entire data is to merge distributed summaries (i.e. histograms of local sites) to retrieve a histogram over aggregated data in an efficient way.

There has been theoretical research on mergeable summaries [229] (e.g. for heavy hitters and quantiles) while little was known for merging multidimensional indices such as $kd$-tree. Yi and Zhang [210] study the problem of online tracking for both one-dimensional and multi-dimensional spaces, but they only consider the centralized setting. For both cases, they provide bounded competitive ratios.

Studies have also been carried out in the context of distributed functional monitoring [230], [231], [232]. There are multiple observers each monitors its inputs over a function continuously and communications are triggered with a coordinator to send their function outputs. The coordinator then calculates a function over the union of all outputs received from observers. The main goal of this line of research is to reduce communication costs.

Distributed streaming models have been studied in [224], [233], [234], and [235], where the goal is to find an approximation of a global function over the union of all distributed data streams seen so far. For the models with sliding windows [236], [237] and [238] calculate an aggregation function over elements within a sliding window of a predefined size. In [239], outlier detection in sensor networks has been studied. In their proposed framework, sensor measurements of a sliding window are approximated using some kernel density estimators. Recently in [234], the authors extended the problem of

distributed online tracking from two party model to more complex topologies of chain, broom and tree.

## 5.4 Preliminaries

### 5.4.1 $Kd$-tree

$kd$-tree is a generalization of binary search tree that recursively halves a space into two equal partitions interleaving on each dimension until there are one or more data points left at each leaf indexed partition. Step by step visualization of partitioning 2D-space using a kd-tree is shown in Figure 5.1 (a1-a6). Furthermore, a tree representation of $kd$-tree, Algorithm 3, is demonstrated in Figure 5.2.

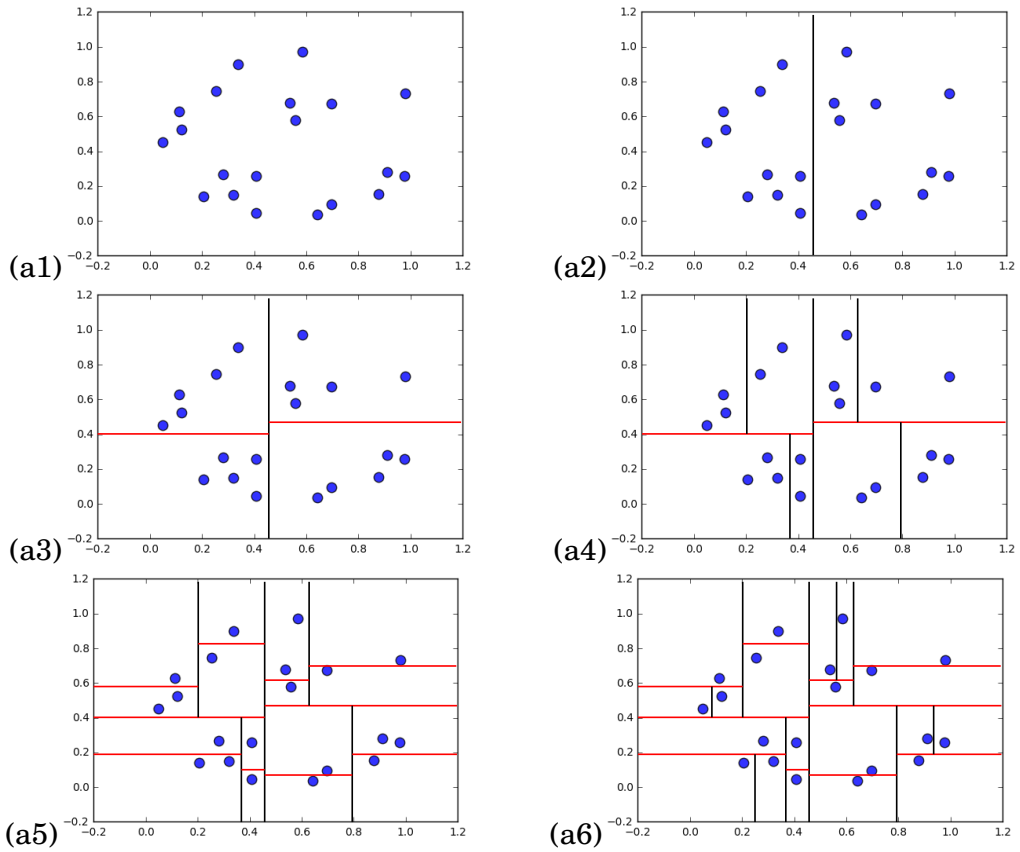



Figure 5.1: *Visualization of Kd-tree space partitioning.*

Assume $P$ is a set of $n$ data points in $R^2$. The Kd-tree algorithm starts by sorting data points on one of the dimensions e.g. $X$. Then, it finds a split point (i.e. median) on

**Algorithm 3** Kd-tree construction

**INPUT:** list of data points as pointlist, depth of tree, k=number of dimensions
**OUTPUT:** Kd-tree

1: axis:= depth mod k
2: $sortedlist \leftarrow sort(pointlist_{axis})$
3: $median \leftarrow median(sortedlist)$
4: $node \leftarrow create(node)$
5: $node \leftarrow median$
6: $node.leftChild := kdtree$(data points in pointList before median, $depth+1$)
7: $node.rightChild := kdtree$(data points in pointList after median, $depth+1$)

this dimension, and stores this middle point in the root level. Now space $P$ is partitioned into two halves, $P_l$ and $P_r$ as left and right subspace. In the next level, each of these two half spaces are partitioned into two halves on the second dimension $y$. The splitting points are stored in the second level of the tree in left and right children of the root level respectively. Figure 5.2 shows a $2d$-tree on two-dimensional data that recursively partitions the left and right sub-trees on one dimension until only one data point is left at each leaf.

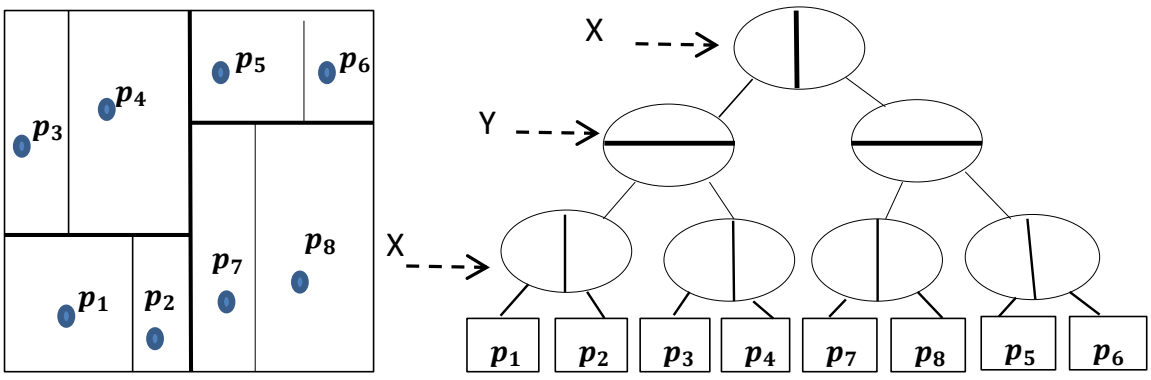



Figure 5.2: *Example of a $2d$-tree*

### 5.4.2 Gaussian Mixture Model

In this section, we explain the GMM as one of the preliminaries of our work which has been used as a component of our framework.

A mixture model is a probabilistic model for representing the presence of sub-populations within an overall population, without requiring that an observed data set should identify the sub-population to which an individual observation belongs. Formally, a mixture model corresponds to the mixture distribution that represents the probability distribution of observations in the overall population. However, while problems associated with "mixture distributions" relate to deriving the properties of the overall population from those of the sub-populations, "mixture models" are used to make statistical inferences about the properties of the sub-populations given only observations on the pooled population, without sub-population identity information. The multivariate Gaussian distribution of a cluster $c_i \in R^d$ as shown in Equation5.1 is parameterized by its mean $\mu_i \in R^d$ and co-variance matrix $\Sigma_i \in R^{d \times d}$ and the probability of a data point $x$ given the cluster is:

$$P(x|c_i) = \frac{1}{\sqrt{(2\pi)^d \Sigma_i}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)} \tag{5.1}$$

For a Gaussian mixture of $m$ clusters, the probability density function is represented in Equation 5.2.

$$P(x) = \sum_{i=1}^{m} w_i p_i(x|c_i) \tag{5.2}$$

where $w_i$ is the weight of cluster $c_i$ in the mixture model. Given the data and its prescribed number of clusters $m$, its GMMs can be represented as vectors of parameters $(w_i, \mu_i, \Sigma_i)$ for $i = 1, \cdots, m$ and these cluster and parameters can be learned through the well-known iterative Expectation Maximization (EM) algorithm similar to $k$-means algorithm.

Examples of clustering using GMM have been shown in Figures 5.3.(a-d). As can be seen, GMM discovers various number of clusters in a multidimensional space based on a user-defined parameter setting. For example in Figure (5.3-b) by setting number of components (i.e. number of clusters) to one, GMM is forced to find one cluster in data while in Figures a, c and d the number of clusters has been set to 3 and 2 respectively. However, it needs to be acknowledged that seed initialization and number of iterations are two other important user-defined parameters in finding clusters through GMM.

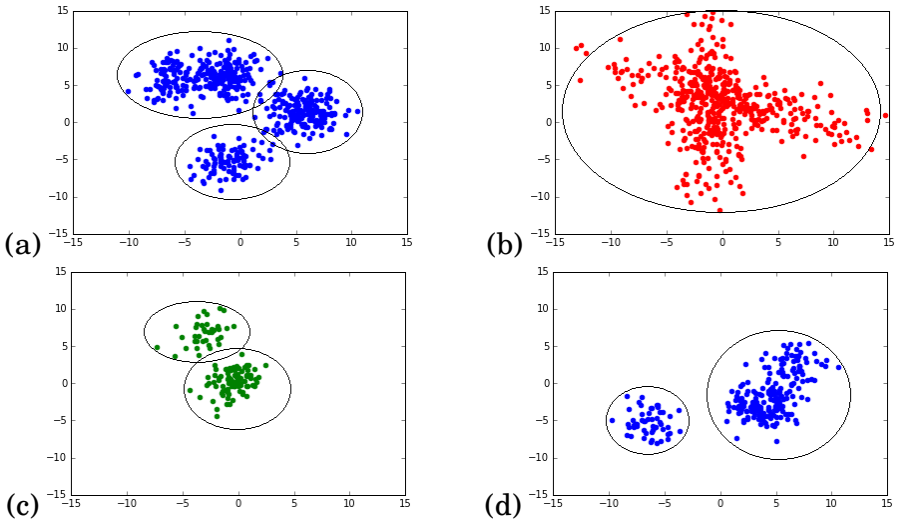



Figure 5.3: *Examples of clustering using GMM.*

### 5.4.3 Generative Adversarial Network

GAN is another technique of data approximation in our framework that we explain in this section. The idea of a deep generative model is to take noise such as Gaussian noise and inputs to the neural network to transform Gaussian noise to the desired distribution. Indeed, the aim of generative models is learning data distribution to generate new data points without having access to the actual data points. It is difficult to learn distribution of data in a way to generate exactly the same data as the real data. However, using neural networks data distribution can be learnt to reduce approximation error between generated and real data.

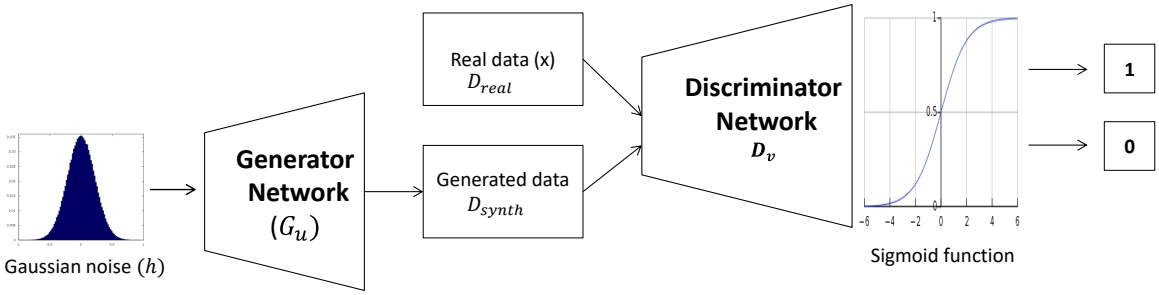



Figure 5.4: *Generative Adversarial Network.*

In this regard, Generative adversarial Networks (GAN) [7] as depicted in Figure 5.4 has been introduced based on the idea of minmax in game theory from Artificial Intelligence (AI) to minimize the possible loss of worst case scenarios (maximum loss).

GAN consists of two networks: Generator (G) and Discriminator (D). The main objective is to find an equilibrium between these two networks. The idea is to transform a random distribution such as Gaussian noise to the real distribution. G and D networks are trained in an adversarial process as a two-players game (minmax theory).

The Generator network does its best to make generated data look like real data to D and Discriminator tries to output 1 on real data and 0 on generated data. Generator does its best to make synthetic inputs look like real inputs to Discriminator.

The objective of GAN as formulated in Equation 5.3, and in its simpler version 5.4, is to estimate the probability of distinguishing between two distributions of $D_v(x)$ and $D_v(G_u(h))$, where $G_u$ is minimizer and $D_v$ is maximizer. In fact, as shown in Equation 5.4, the objective is to get a difference between two expectations. The Back-propagation method is used to update and retrain the model. Backprop updates Discriminator towards more 1s on real inputs and more 0s on synthetic data.

$$min_{u\in U} max_{v\in V}\, \mathbb{E}_{x\sim D_{real}}[\log D_v(x)] + \mathbb{E}_{h\sim D_h}[\log(1 - D_v(G_u(h)))] \tag{5.3}$$

$$min_{u\in U} max_{v\in V}\, \mathbb{E}_{x\sim D_{real}}[D_v(x)] - \mathbb{E}_h[D_v(G_u(h))] \tag{5.4}$$

where u and v are trainable parameters of generator and discriminator nets respectively and $h$ is Gaussian noise.

Simply speaking and as shown in Algorithm 4, with an assumption of using GAN in image enhancement, GAN works as follows. A Gaussian noise is given to the GAN network to generate some synthetic image data, then the actual image and generated image (fake image) are given to another network called discriminator to estimate the difference between the fake and real image using sigmoid function. In fact, the two networks (G) and (D) compete against each other. The generator makes fake data to pass to the discriminator. The discriminator sees the real data and predicts if the received data from the generator is real or fake. By looking at the output of sigmoid function, GAN enhances the synthetic/fake image and make it closer to the real image by back-propagation and retrains the model until it converges as shown in Figure 5.5. Hence, the discriminator can not distinguish any difference between real and fake data. Thereby, generator will win the competition, such as in all two player games. The

**Algorithm 4** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator,$k$, is hyper-parameter.(Source [7])

1: **for** number of training iterations **do**
2: **for** $K$ steps **do**
3: Sample minibatch of m noise samples $\{h^{(1)},\cdots,h^{(m)}\}$ from noise prior $p_g(h)$
4: Sample minibatch of $m$ examples $\{x^{(1)},\cdots,x^{(m)}\}$ from generating distribution $p_{data}(x)$
5: Update the discriminator by ascending its stochastic gradient:

$$\bigtriangledown_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}[\log D(x^{(i)})+\log(1-D(G(h^{(i)})))] \tag{5.5}$$

6: **end for**
7: Sample minibatch of $m$ noise samples $\{h^{(1)},\cdots,h^{(m)}\}$ from noise prior $p_g(h)$.
8: update the generator by descending its stochastic gradient:

$$\bigtriangledown_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log(1-D(G(h^{(i)}))) \tag{5.6}$$

9: **end for**

dotted black line in Figure 5.5 is real/actual data, the dashed blue line is discriminative distribution and the green line is generative distribution. The discriminative distribution distinguishes between black and green lines until it can not differentiate between these two distributions as shown in the right plot of Figure 5.5, where GAN converges.

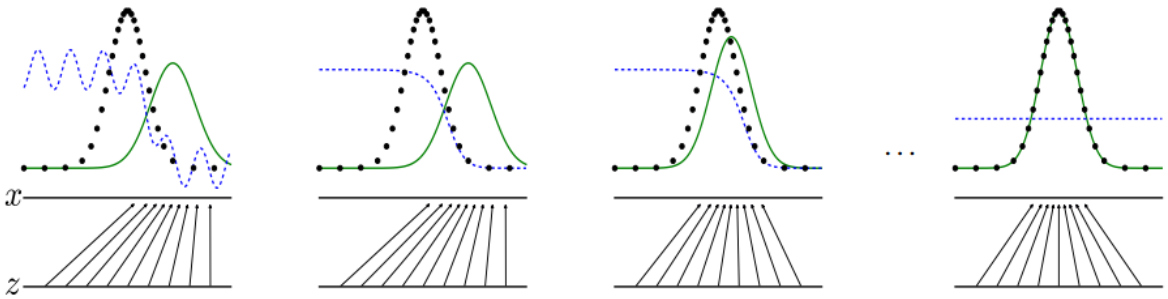



Figure 5.5: *Iterations of Generative Adversarial Network (source [7]).*

### 5.4.4 Regression Model

In statistics, regression analysis is used to estimate the relationships among different variables in a data set. Regression analysis finds a model among a dependent variable and a set of independent variables. This model is defined as a linear or non-linear function to represent how changes in the independent variables influence the dependent variable. In other words, regression analysis fits a curve on data points to discover a correlation among different dimensions/attributes/columns within a data set which is mostly used for prediction and forecasting.
In a regression equation (i.e. linear regression) for a given set of data points $\{y_i, x_{i1}, \cdots, x_{im}\}$, where $i = 1:n$, dependent variable $y_i$ is a linear combination of one or a set of independent variables of $x_i$ as shown in Equation 5.7.

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_m x_{im} + \epsilon_i = X_i^T \beta + \epsilon_i, i = 1, \cdots, n. \tag{5.7}$$

The best fitted curve or line is the one that has minimum distances to data points. One of the metrics to measure goodness of a fitted model is R-squared ($R^2$).

R-squared or coefficient of determination measures closeness of data points to the fitted regression curve/line. $R^2$ represents how much percentage of the response variable variation has been described by a regression model. Equation 5.8 shows the R-squared formula.

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \tag{5.8}$$

where $SST = \sum(y - \bar{y})^2$ is the sum of squares, $SSR = \sum(\acute{y} - \bar{\acute{y}})^2$ is the sum of regressions, and $SSE = \sum(y - \acute{y})^2$ is the sum of errors. $\acute{y}$ is the estimation of y and $\bar{y}$ is the mean of y.

Simply speaking, R-squared is a fraction of the described variation over the total variation that is always between 0 and 100%. 0 means that the model does not explain any variation in data and 100 means that model perfectly explains all the variability in the data around its mean. Therefore, usually the higher the R-squared, the better the model fits data.

## 5.5 The iDMS Framework

A $Kd$-tree can be viewed as representative of different granularity of equi-height histograms at each different level. An equi-height histogram is a type of histogram in which frequency of appearance of data points/items in all bins/buckets are equal; likewise, uniform distribution is shown in Figure 5.6 ,while intervals are changing and need to be continuously updated by arriving new data points in order to maintain the equi-height histogram. In multidimensional space, building and maintaining an equi-height histogram is not an easy task with respect to its wide applicability such as in range queries problems. Indeed, a $kd$-tree maintains and updates intervals of an equi-height histogram in multidimensional space. For instance, at level 2 of Figure 5.2, all 4 partitions (i.e. buckets/bins) have equal height of 2 (i.e. the number of data points at each partition) like ($P_3$,$P_4$) at the left upper corner are the same as ($P_5$, $P_6$), ($P_1$, $P_2$), ($P_7$, $P_8$).

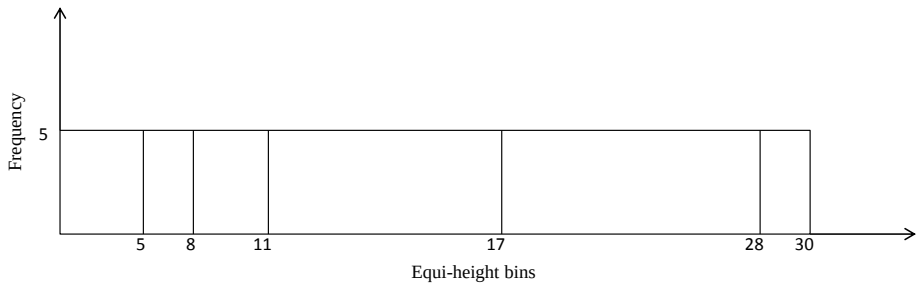



Figure 5.6: *Example of* 1$d$*-equi-height histogram*

To have an overall view of all local histograms in a central site, we also need to build a global $kd$-tree by merging local $kd$-trees at the central site. To achieve this, each local site needs to send its local $kd$-tree to the central site. Received kd-trees are then aggregated or merged in the central site to build a global $kd$-tree. However, it is impractical to merge even *two* general $kd$-trees. Therefore the apparent solution is by having all data points transmitted to the central site. However, in a distributed setting this causes unacceptable communication costs.

Instead, we tackle the problem from the perspective of data approximation techniques. In order to construct the global tree from the union of local $kd$-trees while minimizing the communication cost (or preserving privacy), the idea is to adopt either GMMs or GANS as the unified fitting models and generative models of the local data distributions. GMMs vectors with their learned parameters represent statistical summaries of the underlying data distribution. GANs neural networks also learn the actual/real data distribution

in order to reconstruct distributions that are very close to the actual distributions. The reconstruction process for iDMS-GMM based or iDMS-GAN based simply progress as follows.

**iDMS-GMM/GAN Algorithms:**

1. As shown in Algorithm 5, at a local site, for each chunk of incoming data, either its GMMs are learned with prescribed cluster size or GAN learns distribution of the chunk.

2. Local sites send the learned GMM vectors or trained GAN nets for each chunk of data as local summaries to the central site instead of sending all arrived data points in the stream.

3. As shown in Algorithm 6, at the central site, new data points are generated from either the received GMM vectors/distribution summaries or GAN nets first and the approximate global $kd$-tree is constructed or maintained from the generated/approximated data points from GMM vectors or GAN nets.

**Algorithm 5** Data summarization at a local site

**INPUT:** Data points of a chunk of a local site
**OUTPUT:** Cluster feature vectors (CFVs) of a local site or trained GAN nets

**if** method = GMM **then**
  $numClusters, clusterSizes, means, covs \leftarrow GMM(datapoints)$
  **for** $i := 1$ **to** $numClusters$ **do**
    $CFV_i \leftarrow clusterSize_i, mean_i, Cov_i$
  **end for**
**else if** method = GAN **then**
  $trainedGANnet \leftarrow GAN(datapoints)$
**else**
  no summarization and send all actual data points
**end if**
Send $CFV_i$'s or $GANs$ to the central site.

**Algorithm 6** One round construction & maintenance of Kd-tree at central site

**INPUT:** Received CFVs/GAN nets from local sites
**OUTPUT:** Build or Update Kd-tree at the central site

**if** summary=CFV **then**
  **for** $CFV$ **in** $CFVs$ **do**
    $generatedData \leftarrow multivariate(CFV)$
  **end for**
**else**
  **for** $GAN$ **in** $GANs$ **do**
    $GaussianNoise \leftarrow generate(GaussianNoise)$
    $generatedData \leftarrow GAN(GaussianNoise)$
  **end for**
**end if**
**if** CFVs/GANs are the first chunks **then**
  Build $kd$-tree on Generated data
**else**
  $kd-tree \leftarrow insert(generatedData)$
  Rebalance kd-tree if necessary
**end if**

In our practical framework iDMS (as shown in Figure 5.7), by having distribution properties as either GMM vectors enriched by the number of data points $N$ in each cluster or trained GAN nets at central site, a set of $N$ random data points from this distribution can be generated as an approximation of actual data points from local sites. Specifically, in iDMS-GMM, every site sends the parameters of its distribution along with the number of data points within each distribution to the central site. These statistical summaries are represented as a vector of $\overrightarrow{CFV} = (\mu, \sum, N)$ , where $\mu$ is a $d$-dimensional mean vector, $\sum$ is the associated co-variance matrix and $N$ is the number of data points. While in iDMS-GAN, only trained GAN networks are sent to the central site and in the central site only random Gaussian noises are generated as inputs to the received GAN nets in order to generate data that are very close to the actual data at local sites. It should be noticed that GAN means only trained Generator nets are sent to the central site and there is no need to send Discriminator networks.

Figure 5.7 displays the overview of the framework with fitting and generative models. Statistical summaries are extracted from local $kd$-trees using distribution fitting models. These summaries are sent to the central site as $\overrightarrow{CFV}$ vectors or Generative nets. In the central site, either a generative model such as GMMs or trained Generative networks are utilized for generating random data points drawn from the local $kd$-trees distribu-

tions using $\overrightarrow{CFV}$ vectors or GANs. After the initial data generation, a global $kd$-tree is constructed and maintained on top of the newly generated data.

In iDMS-GMM, similar to $k$-means that deciding the number of GMM clusters is often a challenge. Figure 5.7 shows the varying cluster sizes (the number of sent vectors) at local sites. For instance, if the cluster size is 3, then 3 $\overrightarrow{CFV}$ vectors will be sent to the central site. Deciding the proper number of clusters is helpful for the data generative model to produce more accurate data points at the central site which eventually leads to a better estimated global $kd$-tree. In iDMS-GAN also choosing the learning rate and number of layers are effective in training a GAN network.

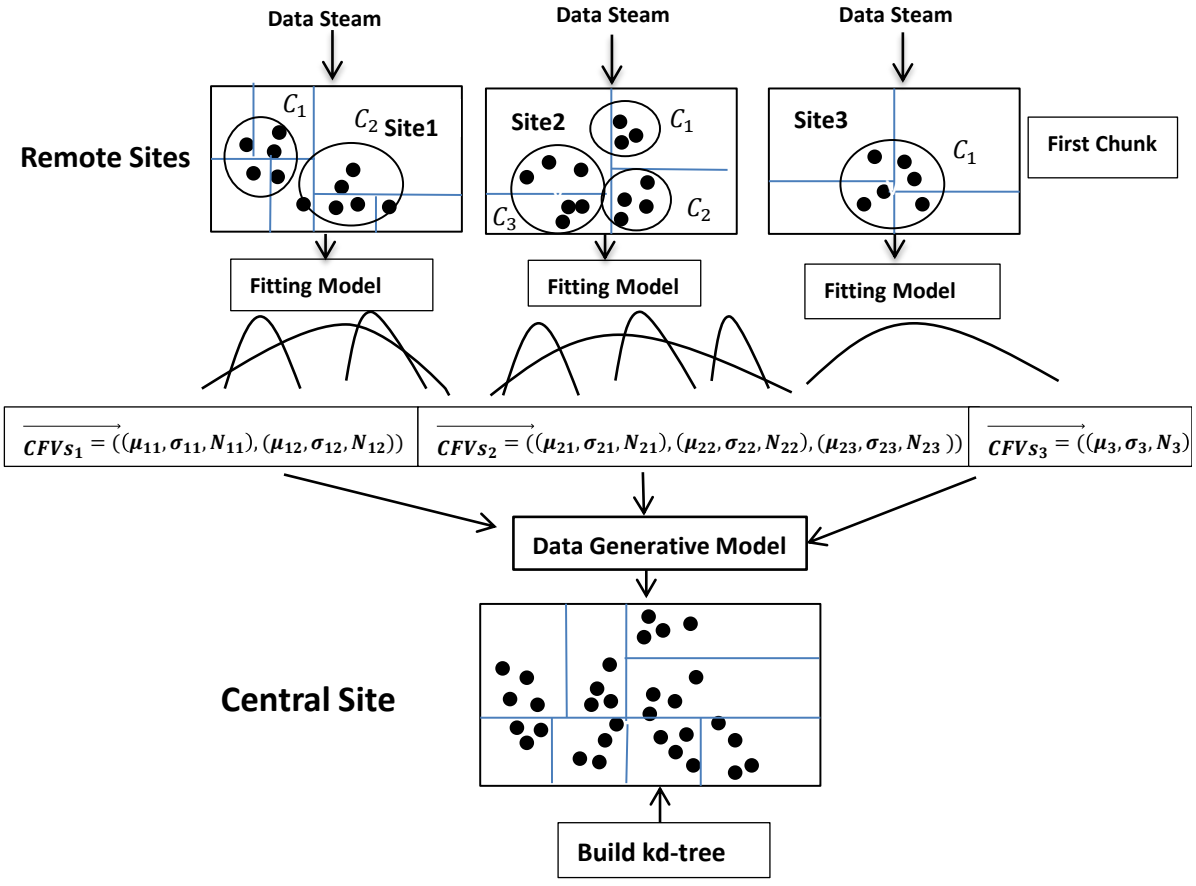



Figure 5.7: *Overview of iDMS framework*

## 5.6 Experimental Results

In this section, we report experimental studies on the proposed framework iDMS. We evaluate iDMS in terms of the approximation quality of the constructed global $kd$-tree from received GMM distribution vectors and GAN models, and the communication cost in sending vectors and trained GAN networks as data summaries. We compare these results from iDMS against a baseline distributed framework where all incoming data points at local sites are sent to the central site, i.e. the central site has all data points to construct the exact global $kd$-tree. We also compare differences between actual data and generated data using GAN and GMM. We show the effect of number of clusters on data approximation error when using GMM.

The framework is tested on two real data sets of California Housing Prices [240] and Wine data [241].

- Housing data set has nine numerical attributes (dimensions/variables) and one categorical attribute with 17000 instances (rows) and we only use 9 numeric attributes.
- The other data set is a wine data set which has 12 numerical attributes and 3,919 instances.

The results are reported on chunk-size, bin-size, number of sites, number of clusters and number of dimensions of data points (to be explained in detail later). Results reported are on average numbers from 10 runs.

We are not aware of any work in the literature that merges and maintains a $kd$-tree to maintain equi-height histograms in distributed streaming environment for further comparison. Therefore, we compare results of GMM with GAN, and also each of them with baseline where all data points are available in the central site.

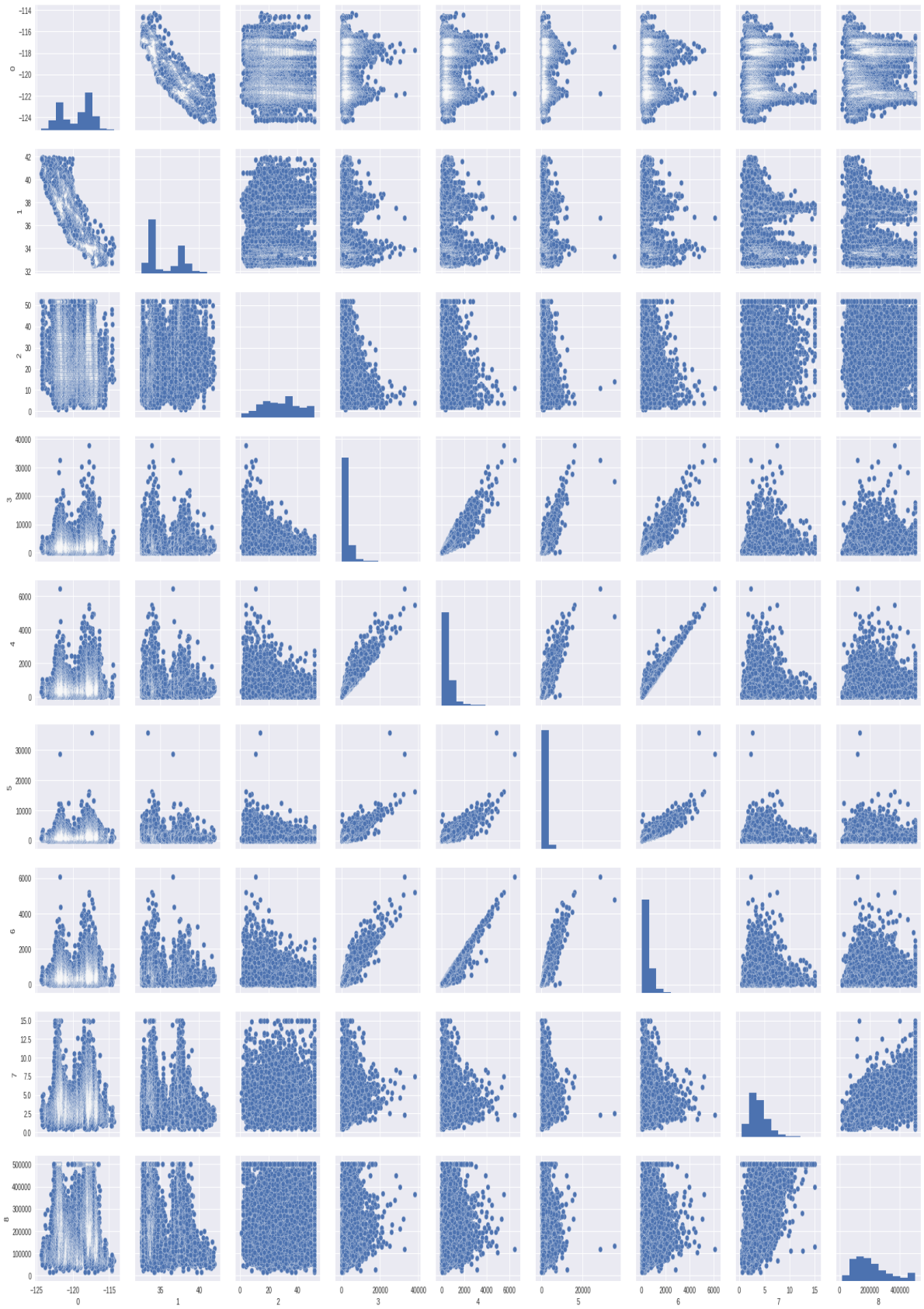


Figure 5.8: *Scatter plots of original housing data on 9 dimensions.*

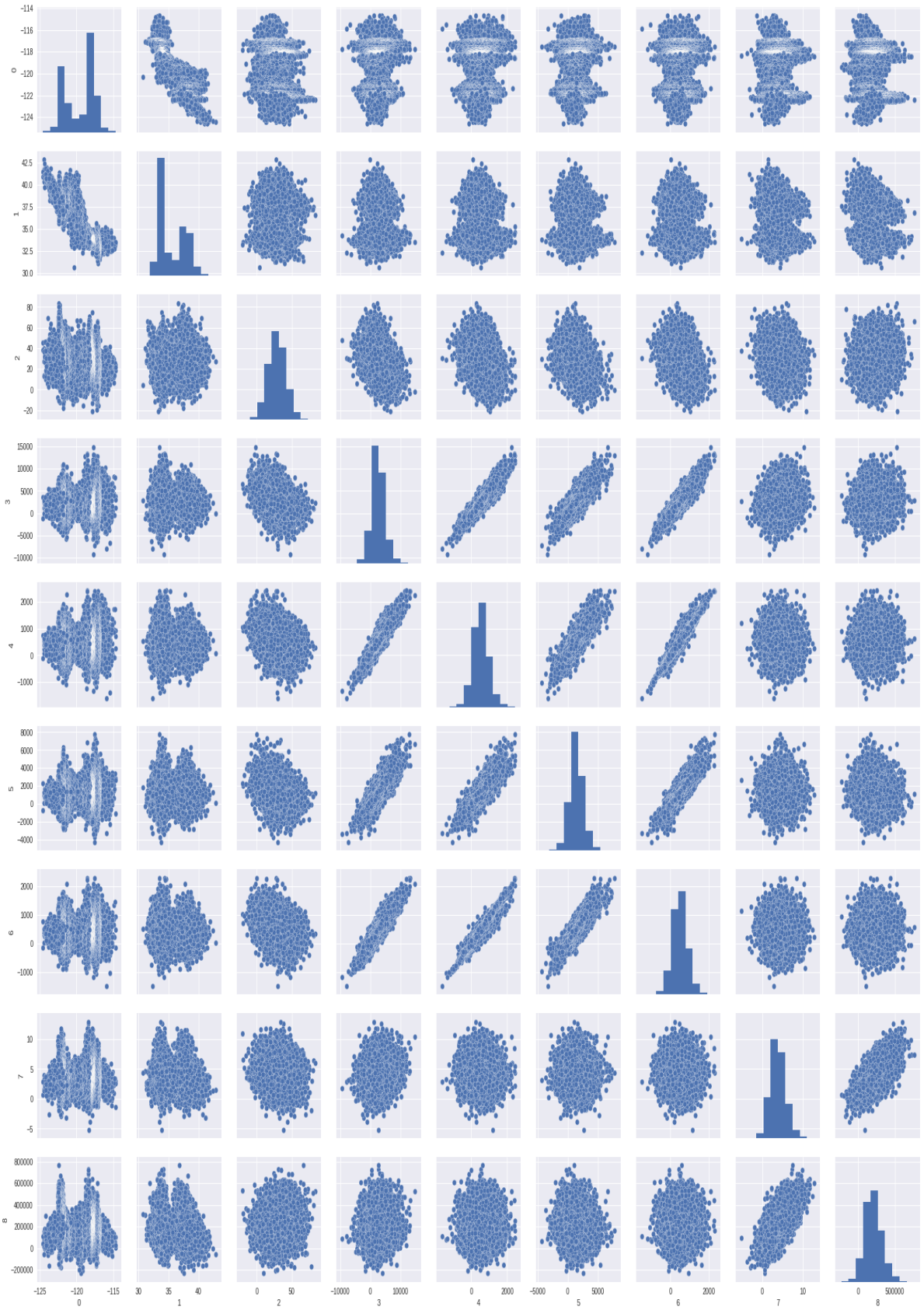


Figure 5.9: *Scatter plots of generated housing data set using GMM on 9 dimensions.*

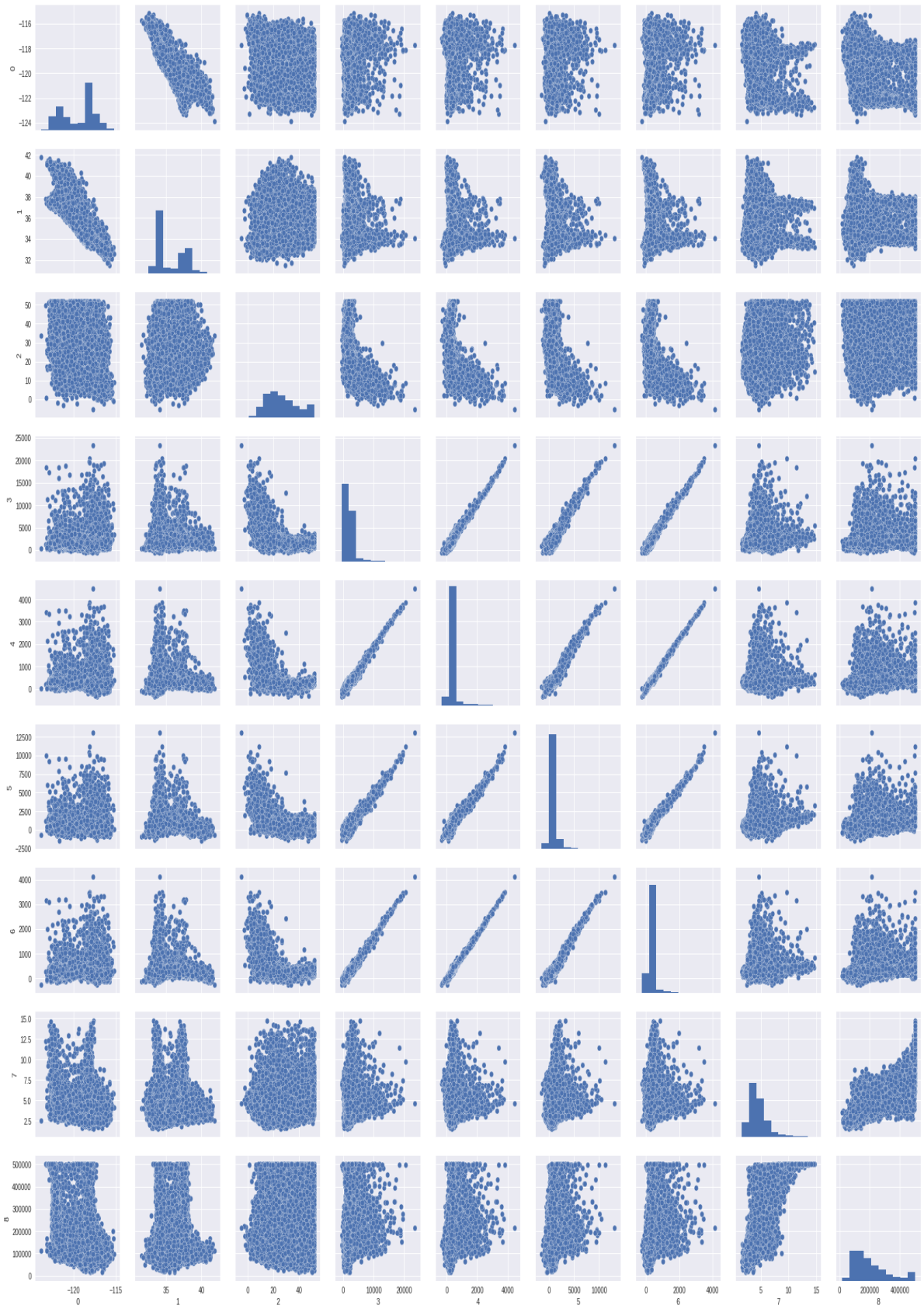


Figure 5.10: *Scatter plots of generated housing data set using GAN on 9 dimensions.*

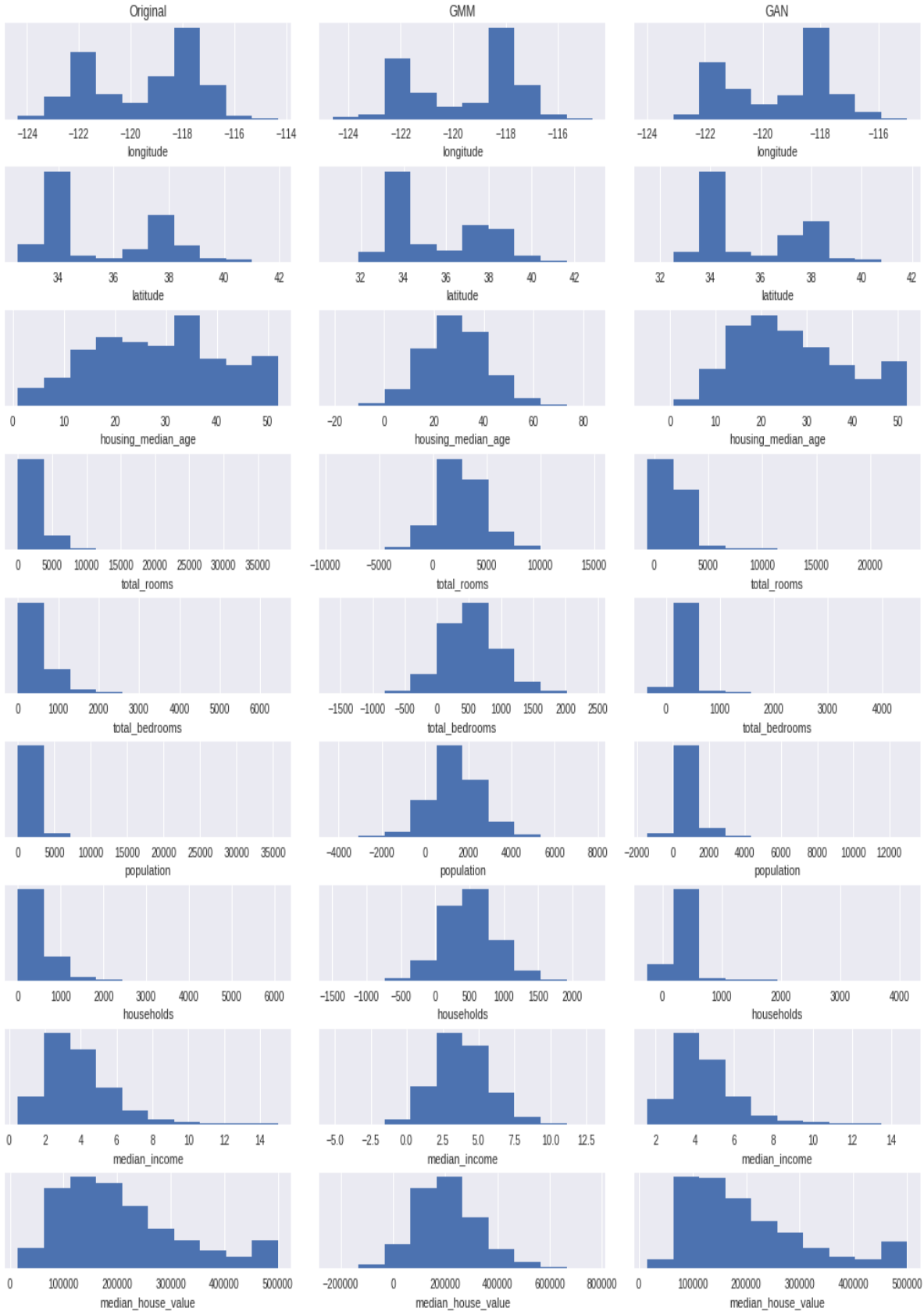



Figure 5.11: *Histograms of generated GMM /GAN vs actual Housing data on 9 dimensions.*

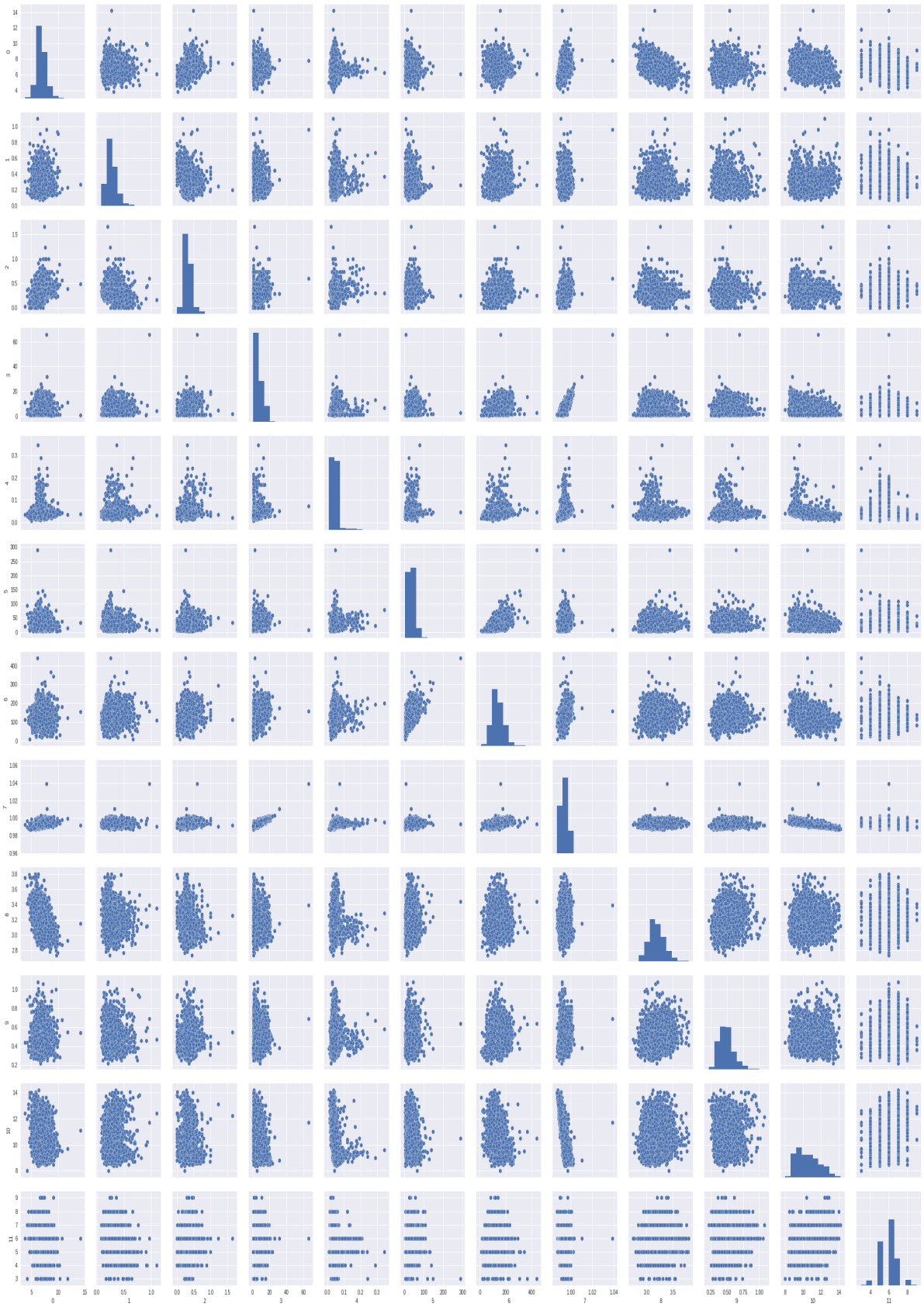


Figure 5.12: *Scatter plots of original Wine data set on 12 dimensions.*

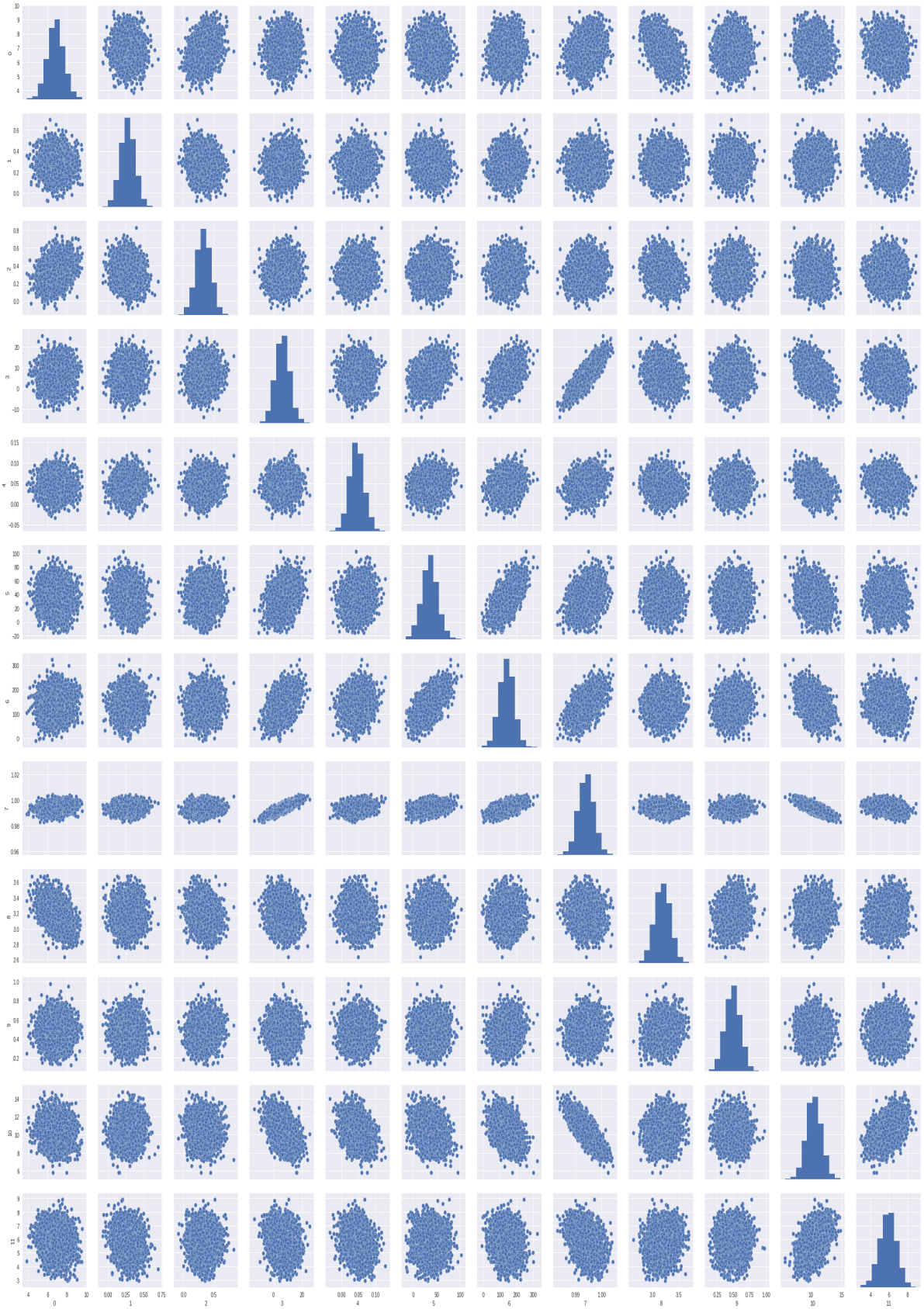


Figure 5.13: *Scatter plots of generated Wine data set using GMM on 12 dimensions.*

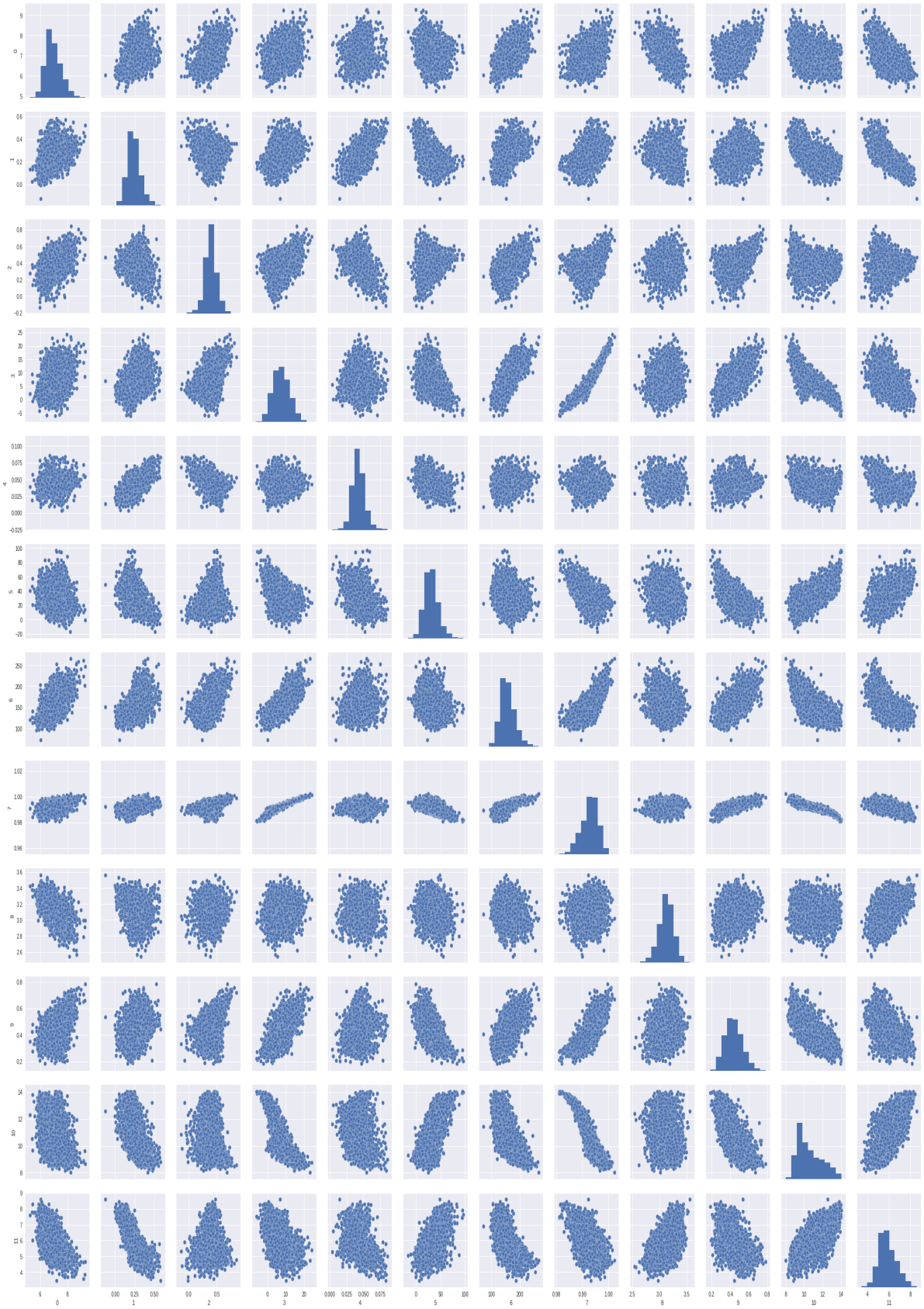


Figure 5.14: *Scatter plots of generated wine data set using GAN on 12 dimensions.*


Figure 5.15: *Histograms of generated GMM /GAN vs actual data on 12 dimensions of Wine data set.*

**Approximation quality** We plot pairwise correlations of attributes/dimensions of actual, GMM and GAN generated data in Figures [5.8, 5.9, 5.10] and [5.12,5.13, 5.14], for both house and wine data sets respectively.

As can be seen, in generated data using GAN, correlations between each pair of dimensions/attributes have been preserved much better than GMM on both data sets. However, relative relationships between dimensions/attributes of GAN generated housing are preserved much better than the wine data set. We believe that the reason for this trend is because of high sparsity and density in the wine data set in addition to size of data set in comparison with the housing data as plotted in Figures 5.12 and 5.14.

We also plot histograms of each dimension of house and wine data sets as shown in Figures 5.11 and 5.15. Histograms of GAN are closer to the histograms of actual data compared to histograms of GMM. GMM generates data within a normal distribution while GAN learns actual distribution of data and generates data that are close to the actual data.

We also use $R^2$ error to measure closeness of generated data to a regression model built on the actual data. We build a regression model on the entire actual data set to evaluate how close the generated data are to the fitted regression model.

R-squared is the percentage of the response variable variation from a trained regression model. It varies between 0 and 100% in which 0% means that the model does not capture any variability of the response data while 100% means that all the variability of the data are captured by the model.

We measure $R^2$ between the actual data sets and the trained regression models over the entire data sets as a baseline. The obtained $R^2$ errors are 87% and 77% for housing and wine data sets respectively, which show the variability of these data sets. We then generate several data sets from GMM by varying the number of clusters ($nc$) as shown in Figure 5.18 and find that the closest generated data with 7 clusters to the regression models have $R^2$ values of 75% on housing data and 62% on wine data (an indication of closeness where 100% is the perfect scenario) as shown in Figure 5.16.

We also generate data sets from GAN for 500 epochs and find that the generated data sets have $R^2$ values of 81% on housing and 69% on wine data set as shown in Figure 5.16. These results indicate that generated data using GAN are closer to the actual data compared to generated data using GMM for both data sets.

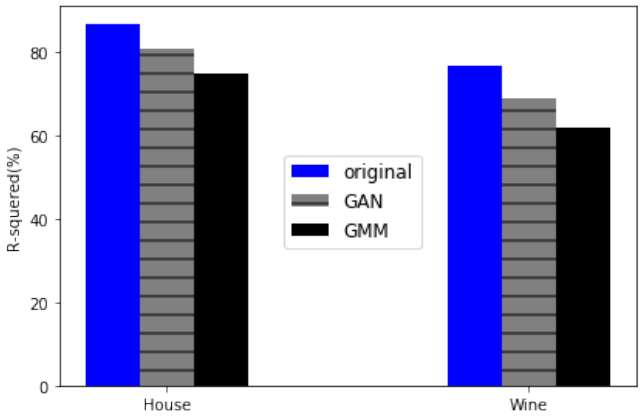



Figure 5.16: $R^2$ *errors of original, GAN and GMM of wine and house data sets.*

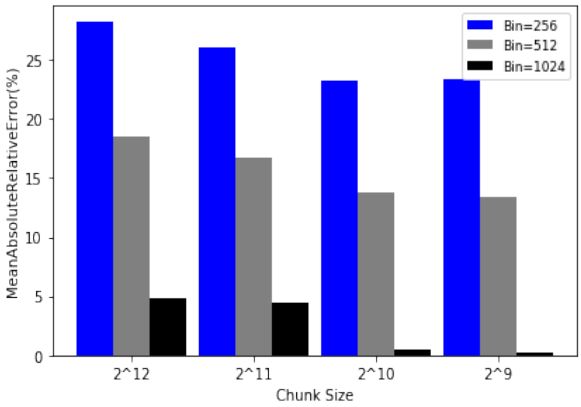
We use the mean absolute relative error (MARE) in Equation 5.9 for calculating the approximation error between medians of constructed $kd$-tree by generated data using GAN and GMM with the best number of clusters and the constructed $kd$-tree with original data.

$$MARE = \frac{1}{M}\sum_{i=1}^{M}\left|\frac{m_i - m'_i}{m_i}\right| \tag{5.9}$$

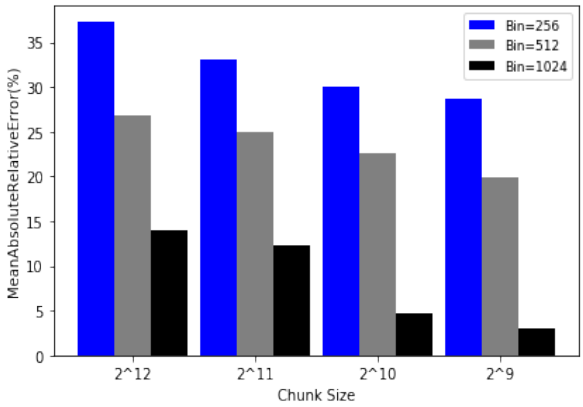
MARE measures the average difference between median values $m_i$ in the exact $kd$-tree and the corresponding medians $m'_i$ in the approximate $kd$-tree built from GMMs or GANs where $M$ denotes the total number of medians.

Figures 5.17a and 5.17b display MARE values between constructed exact and approximate global $kd$-trees for different bin (i.e.leaf) sizes of 256, 512, 1024 and chunk-sizes of $2^{12}$, $2^{11}$, $2^{10}$ and $2^9$ on housing and wine data sets. We chose this range of numbers to be able to limit the size of $kd$-tree to our predefined bin-sizes. Chunk-size is the total number of data points received at the central site and the exact $kd$-tree is purely built on these. Bin-size represents the maximum number of data points each tree leaf contains (i.e. the smallest partition size in space as bin/bucket in histograms).

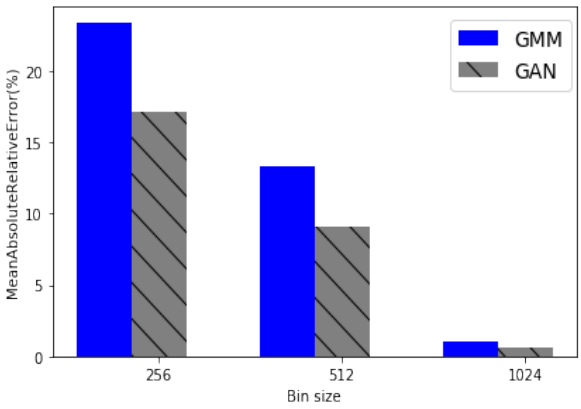
Results show that increasing chunk-size considerably increases MARE irrespective of the bin-sizes. These figures show that by increasing variability of data, GMM estimates cluster distributions and not actual data. On the other hand, increasing leaf/bin size and hence decreasing tree height reduces MARE. This is expected as $kd$-tree decomposition propagates median error from root to leaves. Also, a larger leaf size translates to a larger partition/sample size which leads to better estimated medians from GMMs through generating a set of estimated data points that are closer to the actual data points.



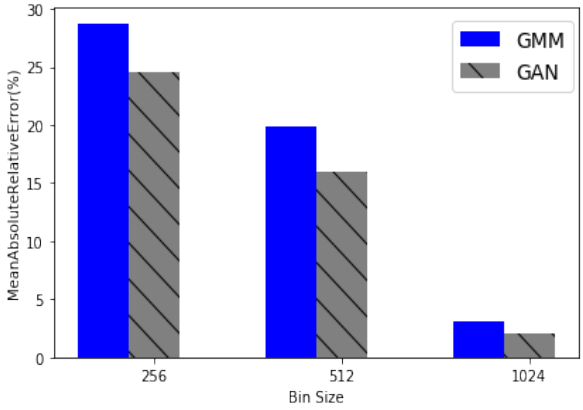

(a) *MAREs between GMM and original data on different chunk and bin-sizes of housing data.*



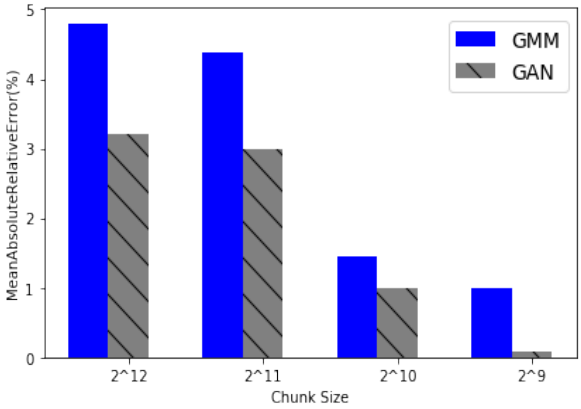

(b) *MAREs between GMM and original data on different chunk and bin-sizes of wine data.*



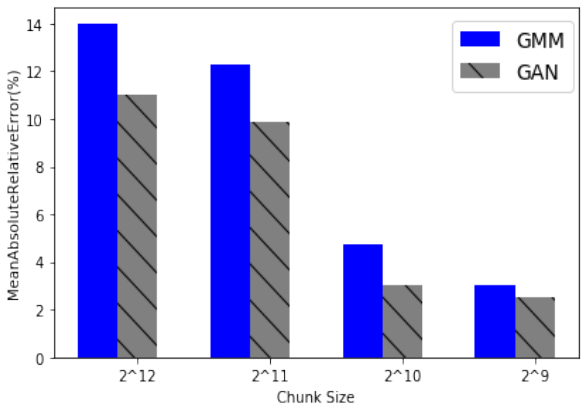

(c) *MAREs comparison of GAN and GMM on diferent bin-sizes, chunk-size=$2^9$, housingdata.*




(d) *MAREs comparison of GAN and GMM on different bin-sizes, chunk-size=$2^9$, winedata.*




(e) *MAREs comparison of GAN and GMM on different chunk-sizes and bin-size=1024, housing data.*




(f) *MAREs comparison of GAN and GMM on diferent chunk-sizes for bin-size=1024, wine data.*

Figure 5.17: *Comparison of constructed $kd$-tree using GMM or GAN generated data with original data.*

From the above experiments we can conclude the best bin and chunk-sizes are 1024 and $2^9$, respectively. Hence, we evaluate and compare results of constructed $kd$-tree built on GMM and GAN data generated for chunk-size of $2^9$ and bin-sizes of 256, 512 and 1024. Figures 5.17c and 5.17d show that MAREs of constructed $kd$-tree using GAN is significantly less than GMM for different bin-sizes of 256, 512 and 1024 for both housing and wine data sets.

We also compare results of constructed $kd$-tree using GMM and GAN for different chunk-sizes of $2^{12}$, $2^{11}$, $2^{10}$ and $2^9$, when bin-size is 1024 as shown in Figures 5.17e and 5.17f. As can be seen, these results also indicate that GAN generates data that are closer to the actual data; therefore for both data sets with different distributions (i.e. distribution of wine data is smoother and skewed compared to house data) GAN ends with less MARE error than the GMM-based $kd$-tree. As stated earlier, GMM generates normal distribution regardless of type of distribution- whether it is normal or skewed. When data distribution is normal, median and mean are almost the same while skewness in data distribution increases median and mean are getting further away from each other.

**Effect of the actual number of clusters on GMM data generation** We test the effect of varying number of clusters over several created data sets from GMM for both housing and wine data sets. As Figure 5.18 shows, increasing the number of actual clusters decreases $R^2$ error of generated data to the actual data. This results in generating closer data to the actual data.

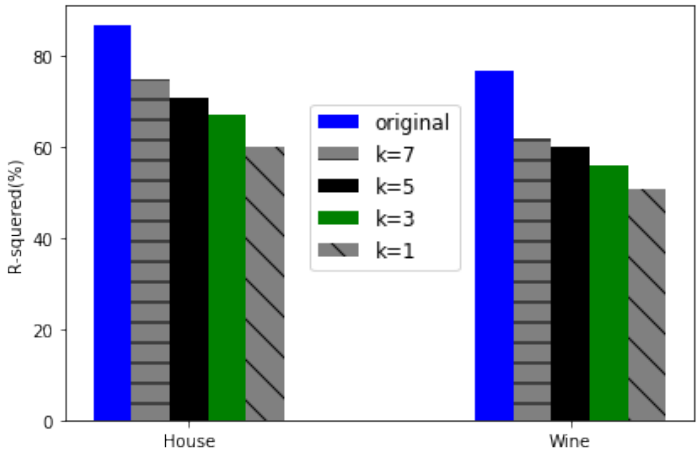
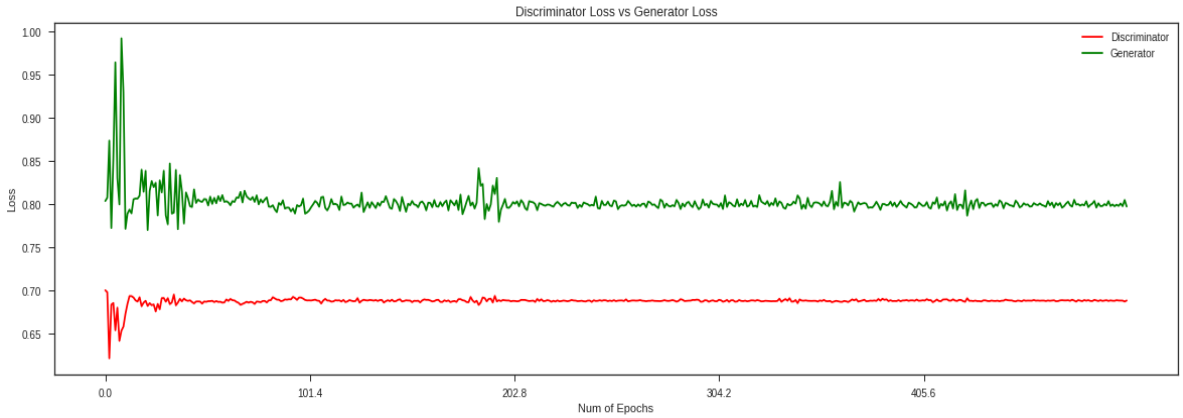


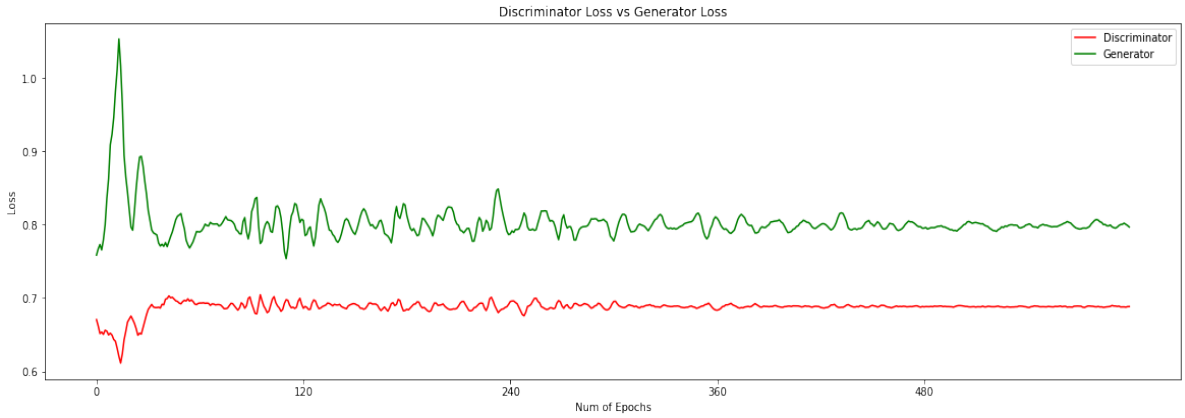

Figure 5.18: *Effect of varying number of clusters on GMM data generation.*

**Accuracy and Loss of Discriminator vs Generator in GAN** We implemented GAN in keras [242] with ADAM optimizer, learning rate (lr) of 0.001 and the network of 3 layers.

Plots of Figure 5.19 show the variation of losses when Generator and Discriminator networks are competing against each other to train GAN net at 500 and 600 epochs for housing and wine datasets, respectively. As these plots depict, the generator loss in both data sets is decreasing and the discriminator losses are increasing. That means that as generators improve, that results in higher losses in discriminator. In both plots, after a few steps/epochs of training, losses of generator and discriminator remain steady. This loss plateau denotes that the GAN model has found some optimum and can not improve further: the model has learned enough.



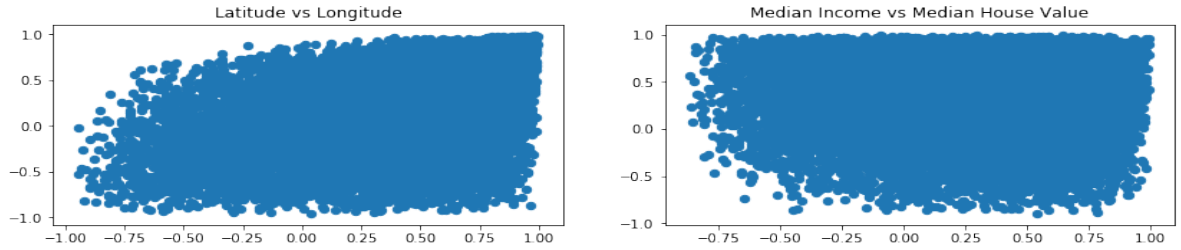

(a) *Housing dataset.*



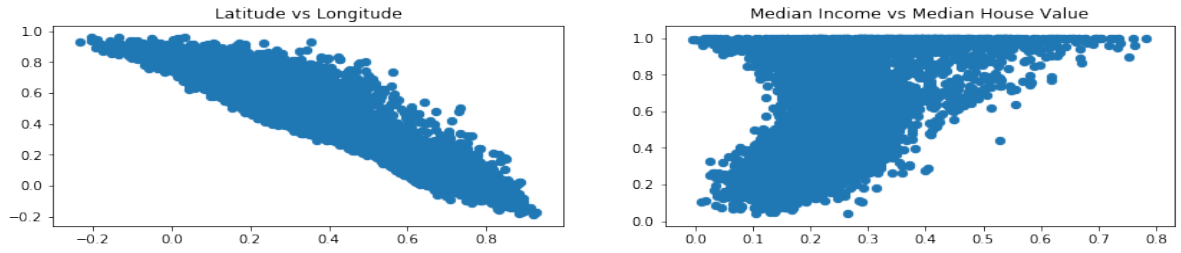

(b) *Wine dataset.*

Figure 5.19: *Plot showing the variation of losses while training the GAN using Adam optimizer at different epochs for housing and wine datasets.*

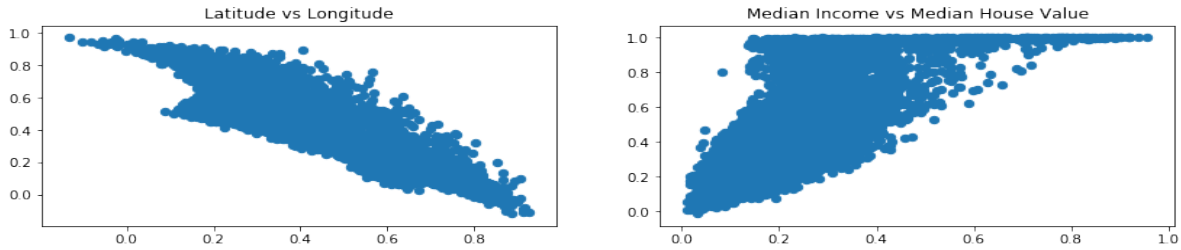
Figure 5.20 is also another demonstration of improvements of GAN on 4 dimensions of the housing dataset at different training epochs of 1, 300, and 500. As can be seen, as the number of epochs increases, a random distribution at plot (a) becomes closer to the original data (d).



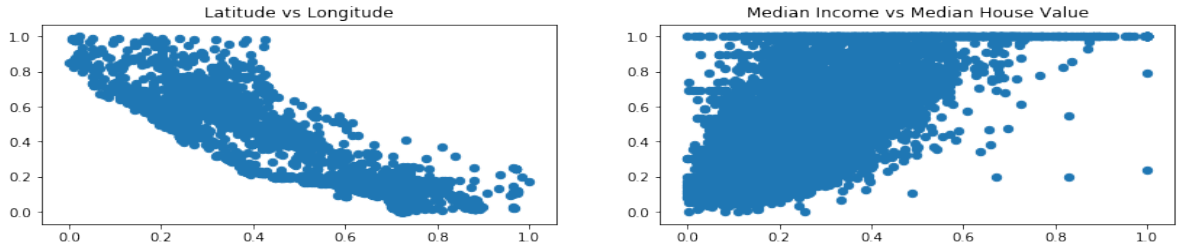

(a) *epoch 1.*




(b) *epoch 300.*




(c) *epoch 500.*

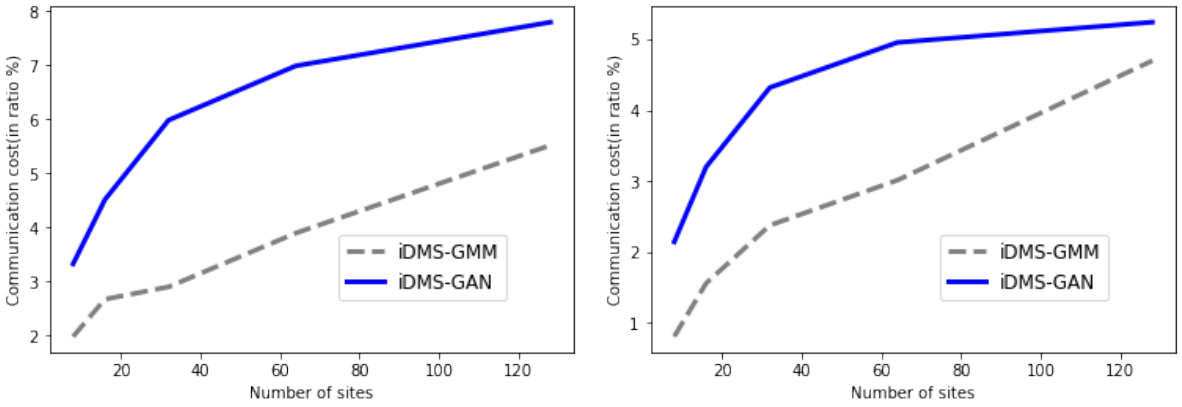



(d) *original data.*

Figure 5.20: *Scatter Plot of distribution of original housing data vs generated housing data on 4 dimensions using GAN at different epochs of 1, 300 and 500.*

**Communication cost** We measured the communication cost in terms of the total number of bytes transmitted by either iDMS-GMM or iDMS-GAN frameworks.

Figure 5.21 compares communication ratio of iDMS-GMM with iDMS-GAN based on the baseline distributed model for housing data set with 9 dimensions and wine data set with 12 dimensions while the number of local sites increase from 4 to 120. These graphs show increasing trends for both iDMS-GMM and iDMS-GAN in both datasets. The changes appear almost linear, but iDMS-GMM are happening at a slower rate than iDMS-GAN. This is expected as in iDMS-GMM only small cluster feature vectors are sent to the central site, while in case of iDMS-GAN, the trained networks are sent which might be different in sizes in terms of number of their nodes, layers and activation functions. In high dimensional spaces, communication costs may be affected by increasing number of dimensions for both iDMS-GMM and iDMS-GAN.




(a) *Wine dataset.* (b) *Housing dataset.*

Figure 5.21: *Comparison of communication ratio of iDMS-GMM and iDMS-GAN with the basic distributed model for different number of sites on (a) wine data set and b) Housing data set.*

## 5.7 Summary

We proposed a novel practical framework iDMS that tracks and approximately merges $kd$-trees leveraging Gaussian mixture models (GMMs) or GAN networks. We experimentally evaluated the proposed framework and the results demonstrate its practicality with low reconstruction error and communication cost while being efficient in comparison with sending all data points to a central site. The results show that the approximation error of data reconstruction in GAN is less than in case of GMM but at the expense of increasing

communication cost in comparison to GMM. GMM is simpler but still needs to decide on the number of clusters, GAN requires careful training parameters, communication cost and dependency on chunk size.

In the next chapter, we will review all works we have undertaken in this thesis and will discuss possible future work(s) on big data summarization.

CHAPTER 6

# CONCLUSION AND FUTURE WORK

This thesis has focused on processing, storage and communication aspects of big data by constructing synopses. Many studies have been conducted to develop and improve summarization techniques. We stress on the above as important challenges of big data summarization. In our work we have proposed different summarization frameworks to reduce limitations of processing, storage and communication. We have focused on three summarization techniques, namely dimensionality reduction, clustering and histograms with respect to their importance and wide applicability in different areas. We studied these challenges in the cases of static and dynamic data, and in centralized and decentralized manners for different application scenarios. In the rest of this chapter we summarize our main contributions, followed by possible interesting research directions that can be pursued in the future.

## 6.1 Summarization of Matrix-based Data

We studied the problem of storage and transmission of large scale medical images. Many medical images such as X-ray images need to be transferred between remote and centralized hospitals for further processing and recognition. However, transmission and storage of these voluminous images are problematic.

We proposed summarization frameworks to extract and capture similarities among a stream of images. We discussed new methods for lossless compression using the concept of memory-assisted universal coding. The proposed approaches are well suited to com-

press large datasets of medical images, especially for recurrent usage. The algorithms consist of a learning phase followed by a testing phase. In the learning phase, dimensionality reduction techniques, whether PCA or NMF as techniques of summarization are performed on training images to extract a set of eigenimages which are used to reconstruct the different test images. The reconstructed images are simply represented (coded) by low dimensional feature vectors. The error (or residual) images are then compressed using traditional lossless compression algorithms such as the CALIC, JPEG-LS, bzip2 and CTW algorithms. Using the JRST database, our experimental results showed that the performance of traditional lossless algorithms can be improved by using the proposed algorithms. The proposed concept of using memory to enhance the performance of universal coders is expected to have a major impact in areas where images exhibit a high correlation.

## 6.2 Multiple Stream Summarization using Clustering

We proposed two multiple stream clustering algorithms using an indexing tree in centralized and distributed models. Our proposed algorithms are based on one of the well-known micro-clustering techniques, ClusTree [4]. The objectives of this study were to minimize the communication cost of clustering distributed data streams and accelerating speed of processing data streams in both a centralized and a decentralized manner. Indeed, data streams cannot be stored or processed in their entirety in a central location and only a summarized form is stored. Data is summarized by means of micro-clustering techniques; such techniques allow local sites to capture as much information as their local storage permits in the form of cluster feature vectors and to store them in a local tree index structure.

In the centralized framework, we proposed a new, anytime, concurrent, multiple stream clustering algorithm. We captured the summary statistics of multiple data streams concurrently in the online phase. We proposed to maintain statistical information of the data locality in micro-clusters at a dynamic, multiple access index data structure for further offline clustering. In the online phase, the index data structure maintains summaries of data in the format of cluster feature tuples ($CF$) instead of storing all incoming objects. Then, the data structure is traversed through an index to insert new

data objects concurrently into their closest micro-clusters. We proposed a centralized micro-clustering algorithm that can concurrently cluster multiple streams.

We also extended ClusTree into DistClusTree, a comprehensive distributed framework for stream clustering. The framework leverages both spatial index summaries and online tracking for balancing communication cost and clustering quality. We demonstrated that DistClusTree efficiently produces clusters as good as its centralized version. DistClusTree is able to reduce communication costs significantly and it is easily configurable in practice according to the requested clustering quality.

## 6.3 Summarization using Histograms

We proposed two frameworks called iDMS-GMM and iDMS-GAN which synchronously construct and maintain a global $kd$-tree from locally distributed streaming data. In these frameworks we used two generative models to compact local data. These local summaries are sent to the central site chunk by chunk. We significantly reduced communication costs by sending summaries instead of actual and unlimited streams of data. At the central site, we generated approximate data from received local summaries using generative models of GMM and GAN. In iDMS-GMM, we only sent a summary vector for each cluster including a number of data points, mean and a covariance matrix. These three features are enough to generate approximate data in the central site. In iDMS-GAN, we sent trained generator networks to the central site in order to regenerate data.

We introduced a novel distributed framework that maintains the global $kd$-tree at the central site from the locally received data. TThe way of precisely merging $kd$-tree indices is deemed to be impossible due to their recursive structures along different dimensions. However, the prominent characteristics of our iDMS framework provide an efficient and effective implementation. We address the fundamental problem of maintaining a global $kd$-tree via a novel distributed framework. Within the framework, we propose GMM/GAN-based distributed and maintenance algorithms that are succinct, effective and efficient. We demonstrate the effectiveness and efficiency of the framework (in terms of $kd$-tree approximation error and communication cost) through experiments on real datasets and against the baseline setting: i.e. all arrived local data points are sent to the central site. We aim to maintain a histogram over union of all local sites at the central site. Frequency counting is the main key in constructing histogram. Hence, we needed to communicate on all data points and not just on cluster features such as center and radius.

## 6.4 Future work

In this thesis we mostly paid attention to summarization of streaming numerical and structured data in distributed environments. However, techniques for distributed summarization of unstructured data such as health records, audio, video and text remain largely unexplored. For example, synopsis construction algorithms have been developed for summarizing unstructured data [243], [244] but with less focus on merging and aggregating them in distributed and parallel manners. Therefore, developing summarization methods that deal with dynamism of data in distributed settings is still an open problem.

Although static and homogeneous data summarizations have been studied, it has not been sufficiently adopted to distributed, dynamic and heterogeneous data. We believe there are open, motivating and interesting research directions in this area as unlimited data are generated from various distributed sources that require union integration and aggregation over their heterogeneous summaries in a central location.

We developed memory-assisted frameworks to extract commonalities among a static set of medical images in an offline manner. Future work can be carried on to study and enhance them, such as a rule-of-thumb for choosing the number of learning levels and investigating the effects of different encoders on the performance of the proposed techniques. Furthermore, MAC frameworks can be extended for online processing of other two-dimensional arrays (i.e. matrix-based data) by using more advance incremental/online dimensionality reduction techniques such as Incremental Principal Component Analysis (IPCA) and extending the centralized memory-assisted compression into the distributed one. It could be worthwhile to generalize the framework by substituting other dimensionality reduction techniques and adopting a variety of medical/non-medical images.

In this work, we only investigated online distributed summarization in a client-server model, where all summaries are aggregated in a local site. More studies could be conducted on distributed summarization in peer-to-peer models. For example, the DistClusTree algorithm could be expanded from a client-server model to a peer to peer model. Additionally, it would be of interest to generalize this framework by applying other index structures, especially distributed index data structures such as VBI [245].

Another interesting research dimension which remains unexplored in the field of distributed stream clustering is an extensive comparison of different existing distributed clustering algorithms in the literature.

We studied the problem of merging kd-trees as representatives of histograms, i.e. summarization, using generative models. However, there is still room to improve the data generation part by replacing GAN with its variations [246]. The framework can also be extended by using other multi-dimensional index structures and non-mergeable summaries. Plugging update mechanisms (e.g. fully-dynamic maintenance) such as delete and insert into the global index-tree is still open for investigation. Moreover, the framework opens up an interesting theoretical problem on how to effective merge index-trees.

# BIBLIOGRAPHY


[1] [Online]. Available: http://mason.gmu.edu/~jtrichil

[2] D. White and R. Jain, "Similarity indexing with the ss-tree," in *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, Feb 1996, pp. 516–523.

[3] [Online]. Available: https://www.slideserve.com/duc/a-framework-for-clustering-evolving-data-streams

[4] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The clustree: indexing micro-clusters for anytime stream mining," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 249–272, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10115-010-0342-8

[5]

[6] D. A. White and R. Jain, "Similarity indexing with the ss-tree," in *Proceedings of the Twelfth International Conference on Data Engineering*, Feb 1996, pp. 516–523.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[8] G. Cormode, "Sketch techniques for approximate query processing."

[9] C. C. Aggarwal and S. Y. Philip, "A survey of synopsis construction in data streams," in *Data Streams*. Springer, 2007, pp. 169–207.

[10] R. J. Hathaway, J. C. Bezdek, and Y. Hu, "Generalized fuzzy c-means clustering strategies using l/sub p/ norm distances," *IEEE Transactions on Fuzzy Systems*, vol. 8, no. 5, pp. 576–582, Oct 2000.

[11] J.MacQueen, “Some methods for classication and analysis of multivariate observations.”

[12] G. A. Carpenter, S. Grossberg, and D. B. Rosen, “Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system,” *Neural Networks*, vol. 4, no. 6, pp. 759 – 771, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809190056B

[13] M. G. Georgios C. Anagnostopoulos, “Ellipsoid art and artmap for incremental unsupervised and supervised learning,” pp. 4390 – 4390 – 12, 2001. [Online]. Available: https://doi.org/10.1117/12.421180

[14] J. Mao and A. K. Jain, “A self-organizing network for hyperellipsoidal clustering (hec),” *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 16–29, Jan 1996.

[15] C. J. V. Rijsbergen, *Information Retrieval*, 2nd ed. Newton, MA, USA: Butterworth-Heinemann, 1979.

[16] J. Czekanowski, *Zur differentialdiagnose der neandertalgruppe*. Friedr. Vieweg & Sohn, 1909.

[17] R. H. Whittaker, “A study of summer foliage insect communities in the great smoky mountains,” *Ecological monographs*, vol. 22, no. 1, pp. 1–44, 1952.

[18] P. Legendre and L. Legendre, “Numerical ecology, volume 24, (developments in environmental modelling),” 1998.

[19] C. Dunn, “Applied multivariate statistical analysis,” 1989.

[20] P. F. RUSSELL, T. R. Rao *et al.*, “On habitat and association of species of anopheline larvae in south-eastern madras.” *Journal of the Malaria Institute of India*, vol. 3, no. 1, 1940.

[21] R. R. Sokal, “A statistical method for evaluating systematic relationship,” *University of Kansas science bulletin*, vol. 28, pp. 1409–1438, 1958.

[22] P. Jaccard, “Étude comparative de la distribution florale dans une portion des alpes et des jura,” *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[23] D. J. Rogers and T. T. Tanimoto, "A computer program for classifying plants," *Science*, vol. 132, no. 3434, pp. 1115–1118, 1960.

[24] S. Kulczynski, "Classe des sciences mathématiques et naturelles," *Bulletin International de l'Academie Polonaise des Sciences et des Lettres*, pp. 57–203, 1927.

[25] J. D. Tubbs, "A note on binary template matching," *Pattern Recognition*, vol. 22, no. 4, pp. 359–365, 1989.

[26] L. Konopnicki and P. Rousseeuw, "Finding groups in data: An introduction to cluster analysis," 1990.

[27] D. S. Wilks, "Cluster analysis," in *International geophysics*. Elsevier, 2011, vol. 100, pp. 603–616.

[28] P. H. Sneath, "The application of computers to taxonomy," *Microbiology*, vol. 17, no. 1, pp. 201–226, 1957.

[29] T. Sprense, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analysis of the vegetation on danish commons," *Biol. Skr*, vol. 5, no. 4, pp. 1–34, 1948.

[30] A. K. Jain and R. C. Dubes, "Algorithms for clustering data," 1988.

[31] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM Sigmod Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.

[32] T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris, "A robust and scalable clustering algorithm for mixed type attributes in large database environment," in *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2001, pp. 263–268.

[33] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French, "Clustering large datasets in arbitrary metric spaces," in *icde*. IEEE, 1999, p. 502.

[34] S. Guha, R. Rastogi, and K. Shim, "Cure: an efficient clustering algorithm for large databases," in *ACM Sigmod Record*, vol. 27, no. 2. ACM, 1998, pp. 73–84.

[35] ——, "Rock: A robust clustering algorithm for categorical attributes," *Information systems*, vol. 25, no. 5, pp. 345–366, 2000.

[36] E. Forgey, "Cluster analysis of multivariate data: Efficiency vs. interpretability of classification," *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.

[37] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[38] S. Acharya, P. B. Gibbons, and V. Poosala, "Congressional samples for approximate answering of group-by queries," in *ACM SIGMOD Record*, vol. 29, no. 2. ACM, 2000, pp. 487–498.

[39] J. Mao and A. K. Jain, "A self-organizing network for hyperellipsoidal clustering (hec)," *Ieee transactions on neural networks*, vol. 7, no. 1, pp. 16–29, 1996.

[40] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," 1973.

[41] W. Peizhuang, "Pattern recognition with fuzzy objective function algorithms (james c. bezdek)," *SIAM Review*, vol. 25, no. 3, p. 442, 1983.

[42] S. Eschrich, J. Ke, L. O. Hall, and D. B. Goldgof, "Fast accurate fuzzy clustering through data reduction," *IEEE transactions on fuzzy systems*, vol. 11, no. 2, pp. 262–270, 2003.

[43] M. Steinbach, G. Karypis, V. Kumar *et al.*, "A comparison of document clustering techniques," in *KDD workshop on text mining*, vol. 400, no. 1. Boston, 2000, pp. 525–526.

[44] D. Pelleg and A. Moore, "Accelerating exact k-means algorithms with geometric reasoning," CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 2000.

[45] D. Pelleg, A. W. Moore *et al.*, "X-means: Extending k-means with efficient estimation of the number of clusters." in *Icml*, vol. 1, 2000, pp. 727–734.

[46] R. Soentpiet *et al.*, *Advances in kernel methods: support vector learning*. MIT press, 1999.

[47] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[48] R. T. Ng and J. Han, "Efficient and effective clustering methods for spatial data mining," in *Proceedings of VLDB*, 1994, pp. 144–155.

[49] 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, no. 34, 1996, pp. 226–231.

[50] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander, "A distribution-based clustering algorithm for mining in large spatial databases," in *Data Engineering, 1998. Proceedings., 14th International Conference on*. IEEE, 1998, pp. 324–331.

[51] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[52] A. Hinneburg, D. A. Keim *et al.*, "An efficient approach to clustering in large multimedia databases with noise," in *KDD*, vol. 98, 1998, pp. 58–65.

[53] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," in *ACM Sigmod record*, vol. 28, no. 2. ACM, 1999, pp. 49–60.

[54] P. Grabusts and A. Borisov, "Using grid-clustering methods in data classification," in *Parallel Computing in Electrical Engineering, 2002. PARELEC'02. Proceedings. International Conference on*. IEEE, 2002, pp. 425–426.

[55] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[56] S. A. Elavarasi, J. Akilandeswari, and B. Sathiyabhama, "A survey on partition clustering algorithms," *International Journal of Enterprise Computing and Business Systems*, vol. 1, no. 1, 2011.

[57] W. Wang, J. Yang, R. Muntz *et al.*, "Sting: A statistical information grid approach to spatial data mining," in *VLDB*, vol. 97, 1997, pp. 186–195.

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

[59] E. Schikuta, "Grid-clustering: An efficient hierarchical clustering method for very large data sets," in *Pattern Recognition, 1996., Proceedings of the 13th International Conference On*, vol. 2. IEEE, 1996, pp. 101–105.

[60] D. Barbará and P. Chen, "Using the fractal dimension to cluster datasets," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 260–264.

[61] A. Hinneburg and D. A. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," 1999.

[62] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications*. ACM, 1998, vol. 27, no. 2.

[63] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping multidimensional data*. Springer, 2006, pp. 25–71.

[64] P. Kaur and S. Aggrawal, "Comparative study of clustering techniques," *International Journal on Advanced Research in Engineering and Technology*, vol. 1, pp. 69–75, 2013.

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

[66] W. G. Cochran, *Sampling Techniques: 3d Ed*. Wiley New York, 1977.

[67] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.

[68] ——, "Faster methods for random sampling," *Communications of the ACM*, vol. 27, no. 7, pp. 703–718, 1984.

[69] J. Zhang, J. Xu, and S. S. Liao, "Sampling methods for summarizing unordered vehicle-to-vehicle data streams," *Transportation Research Part C: Emerging Technologies*, vol. 23, pp. 56–67, 2012.

[70] M. Dash and W. Ng, "Efficient reservoir sampling for transactional data streams," in *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*. IEEE, 2006, pp. 662–666.

[71] G. Zou, "A modified poisson regression approach to prospective studies with binary data," *American journal of epidemiology*, vol. 159, no. 7, pp. 702–706, 2004.

[72] B. Babcock, M. Datar, and R. Motwani, "Sampling from a moving window over streaming data," in *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2002, pp. 633–634.

[73] C. C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 607–618.

[74] R. Gemulla, W. Lehner, and P. J. Haas, "A dip in the reservoir: Maintaining sample synopses of evolving datasets," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 595–606.

[75] P. B. Gibbons and Y. Matias, "New sampling-based summary statistics for improving approximate query answers," in *ACM SIGMOD Record*, vol. 27, no. 2. ACM, 1998, pp. 331–342.

[76] R. Gemulla, W. Lehner, and P. J. Haas, "Maintaining bernoulli samples over evolving multisets," in *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2007, pp. 93–102.

[77] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya, "Overcoming limitations of sampling for aggregation queries," in *Data Engineering, 2001. Proceedings. 17th International Conference on*. IEEE, 2001, pp. 534–542.

[78] C. Hua-Hui and L. Kang-Li, "Weighted random sampling based hierarchical amnesic synopses for data streams," in *Computer Science and Education (ICCSE), 2010 5th International Conference on*. IEEE, 2010, pp. 1816–1820.

[79] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Information Processing Letters*, vol. 97, no. 5, pp. 181–185, 2006.

[80] H.-J. Chang and K.-C. Huang, "Remainder linear systematic sampling," *Sankhyā: The Indian Journal of Statistics, Series B*, pp. 249–256, 2000.

[81] N. Uthayakumaran, "Additional circular systematic sampling methods," *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, vol. 40, no. 4, pp. 467–474, 1998.

[82] C.-H. Leu and F.-F. Kao, "Modified balanced circular systematic sampling," *Statistics & probability letters*, vol. 76, no. 4, pp. 373–383, 2006.

[83] M. A. Bujang, P. Ab Ghani, N. A. Zolkepali, M. M. Ali, T. H. Adnan, S. Selvarajah, and J. Haniff, "Modification of systematic sampling: A comparison with a conventional approach in systematic sampling," in *Statistics in Science, Business, and Engineering (ICSSBE), 2012 International Conference on*. IEEE, 2012, pp. 1–4.

[84] M. Al-Kateb, B. S. Lee, and X. S. Wang, "Adaptive-size reservoir sampling over data streams," in *null*. IEEE, 2007, p. 22.

[85] M. Al-Kateb and B. S. Lee, "Adaptive stratified reservoir sampling over heterogeneous data streams," *Information Systems*, vol. 39, pp. 199–216, 2014.

[86] M. D. Bankier, "Power allocations: determining sample sizes for subnational areas," *The American Statistician*, vol. 42, no. 3, pp. 174–177, 1988.

[87] S. Chaudhuri, G. Das, and V. Narasayya, "Optimized stratified sampling for approximate query processing," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 2, p. 9, 2007.

[88] T. Liu and G. Agrawal, "Stratified k-means clustering over a deep web data source," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 1113–1121.

[89] H. Sug, "A structural sampling technique for better decision trees," in *Intelligent Information and Database Systems, 2009. ACIIDS 2009. First Asian Conference on*. Ieee, 2009, pp. 24–27.

[90] A. Pol, C. Jermaine, and S. Arumugam, "Maintaining very large random samples using the geometric file," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 17, no. 5, pp. 997–1018, 2008.

[91] T. S. Buda, J. Murphy, and M. Kristiansen, "Towards realistic sampling: generating dependencies in a relational database," 2013.

[92] S. Cong, J. Han, J. Hoeflinger, and D. Padua, "A sampling-based framework for parallel data mining," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2005, pp. 255–265.

[93] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic sample selection for approximate query processing," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 539–550.

[94] R. Gemulla, W. Lehner, and P. J. Haas, "Maintaining bounded-size sample synopses of evolving datasets," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 17, no. 2, pp. 173–201, 2008.

[95] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A. I. Verkamo *et al.*, "Fast discovery of association rules." *Advances in knowledge discovery and data mining*, vol. 12, no. 1, pp. 307–328, 1996.

[96] B. Chen, P. Haas, and P. Scheuermann, "A new two-phase sampling based algorithm for discovering association rules," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 462–468.

[97] F. Olken, "Random sampling from databases," Ph.D. dissertation, University of California, Berkeley, 1993.

[98] I. Boxill, C. M. Chambers, and E. Wint, *Introduction to social research: With applications to the Caribbean*. University of The West Indies Press, 1997.

[99] C. Sibona and S. Walczak, "Purposive sampling on twitter: A case study," *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 3510–3519, 01 2012.

[100] D. F. Nettleton, "Data mining of social networks represented as graphs," *Computer Science Review*, vol. 7, pp. 1–34, 2013.

[101] P. Griinwald, "Minimum description length tutorial," *Advances in minimum description length: Theory and applications*, pp. 23–80, 2005.

[102] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[103] P. D. Grünwald, "The minimum description length principle and reasoning under uncertainty," Ph.D. dissertation, Quantum Computing and Advanced System Research, 1998.

[104] J. Kiernan and E. Terzi, "Constructing comprehensive summaries of large event sequences," 2008.

[105] ——, "Constructing comprehensive summaries of large event sequences," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 4, pp. 21:1–21:31, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1631162.1631169

[106] P. Wang, H. Wang, M. Liu, and W. Wang, "An algorithmic approach to event summarization," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 183–194. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807189

[107] Y. Jiang, C.-S. Perng, and T. Li, "Natural event summarization," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, ser. CIKM '11. New York, NY, USA: ACM, 2011, pp. 765–774. [Online]. Available: http://doi.acm.org/10.1145/2063576.2063688

[108] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad, "Depth first generation of long patterns," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 108–118. [Online]. Available: http://doi.acm.org/10.1145/347090.347114

[109] D. Burdick, M. Calimlim, and J. Gehrke, "Mafia: a maximal frequent itemset algorithm for transactional databases," in *Proceedings 17th International Conference on Data Engineering*, April 2001, pp. 443–452.

[110] J. Pei, J. Han, R. Mao *et al.*, "Closet: An efficient algorithm for mining frequent closed itemsets."

[111] W. Zhou, H. Liu, and H. Cheng, "Mining closed episodes from event sequences efficiently," in *Advances in Knowledge Discovery and Data Mining*, M. J. Zaki, J. X. Yu, B. Ravindran, and V. Pudi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 310–318.

[112] A. Siebes, J. Vreeken, and M. van Leeuwen, *Item Sets That Compress*, pp. 395–406. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972764.35

[113] M. van Leeuwen, J. Vreeken, and A. Siebes, "Compression picks item sets that matter," in *Knowledge Discovery in Databases: PKDD 2006*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 585–592.

[114] J. Vreeken, M. van Leeuwen, and A. Siebes, "Krimp: mining itemsets that compress," *Data Mining and Knowledge Discovery*, vol. 23, no. 1, pp. 169–214, Jul 2011. [Online]. Available: https://doi.org/10.1007/s10618-010-0202-x

[115] M. van Leeuwen and A. Siebes, "Streamkrimp: Detecting change in data streams," in *Machine Learning and Knowledge Discovery in Databases*, W. Daelemans, B. Goethals, and K. Morik, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 672–687.

[116] K. Smets and J. Vreeken, "Slim: Directly mining descriptive patterns," in *SDM*, 2012.

[117] N. Tatti and J. Vreeken, "The long and the short of it: Summarising event sequences with serial episodes," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 462–470. [Online]. Available: http://doi.acm.org/10.1145/2339530.2339606

[118] L. Chang, D. Yang, S. Tang, and T. Wang, "Mining compressed sequential patterns," in *Advanced Data Mining and Applications*, X. Li, O. R. Zaïane, and Z. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 761–768.

[119] F. Moerchen, M. Thies, and A. Ultsch, "Efficient mining of all margin-closed itemsets with applications in temporal knowledge discovery and classification by compression," *Knowledge and Information Systems*, vol. 29, no. 1, pp. 55–80, 2011.

[120] R. Polikar *et al.*, "The wavelet tutorial."

[121] G. Strang, "Wavelet transforms versus fourier transforms," *Bulletin of the American Mathematical Society*, vol. 28, no. 2, pp. 288–305, 1993.

[122] A. Haar, "Zur theorie der orthogonalen funktionensysteme," *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, Sep 1910. [Online]. Available: https://doi.org/10.1007/BF01456326

[123] I. Daubechies, *Ten Lectures on Wavelets*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

[124] M. Garofalakis and P. B. Gibbons, "Probabilistic wavelet synopses," *ACM Trans. Database Syst.*, vol. 29, no. 1, pp. 43–90, Mar. 2004. [Online]. Available: http://doi.acm.org/10.1145/974750.974753

[125] Y. Matias, J. S. Vitter, and M. Wang, "Wavelet-based histograms for selectivity estimation," *SIGMOD Rec.*, vol. 27, no. 2, pp. 448–459, Jun. 1998. [Online]. Available: http://doi.acm.org/10.1145/276305.276344

[126] Y. Matias and D. Urieli, "Inner-product based wavelet synopses for range-sum queries," in *Algorithms – ESA 2006*, Y. Azar and T. Erlebach, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 504–515.

[127] J. S. Vitter and M. Wang, "Approximate computation of multidimensional aggregates of sparse data using wavelets," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 193–204. [Online]. Available: http://doi.acm.org/10.1145/304182.304199

[128] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim, "Approximate query processing using wavelets," *The VLDB Journal*, vol. 10, no. 2-3, pp. 199–223, Sep. 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=767141.767147

[129] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surfing wavelets on streams: One-pass summaries for approximate aggregate queries," in *VLDB*, 2001.

[130] D. Sacharidis, A. Deligiannakis, and T. Sellis, "Hierarchically compressed wavelet synopses," *The VLDB Journal*, vol. 18, no. 1, pp. 203–231, Jan 2009. [Online]. Available: https://doi.org/10.1007/s00778-008-0096-z

[131] A. Deligiannakis and N. Roussopoulos, "Extended wavelets for multiple measures," in *Proceedings of the 2003 ACM SIGMOD International Conference*

*on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 229–240. [Online]. Available: http://doi.acm.org/10.1145/872757.872786

[132] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos, "Extended wavelets for multiple measures," *ACM Trans. Database Syst.*, vol. 32, p. 10, 06 2007.

[133] S. Guha, C. Kim, and K. Shim, "Xwave: Approximate extended wavelets for streaming data." 01 2004, pp. 288–299.

[134] S. Guha and B. Harb, "Approximation algorithms for wavelet transform coding of data streams," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, ser. SODA '06. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006, pp. 698–707. [Online]. Available: http://dl.acm.org/citation.cfm?id=1109557.1109633

[135] Y. Matias, J. S. Vitter, and M. Wang, "Dynamic maintenance of wavelet-based histograms," 2000.

[136] G. Cormode, M. Garofalakis, and D. Sacharidis, "Fast approximate wavelet tracking on streams," in *Proceedings of the 10th International Conference on Advances in Database Technology*, ser. EDBT'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 4–22. [Online]. Available: http://dx.doi.org/10.1007/11687238_4

[137] P. Karras and N. Mamoulis, "One-pass wavelet synopses for maximum-error metrics," in *VLDB*, 2005.

[138] L. Kang-Li, C. Hua-Hui, Q. Jiang-Bo, and D. Yi-Hong, "Wavelet decomposition algorithm for uncertain data streams," in *2011 6th International Conference on Computer Science Education (ICCSE)*, Aug 2011, pp. 965–970.

[139] Y. Zhao, C. C. Aggarwal, and P. S. Yu, "On wavelet decomposition of uncertain time series data sets," in *CIKM*, 2010.

[140] J. Jestes, K. Yi, and F. Li, "Building wavelet histograms on large data in mapreduce," *Proc. VLDB Endow.*, vol. 5, no. 2, pp. 109–120, Oct. 2011. [Online]. Available: http://dx.doi.org/10.14778/2078324.2078327

[141] M. Stern, E. Buchmann, and K. Böhm, "A wavelet transform for efficient consolidation of sensor relations with quality guarantees," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 157–168, Aug. 2009. [Online]. Available: https://doi.org/10.14778/1687627.1687646

[142] C. Aggarwal, *Data Streams: Models and Algorithms*, 01 2007, vol. 31.

[143] R. P. Kooi, "The optimization of queries in relational databases," Ph.D. dissertation, Cleveland, OH, USA, 1980, aAI8109596.

[144] M. Muralikrishna and D. J. DeWitt, "Equi-depth histograms for estimating selectivity factors for multi-dimensional queries," in *SIGMOD Conference*, 1988.

[145] Y. E. Ioannidis and V. Poosala, "Balancing histogram optimality and practicality for query result size estimation," *SIGMOD Rec.*, vol. 24, no. 2, pp. 233–244, May 1995. [Online]. Available: http://doi.acm.org/10.1145/568271.223841

[146] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita, "Improved histograms for selectivity estimation of range predicates," *SIGMOD Rec.*, vol. 25, no. 2, pp. 294–305, Jun. 1996. [Online]. Available: http://doi.acm.org/10.1145/235968.233342

[147] A. C. König and G. Weikum, "Combining histograms and parametric curve fitting for feedback-driven query result-size estimation," in *VLDB*, 1999.

[148] V. Poosala and Y. E. Ioannidis, "Selectivity estimation without the attribute value independence assumption," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, ser. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 486–495. [Online]. Available: http://dl.acm.org/citation.cfm?id=645923.673638

[149] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi, "Approximating multi-dimensional aggregate range queries over real attributes," *SIGMOD Rec.*, vol. 29, no. 2, pp. 463–474, May 2000. [Online]. Available: http://doi.acm.org/10.1145/335191.335448

[150] N. Bruno and S. Chaudhuri, "Exploiting statistics on query expressions for optimization," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '02. New York, NY, USA: ACM, 2002, pp. 263–274. [Online]. Available: http://doi.acm.org/10.1145/564691.564722

[151] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03. VLDB Endowment, 2003, pp. 81–92. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315460

[152] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," vol. 2006, 04 2006.

[153] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 133–142. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281210

[154] 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, Aug 2009, pp. 248–252.

[155] W. Ng and M. Dash, *Discovery of Frequent Patterns in Transactional Data Streams*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–30. [Online]. Available: https://doi.org/10.1007/978-3-642-16175-9_1

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

[157] C. Ruiz, E. M. Ruiz, and M. Spiliopoulou, "C-denstream: Using domain knowledge on a data stream," in *Discovery Science*, 2009.

[158] W.-H. ZHU, Y. Jian, and X. Yi-Huang, "Arbitrary shape cluster algorithm for clustering data stream," *Journal of Software*, vol. 17, 03 2006.

[159] H. Wang, Y. Yu, Q. Wang, and Y. Wan, "A density-based clustering structure mining algorithm for data streams," in *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, ser. BigMine '12. New York, NY, USA: ACM, 2012, pp. 69–76. [Online]. Available: http://doi.acm.org/10.1145/2351316.2351326

[160] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The clustree: indexing micro-clusters for anytime stream mining," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 249–272, Nov 2011. [Online]. Available: https://doi.org/10.1007/s10115-010-0342-8

[161] A. Amini, T. Y. Wah, M. R. Saybani, and S. R. A. S. Yazdi, "A study of density-grid based clustering algorithms on data streams," in *2011 Eighth International*

*Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, vol. 3, July 2011, pp. 1652–1656.

[162] A. Amini and T. Y. Wah, "Density micro-clustering algorithms on data streams : A review," 2011.

[163] A. Beirami, M. Sardari, and F. Fekri, "Results on the fundamental gain of memory-assisted universal source coding," in *2012 IEEE International Symposium on Information Theory Proceedings*. IEEE, 2012, pp. 1087–1091.

[164] Z. R. Hesabi, M. Sardari, A. Beirami, F. Fekri, M. Deriche, and A. Navarro, "A memory-assisted lossless compression algorithm for medical images," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2030–2034.

[165] Y. El-Sonbaty, M. Hamza, and G. Basily, "Compressing sets of similar medical images using multilevel centroid technique," in *In Proceedings of Digital Image Computing: Techniques and Applications*. Citeseer, 2003.

[166] S. Ait-Aoudia and A. Gabis, "A comparison of set redundancy compression techniques," *EURASIP Journal on Applied Signal Processing*, vol. 2006, pp. 216–216, 2006.

[167] H. MacMahon, K. Doi, S. Sanada, S. Montner, M. Giger, C. E. Metz, N. Nakamori, F. Yin, X. Xu, and H. Yonekawa, "Data compression: effect on diagnostic accuracy in digital chest radiography." *Radiology*, vol. 178, no. 1, pp. 175–179, 1991.

[168] N. Ekstrand, "Lossless compression of grayscale images via context tree weighting," 04 1996, pp. 132 – 139.

[169] R. Asraf, M. Akbar, and N. Jafri, "Statistical analysis of difference image for absolutely lossless compression of medical images," in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2006, pp. 4767–4770.

[170] S.-K. Kil, J.-S. Lee, D. Shen, J. Ryu, E. Lee, H. Min, and S. Hong, "Lossless medical image compression using redundancy analysis," 2006.

[171] S. Ghrare, M. M. Ali, K. Jumari, and M. Ismail, "An efficient low complexity lossless coding algorithm for medical images," *American Journal of Applied Sciences*, vol. 6, no. 8, p. 1502, 2009.

[172] S.-G. Miaou, F.-S. Ke, and S.-C. Chen, "A lossless compression method for medical image sequences using jpeg-ls and interframe coding," *IEEE transactions on information technology in biomedicine*, vol. 13, no. 5, pp. 818–821, 2009.

[173] K. Karadimitriou, "Set redundancy, the enhanced compression model, and methods for compressing sets of similar images," Ph.D. dissertation, Citeseer, 1996.

[174] I. Jolliffe, "Principal component analysis," *Springer Series in Statistics, Berlin: Springer, 1986*, 1986.

[175] S. Sra and I. S. Dhillon, "Generalized nonnegative matrix approximations with bregman divergences," in *Advances in neural information processing systems*, 2006, pp. 283–290.

[176] P. Rodrigues and J. Gama, "Distributed clustering of ubiquitous data streams," vol. 4, 01 2014.

[177] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[178] N. Ekstrand, "Lossless compression of grayscale images via context tree weighting," in *Data Compression Conference, 1996. DCC'96. Proceedings*. IEEE, 1996, pp. 132–139.

[179] [Online]. Available: http://db.jsrt.or.jp/eng.php

[180] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. New York, NY, USA: ACM, 2002, pp. 1–16. [Online]. Available: http://doi.acm.org.ezproxy.lib.rmit.edu.au/10.1145/543613.543615

[181] C. Aggarwal and P. Yu, "A survey of synopsis construction in data streams," in *Data Streams*, ser. Advances in Database Systems, C. Aggarwal, Ed. Springer US, 2007, vol. 31, pp. 169–207. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-47534-9_9

[182] A. Guerrieri and A. Montresor, "Ds-means: Distributed data stream clustering," in *Euro-Par 2012 Parallel Processing*, ser. Lecture Notes in Computer Science, C. Kaklamanis, T. Papatheodorou, and P. Spirakis, Eds. Springer Berlin Heidelberg, 2012, vol. 7484, pp. 260–271. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32820-6_27

[183] J. a. Gama, P. P. Rodrigues, and L. Lopes, "Clustering distributed sensor data streams using local processing and reduced communication," *Intell. Data Anal.*, vol. 15, no. 1, pp. 3–28, Jan. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1937721.1937723

[184] X. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Mining and Knowledge Discovery*, vol. 3, no. 3, pp. 263–290, 1999. [Online]. Available: http://dx.doi.org/10.1023/A%3A1009884809343

[185] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, no. 8, pp. 1313 – 1325, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016781919500017I

[186] M. Kornacker and D. Banks, "High-concurrency locking in r-trees," in *Proceedings of the 21th International Conference on Very Large Data Bases*, ser. VLDB '95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 134–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=645921.673305

[187] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '96. New York, NY, USA: ACM, 1996, pp. 103–114. [Online]. Available: http://doi.acm.org/10.1145/233269.233324

[188] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03. VLDB Endowment, 2003, pp. 81–92. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315460

[189] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA*, 2006, pp. 328–339. [Online]. Available: http://dx.doi.org/10.1137/1.9781611972764.29

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

[191] 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*. SIAM, 2006, pp. 328–339.

[192] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, and J. a. Gama, "Data stream clustering: A survey," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 13:1–13:31, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2522968.2522981

[193] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, no. 8, pp. 1313 – 1325, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016781919500017I

[194] X. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," *Data Min. Knowl. Discov.*, vol. 3, no. 3, pp. 263–290, Sep. 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1009884809343

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

[196] G. Cormode, S. Muthukrishnan, and W. Zhuang, "Conquering the divide: Continuous clustering of distributed data streams," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, April 2007, pp. 1036–1045.

[197] P. P. Rodrigues and J. Gama, "Distributed clustering of ubiquitous data streams," *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, vol. 4, no. 1, pp. 38–54, 2014. [Online]. Available: http://dx.doi.org/10.1002/widm.1109

[198] A. T. Vu, G. D. F. Morales, J. Gama, and A. Bifet, "Distributed adaptive model rules for mining big data streams," in *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, 2014, pp. 345–353. [Online]. Available: http://dx.doi.org/10.1109/BigData.2014.7004251

[199] M.-Y. Yeh, B.-R. Dai, and M.-S. Chen, "Clustering over multiple evolving streams by events and correlations," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 10, pp. 1349–1362, Oct 2007.

[200] B.-R. Dai, J.-W. Huang, M.-Y. Yeh, and M.-S. Chen, "Adaptive clustering for multiple evolving streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 18, no. 9, pp. 1166–1180, Sept 2006.

[201] G. Cormode, S. Muthukrishnan, and W. Zhuang, "Conquering the divide: Continuous clustering of distributed data streams," in *2007 IEEE 23rd International Conference on Data Engineering*, April 2007, pp. 1036–1045.

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

[203] E. Januzaj, H.-P. Kriegel, and M. Pfeifle, "Towards effective and efficient distributed clustering," in *In Workshop on Clustering Large Data Sets (ICDM*, 2003, pp. 49–58.

[204] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[205] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson, "Distributed clustering using collective principal component analysis," *Knowledge and Information Systems*, vol. 3, p. 2001, 1999.

[206] M. Klusch, S. Lodi, and G. Moro, "Distributed clustering based on sampling local density estimates," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, ser. IJCAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 485–490. [Online]. Available: http://dl.acm.org/citation.cfm?id=1630659.1630731

[207] Q. Zhang, J. Liu, and W. Wang, "Approximate clustering on distributed data streams," in *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 2008, pp. 1131–1139.

[208] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: An efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '96. New York, NY, USA: ACM, 1996, pp. 103–114. [Online]. Available: http://doi.acm.org/10.1145/233269.233324

[209] B. Cun and C. Roucairol, "Concurrent data structures for tree search algorithms," in *Parallel Algorithms for Irregular Problems: State of the Art*, A. Ferreira and J. Rolim, Eds. Springer US, 1995, pp. 135–155. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-6130-6_7

[210] K. Yi and Q. Zhang, "Multidimensional online tracking," *ACM Transactions on Algorithms (TALG)*, vol. 8, no. 2, p. 12, 2012.

[211] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *J. Mach. Learn. Res.*, vol. 11, pp. 1601–1604, Aug. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1756006.1859903

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

[213] Z. R. Hesabi, T. Sellis, and K. Liao, "Distclustree: A framework for distributed stream clustering," in *Databases Theory and Applications - 29th Australasian Database Conference, ADC 2018, Gold Coast, QLD, Australia, May 24-27*, 2018, pp. 288–299.

[214] V. Vasaitis, A. Nanopoulos, and P. Bozanis, "Merging r-trees," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. IEEE, 2004, pp. 141–150.

[215] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[216] N. Thaper, S. Guha, P. Indyk, and N. Koudas, "Dynamic multidimensional histograms," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. ACM, 2002, pp. 428–439.

[217] D. Reynolds, "Gaussian mixture models," *Encyclopedia of biometrics*, pp. 827–832, 2015.

[218] K. Kashino, T. Kurozumi, and H. Murase, "A quick search method for audio and video signals based on histogram pruning," *IEEE Transactions on Multimedia*, vol. 5, no. 3, pp. 348–357, 2003.

[219] Y. Wang and F. Makedon, "R-histogram: quantitative representation of spatial relations for similarity-based image retrieval," in *Proceedings of the eleventh ACM international conference on Multimedia*. ACM, 2003, pp. 323–326.

[220] Y. E. Ioannidis and V. Poosala, "Histogram-based approximation of set-valued query-answers," in *VLDB*, vol. 99, 1999, pp. 174–185.

[221] V. Poosala, V. Ganti, and Y. E. Ioannidis, "Approximate query answering using histograms," *IEEE Data Eng. Bull.*, vol. 22, no. 4, pp. 5–14, 1999.

[222] Y. E. Ioannidis and V. Poosala, "Histogram-based approximation of set-valued query-answers," Jan. 14 2003, uS Patent 6,507,840.

[223] S. Guha and N. Koudas, "Approximating a data stream for querying and estimation: Algorithms and performance evaluation," in *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 2002, pp. 567–576.

[224] K. Yi and Q. Zhang, "Optimal tracking of distributed heavy hitters and quantiles," *Algorithmica*, vol. 65, no. 1, pp. 206–223, 2013.

[225] H. Xie, E. Tanin, and L. Kulik, "Distributed histograms for processing aggregate data from moving objects," in *2007 International Conference on Mobile Data Management*. IEEE, 2007, pp. 152–157.

[226] F. Furfaro, G. M. Mazzeo, D. Saccà, and C. Sirangelo, "Hierarchical binary histograms for summarizing multi-dimensional data," in *Proceedings of the 2005 ACM symposium on Applied computing*. ACM, 2005, pp. 598–603.

[227] D. Achakeev and B. Seeger, "A class of r-tree histograms for spatial databases," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '12. New York, NY, USA: ACM, 2012, pp. 450–453. [Online]. Available: http://doi.acm.org/10.1145/2424321.2424387

[228] D. Donjerkovic, Y. Ioannidis, and R. Ramakrishnan, "Dynamic histograms: Capturing evolving data sets," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 1999.

[229] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, "Mergeable summaries," *ACM Transactions on Database Systems (TODS)*, vol. 38, no. 4, p. 26, 2013.

[230] G. Cormode, S. Muthukrishnan, and K. Yi, "Algorithms for distributed functional monitoring," in *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2008, pp. 1076–1085.

[231] B. Babcock and C. Olston, "Distributed top-k monitoring," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 28–39.

[232] G. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi, "Holistic aggregates in a networked world: Distributed tracking of approximate quantiles," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 25–36.

[233] Z. Huang, K. Yi, and Q. Zhang, "Randomized algorithms for tracking distributed count, frequencies, and ranks," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. ACM, 2012, pp. 295–306.

[234] M. Tang, F. Li, and Y. Tao, "Distributed online tracking," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 2047–2061.

[235] G. Cormode and M. Garofalakis, "Sketching streams through the net: Distributed approximate query tracking," in *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005, pp. 13–24.

[236] O. Papapetrou, M. Garofalakis, and A. Deligiannakis, "Sketch-based querying of distributed sliding-window data streams," *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 992–1003, 2012.

[237] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston, "Finding (recently) frequent items in distributed data streams," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. IEEE, 2005, pp. 767–778.

[238] G. Cormode and K. Yi, "Tracking distributed aggregates over time-based sliding windows," in *International Conference on Scientific and Statistical Database Management*. Springer, 2012, pp. 416–430.

[239] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 187–198.

[240] [Online]. Available: https://www.kaggle.com/camnugent/california-housing-prices

[241] https://www.kaggle.com/piyushgoyal443/red-wine-dataset#wineQualityReds.csv.

[242] [Online]. Available: https://keras.io

[243] C. C. Aggarwal, "Text summarization," in *Machine Learning for Text*. Springer, 2018, pp. 361–380.

[244] S. Elfayoumy and J. Thoppil, "Improving unstructured text summarization using an ensemble approach," *GSTF Journal on Computing (JoC)*, vol. 4, no. 1, 2018.

[245] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou, "Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes," in *Proceedings of the 22Nd International Conference on Data Engineering*, ser. ICDE '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 34–. [Online]. Available: https://doi.org/10.1109/ICDE.2006.169

[246] [Online]. Available: https://github.com/hindupuravinash/the-gan-zoo