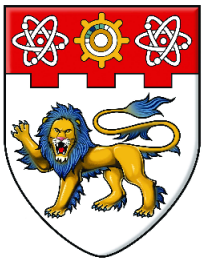


NANYANG TECHNOLOGICAL UNIVERSITY

# Advanced Classification for Streaming Time series and Data Streams

A Thesis Submitted to the School of Computer Engineering
of the Nanyang Technological University

by

**NGUYEN HAI LONG**

In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

2013

*"Gratitude is the fairest blossom which springs from the soul."*

Henry Ward Beecher, Writer

# Acknowledgments

I would like to express my gratitude to many people and organizations for their support and contributions:

**EADS Innovation Works South Asia** and **EDB** (Economic Development Board of Singapore) for my research scholarship.

**Associate Professor Ng Wee Keong**, my supervisor, for his invaluable guidance, encouragement, and sharing of his knowledge. Prof. Ng helped me a lot initially when I first came to Singapore; he taught me how to do research, think critically, and present my work professionally.

**Dr. David Woon Yew-Kwong**, my co-supervisor, for his constant support and practical advice. I wish to express my deep appreciation to David, who make I feel very lucky to work with.

**Frederic Viniacourt and other EADS colleagues** for the sharing of their industrial experience and warm friendship during the time I spent at EADS.

**Wan Li, Luan M. Nguyen, Duc H. Tran, Rajesh Sharma, Tam T. Nguyen, and Tran H. Ha** for sharing their research experience.

**Chee Keong Lai, Chiew Song Chua, and Loo Kian Hock** for their technical support at the Centre for Advanced Information Systems (CAIS).

Last but not least, I must really thank **my family and my wife** for their unconditional love and unwavering support especially during the trying moments of my exciting research journey.

*"By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest."*

CONFUCIUS (551-479 B.C.) Philosopher

# Abstract

Nowadays, overwhelming volumes of sequential data are very common in scientific and business applications, such as biomedicine, stock markets, retail industry, and communication networks. Time series and data streams are the two most popular types of sequential data. The main difference between them is that time series is on a *single variable* domain, while data streams are generally on a *multivariate* domain. However, they do share some unique characteristics: possibly infinite volume, time-ordered and dynamically changing. In this dissertation, we propose classification algorithms for time series and data streams that satisfy strict constraints, such as *bounded memory, single pass, real-time response* and *concept-drift detection*. Here, a concept drift refers to the situation where the data's underlying distribution changes over time.

For massive time series datasets, classification algorithms that are based on *motifs* (frequent subsequences) are preferable since it not only has low complexity but can also achieve high accuracy. However, state-of-the-art algorithms can only find motifs with a predefined length, which greatly affects their performance and practicality. To overcome this challenge, we introduce the notion of a *closed* motif; a motif is *closed* if there is no motif with a longer length having the same number of occurrences. We also propose a novel closed-motif-based classifier, which is lightweight, effective and efficient for time series classification.

Furthermore, we continue to examine a more challenging problem of classifying data streams in a multivariate domain. Here, we are confronted with a *feature drift* problem,

where the importance/relevance of a set of features will change over time. We propose a general framework to integrate feature selection and heterogeneous ensemble learning, which is able to adapt to different types of concept drifts and works well with various kinds of datasets. The ensemble consists of well-chosen online classifiers and is equipped with an optimal weighting method. It updates online classifier members for gradual drifts, and replace outdated members by new ones for feature drifts.

Additionally, we extend our algorithms in a practical environment, where labeled data is very scarce and there is a need for the concurrent mining of data streams in order to make full use of the single-pass data. Conventional stream mining algorithms only focus on stand-alone mining tasks. Therefore, we propose an incremental algorithm that performs clustering and classification concurrently, which not only maximize throughput, but also achieve better mining results. Moreover, enhanced with a novel active learning technique, our algorithm only requires a small number of queries to work well with very sparsely labeled data streams.

Finally, as the volume of sequential data grows steadily, a single computer with limited computing power may soon be insufficient for the mining processes. Cloud computing, a cutting-edge technology that provides elastic computing on demand, will certainly facilitate large sequential data mining. Therefore, we plan to adapt and migrate our algorithms to a cloud computing platform in the future.

# Contents

# List of Figures

# List of Tables

*"Research is creating new knowledge."*

NEIL ARMSTRONG (1930-2012) Astronaut

*"To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science."*

ALBERT EINSTEIN (1879-1955) Physicist

# 1 Introduction

## 1.1 Background

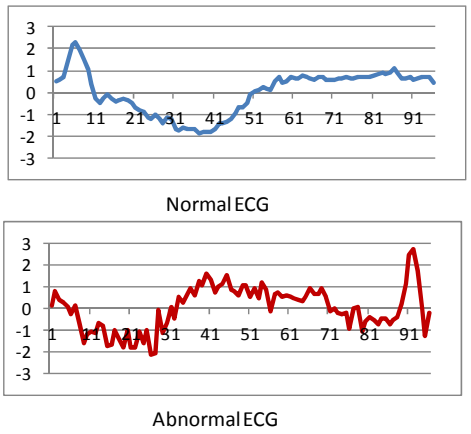
Data mining, which is *'a process of intelligent computation to extract hidden knowledge from large datasets and to present them in compact, understandable models and patterns'*, has attracted much attention in academia and industry [5]. Traditional data mining mostly focused on mining resident and static data repositories that usually fit in main memory and can be read repeatedly. However, with technological developments, many organizations have generated large amounts of data at higher speeds than ever, giving rise to the emergence of massive sequential data. In general, a sequence is an ordered list of events, each of which can be a symbolic value, a numerical value, a vector of real values or a complex data type. In this thesis, we consider the two most popular types of sequential data: *"streaming time series"* and *"data streams"*.

A streaming time series is a string of massive chronological observations, which can be found in scientific and business applications, such as biomedicine, sensor networks and stock markets. For example, electrocardiograms (ECGs) that record heartbeats are

widely used for initial screening and diagnosis. An ECG is usually recorded at 360Hz or above, and a patient generates up to 100MB of ECG time series per day. A large hospital with thousands of patients can easily generate terabytes of data per day. Given a patient's ECG, how does one classify whether he has a heart disease?

Data streams are often generated by real-time surveillance industry, online transactions in retail industry, communication networks, and scientific experiments. For example, on a daily basis, Amazon records millions of click-streams that capture users' activities on its website; Google handles more than 100 million searches; NASA satellites generate around 1.5 TB images; large banks have millions of credit cards and ATM operations; and popular social networks, such as, Facebook and Twitter, generate tremendous amounts of online multimedia and textual data. There are several research problems, such as classifying malicious online transactions, distinguishing different types of users and categorizing discussion topics in social networks.

The main difference between time series classification and data stream classification is that time series classification is on a *single variable* domain, while data stream classification is on a *multivariate* domain. Time series classification looks at each time series (variable) as one unit and each time series with all its data objects is used for classification. Time series classification preserves the integrity of a time series by considering each streaming time series as a whole. On the other hand, data stream classification considers each example with many variables as one unit, it preserves the concurrence of all variables. For example, Figure 1.1(a) illustrates a time series of two ECG cycles: a normal ECG at the top and an abnormal ECG at the bottom. Time series classification looks at a whole ECG series as an input, therefore, preserves the integrity of a time series. Figure 1.1(b) shows a log record of Web click-streams, including a visitor ID, IP address, time-stamp, duration, byte transfer, page URL, etc. These variables are considered as one unit in Web click-stream classification. However, time series data and data streams




| Variables | Information |
|---|---|
| Visitor ID | 41521390 |
| IP Address | 2.111.94.18 |
| Timestamp | 06-12-11 0:55 |
| Access request | GET |
| Status code | 200 |
| Duration (second) | 5.02 |
| Bytes transfer | 1540096 |
| Page URL | http://www.amazon.com/ |
| Referral Page | http://sce.ntu.edu.sg/Pages/Home.aspx |
| User Agent | Safari; iPad; en-us |

(b)

Figure 1.1: Examples of time series data and data streams.

both share some unique characteristics: *possibly infinite volume*, *time-ordered* and *dynamically changing*. Due to these characteristics, the field of mining these complex data remains a challenging problem. To be able to work with these data, algorithms have to satisfy some strict constraints, such as *bounded memory, single pass, real-time response* and *concept-drift detection*. In other words, since the volume of data is huge and comes at a high speed; algorithms must work with limited memory resource, process a data item at most once, produce results in a real-time manner and detect the evolution of the underlying data.

Much work have been proposed in this research field: time series classification [2,6–10], data stream classification [11–23], time series clustering [24–26], data stream clustering [27–39], frequent patterns in time series [3, 40–46] and data stream [47–55]. In this thesis, we consider the *classification* problem for streaming time series and data-streams.

## 1.2 Motivations

Classification or supervised learning is probably the most well-studied problem in data mining. It is widely used for a vast diversity of applications such as document classification, computer vision, natural language processing, biological classification, pattern

recognition, and security informatics. In general, classification is a two-step process. In the learning step, a model is built to generalize a predetermined training dataset with its provided class labels. Once trained, the model is then used to predict the class label of new data in the testing step. There are many classification techniques in the literature, such as decision tree [56–58], Bayesian classification [59,60], rule-based classification [61,62], neural networks [63], support vector machines [64], k-nearest-neighbor [65] and ensemble classifiers [66]. These algorithms are extended to work for time series and data streams. Although there are many classification algorithms for time series [2, 6–10] and data streams [11–23], they still have limitations. We list some of the most important ones as follows:

- For time series classification:
  - Most of the time series classifiers do not work in a streaming manner. They cannot update their models with continuously arriving data.
  - Many time series classifiers are based on finding frequent subsequences, which are representatives of each class. However, the length of the subsequences must be predefined.
- For data stream classification:
  - Many data stream classifiers do not work well with high dimensional data due to the-curse-of-dimensionality problem, where all data objects appear to be sparse and dissimilar.
  - Most data stream classifiers have low accuracy and do not adapt well to different types of concept drifts.
  - Most data stream classifiers do not work well with very sparsely labeled datasets. They usually assume that data is fully labeled; therefore, their accuracy drops quickly when working with sparsely labeled datasets.

– Most data stream algorithms focus on stand-alone mining tasks. Given the single-pass nature of data streams, it makes sense to maximize throughput by performing multiple complementary mining tasks concurrently.

## 1.3 Objectives

To address the above challenges, this thesis first investigates the state-of-the-art classification algorithms for time series and data streams under strict constraints. The ultimate goal of our research is to develop classification frameworks for streaming time series and data streams, which are *fast*, *accurate* and *scalable* with respect to the size and dimensionality of datasets. Here, we focus on the following research issues:

### 1.3.1 Closed Motifs for Streaming Time Series Classification

Motif discovery, which aims to find frequent patterns or subsequences (called *motifs*), is one of the most important time series mining tasks. It is an essential preprocessing procedure that is widely used for time series classification and other mining tasks. Although several algorithms have been proposed to solve this problem, there are many limitations. To reduce computational requirements, most existing work limits the search space using the assumption that the length of motifs $w$ must be predefined [40–44]. However, these algorithms are very sensitive to the choice of $w$. If the value of $w$ is too small, only subsequences of the hidden patterns can be found and a large number of redundant motifs will be produced; if the value of $w$ is too large, patterns may not be found because patterns followed by different adjacency subsequences (tails) are considered unmatched [10]. Therefore, it is difficult to determine the optimal length of interesting motifs; a suboptimal choice results in missing the key motifs or having too many redundant motifs. Even if the value of $w$ is properly configured, only motifs with length $w$ can be discovered.

Consequently, the problem of finding different length motifs for streaming time series is a critical issue. Most time series algorithms in the literature have not properly addressed it yet. To address this issue, we introduce the definition of *closed motifs* and propose a novel algorithm to classify streaming time series.

### 1.3.2 Heterogeneous Ensemble for Feature Drifts in Data Streams

In high-dimensional data stream, not all data features (attributes) are important to the learning process. The critical task of dimension reduction techniques is to extract the set of relevant and non-redundant features so that the learning process is more meaningful and faster. Moreover, in data streams, the importance of a feature evolves over time and is restricted to a certain time period. Features that are previously considered as informative may become irrelevant; and vice-versa, those rejected features may become important features in the future. Data streams algorithms are required to monitor the evolution of features, which can be considered as a sophisticated case of concept drifts called *feature drifts*.

Generally, there are two common approaches to handle to concept drifts: online incremental learning and ensemble learning. Incremental learning trains a single classifier and updates it with newly arriving data. Ensemble learning manages concept drifts using any of the following adaptive approaches: (1) majority vote or weighting combination [67], (2) real-time updates the individual classifiers [18, 68], and (3) changing the ensemble structure by replacing outdated classifiers [19, 22]. However, none of them adapts well to feature drifts. Hence, we propose a framework to incorporate feature selection into a heterogeneous ensemble. This approach can adapt well to different types of concept drifts in high dimensional data streams.

### 1.3.3 Concurrent Semi-Supervised Learning for Data Streams

Although many algorithms have been proposed to work with data streams, they only focus on stand-alone mining tasks. In practical applications, it is desirable to concurrently perform multiple types of mining in order to better exploit data streams. For example, website administrators want to use clickstream data to classify users into specific types and cluster Web pages of similar topics at the same time in order to enhance the user's experience. In addition, concurrent mining offers potential synergy: *"the knowledge gained by one mining task may also be useful to other mining tasks"*. Unfortunately, there is currently little research on concurrent stream mining.

Moreover, data streams are huge so that we are only able to label a small portion of them. It is necessary to support experts to decide on "*which samples should be labeled?*" Active learning attempts to address this problem by finding important samples from newly arrived instances for labeling. It aims to achieve an accurate predictive model using as few labeled instances as possible; thereby, minimizing the labeling cost. In summary, there is a need to provide a concurrent mining framework that is able to concurrently perform many mining tasks for sparsely labeled data streams and is able to help experts on labeling important instances.

## 1.4 Scope and Limitations

This dissertation mainly covers classification problem of massive sequential data. It also includes feature selection, clustering, and active learning in order to improve the learning process. The study will not cover other mining tasks, such as frequent pattern mining, outlier detection, graph mining and social network analysis. Furthermore, this study looked into two popular types of sequential data: time series data and data streams. We currently do not include on other types of sequential data; for example, complex symbolic sequences (such as a sequence of items bought by a customer over a period of

time $< (milk, bread); (egg, potato); (coke, cheese, apple) >$) and complex event sequences (such as a patient record with multiple numerical measurement, category fields and text descriptions). Finally, we implicitly perform basic preprocessing techniques on the input dataset in our experiments. We remove data objects with missing values and then perform z-normalization afterwards.

## 1.5 Approach and Methodology

We adopt an online–offline approach, which will be discussed in Chapter 2. We continuously update the synopsis of upcoming data in a real-time manner and perform classification upon user requests. In Chapter 3, we build a synopsis of streaming time series data in the form of a suffix tree. We update the tree according to a probabilistic model, and a new node is only created whenever there exist a motif belonging to it. Only closed tree nodes are used as inputs to the nearest-neighbor classifier. Similarly, we develop a hierarchical tree structure to summarize statistical properties of data streams in Chapter 5. The tree structure is dynamic; it has the ability to extend to maximum height and merge child nodes that are all dense or sparse. We further deploy a time fading model on this tree structure to diminish the effect of old data. For outdated data, its weight on the tree structure is gradually decreased and it will eventually be deleted. As a result, we have a multi-resolution summary of data streams while conserving memory space. Then, classification and clustering are concurrently performed on this tree synopsis. Clustering utilizes the presence of labeled data to produce more precise clusters; conversely, the classification exploits the clustering results to make its prediction more accurate.

In Chapter 4, we integrate feature selection and ensemble learning for high-dimensional data streams. Typically, there are three feature selection models: the filter model, wrapper model and hyper model. We chose the filter model as it is fast and suitable for data

streams. We detect feature drifts by modifying the FCBF feature selection algorithm in a sliding window manner [69]. A heterogeneous ensemble is then constructed based on the output of the important feature set. The ensemble is updated by replacing outdated classifier members, and then applying an optimal weighting method to achieve high accuracy. The ensemble is able to adapt well to different types of concept drifts.

## 1.6 Contributions

We have developed various advanced classification algorithms for streaming time series and data streams. The contributions of this dissertation are summarized as follows:

- In order to discover motifs with appropriate lengths, we have introduced the definition of *closed motifs* and proposed a novel time series classifier that is based on closed motifs. We first build a synopsis of streaming time series data in the form of a suffix tree. The synopsis is constructed and updated in real-time using only a single pass, so that our algorithm is able to process massive time series at high data rates. Then, we traverse the tree to discover closed motifs by applying the downward closure property. We further implement our algorithm together with a user-friendly toolkit *ARC-VIEW*, which is capable of visualizing high-resolution motifs in streaming time series. Moreover, we demonstrate the usefulness of closed motifs in time series classification. The nearest neighbor classifier we use is based on the most distinctive closed motifs; it outperforms prominent classifiers in terms of accuracy and speed for many datasets in various domains.

- We propose an algorithm called HEFT-Stream (Heterogeneous Ensemble with Feature drifT for Data Streams) that incorporates feature selection into a heterogeneous ensemble for adapting to different types of concept drifts. Feature selection helps

to extract the most informative feature subset that accelerates the learning process and increases accuracy. We first apply feature selection techniques on the data streams in a sliding window manner and monitor the feature subset to detect feature drifts representing sudden concept drifts. The heterogeneous ensemble is constructed from well-chosen online classifiers. The ratios of classifier types are dynamically adjusted to increase the ensemble's diversity, and allows the ensemble to work well with many kinds of datasets. Moreover, the ensemble adapts to the severity of concept drifts; we update the online classifier members for gradual drifts, and replace outdated members by new ones for sudden drifts. We have conducted extensive experiments to show that our ensemble outperforms state-of-the-art ensemble learning algorithms for data streams.

- We investigate the potential of concurrent semi-supervised learning on data streams and propose an incremental algorithm called CSL-Stream (Concurrent Semi–supervised Learning of Data Streams) that performs clustering and classification at the same time. We have conducted extensive experiments to show that CSL-Stream outperforms state-of-the-art data stream algorithms in terms of speed, accuracy and scalability. Experimental results also showed that it can produce mining results in a real-time manner, making it suitable for online applications. Moreover, enhanced with a novel active learning technique, CSL-Stream only requires a small number of queries to work well with very sparsely labeled datasets. The success of CSL-Stream paves the way for new research direction in understanding latent commonalities among various data mining tasks in order to exploit the power of concurrent stream mining.

## 1.7 Thesis Structure

This first chapter introduces the background, motivations, research objectives and briefly described contributions of our study in the area of classification for streaming time series and data streams. The remainder of the thesis is organized as follows:

- Chapter 2 presents common background knowledge of mining streaming time series and data streams. We discuss characteristics of these complex data and present a general mining model for these data. We also survey various traditional classifiers and state-of-the-art classifiers for streaming time series and data streams. We further discuss the relationships among them, summarize their advantages and limitations, and evaluate their capabilities in terms of satisfying the above mining constraints.

- With this background, we begin Chapter 3 with our first contribution in developing a novel classification algorithm for streaming time series, which is based on closed motif discovery. In this chapter and subsequent two chapters, both theoretical proofs and empirical studies are elaborated.

- Chapter 4 introduces a general framework to integrate feature selection and heterogeneous ensemble learning for data stream classification.

- In Chapter 5, we present our proposed framework on concurrent semi-supervised learning.

- Finally, we conclude the thesis in Chapter 6 and more importantly, discuss interesting future work to continue to push boundaries.

*"Look deep into nature, and then you will understand everything better."*

ALBERT EINSTEIN (1879-1955) Physicist

*"Research is to see what everybody else has seen, and to think what nobody else has thought."*

ALBERT SZENT-GYRGYI (1893-1986) Physicist

# 2 Literature Review

## 2.1 Introduction

Nowadays, with the advance of technology, many applications generate huge amounts of sequential data at very high speed. Examples include stock market, biomedicine, sensor networks, network traffic, web click streams, and video surveillance. There are two popular types of massive sequential data: *time series* and *data streams*. Extracting hidden knowledge/patterns from these massive sequential data confronts several common challenges and has become a hot research topic. Unlike traditional data mining where the dataset is static and can be repeatedly read, algorithms for these complex data have to satisfy constraints, such as bounded memory, single-pass, real-time response, and concept-drift detection.

There is much published work on this research field, including surveys that give overviews of various approaches. Gaber *et al.* introduced an illustrative survey with theoretical foundations and basic algorithms on data stream mining [70]. However, it may be hard to understand the survey as the authors did not give any clear distinction among

these algorithms. From a statistical point of view, Gama and Rodrigues [71] focused on illustrating examples of data stream mining with specific algorithms and applications. However, the presented algorithms are quite outdated. Aggarwal [72] attempted to give a broader overview of data stream mining, where more mining tasks and real applications were discussed. Nevertheless, it does not give readers a comprehensive understanding of current data stream mining methods.

To address the above limitations, we introduce a comprehensive survey of classification algorithms for time series and data streams. We present preliminaries and overview of streaming mining in Section 2.2. Then, we discuss in details state-of-the-art algorithms together with their merits and limitations. We sort them into different categories based on their approaches and derive the relationships between traditional classification, time series classification and data stream classification in Sections 2.3, 2.4, 2.5. We also analyze capabilities of each algorithm in terms of addressing the above constraints.

**Sample Applications:**

- *Sensor networks*: A sensor network consists of spatially distributed autonomous sensors that cooperatively monitor an environment. These sensors can sense physical values about the environment, such as temperature, sound, vibration, pressure, humidity and light; they can cooperatively send their data through the network to a center. Sensor networks are involved in many real-life applications, such as traffic monitoring, smart homes, habitat monitoring, and healthcare [73]. For example, modern hospitals are equipped with patient monitoring system to improve health care quality and staff productivity. Many body sensors, which are connected to critically-ill patients, can produce massive physiological data, such as temperature, electrocardiogram, pulse oximetry, and blood pressure. Since sensor devices only store updated data and human eyes cannot detect these signal subtleties, the system must analyze healthcare streaming data in real time and extract meaningful

information for medical professionals, such as clinical rules to identify significant events [74].

- *Mining query streams*: Searching the Web to retrieve information has become an essential activity of our everyday life. Existing search engines such as Google, Bing, and Yahoo handle millions of queries on a daily basis. Mining query streams to provide users better searching results has attracted much research work. For example, Zeng *et al.* clustered Web search results to facilitate users' quick browsing through search results [75]. Users generally enter very short text queries, which may produce imprecise searching results due to the ambiguity of the search terms. Chien and Immorlica studied temporal correlation among text queries over a period of time and refine users' search by suggesting alternative queries [76].

- *Network monitoring*: The Internet has many connected routers, which communicate with each other by sending IP packets. To manage such networks, we need to analyse traffic data to discover usage patterns and unusual activities in real time. Traffic data are recorded in the form of log files at many levels, such as packet logs containing source and destination IP addresses; flow logs storing the number of packets sent, start time, end time, and protocol; and SNMP logs aggregate the number of bytes sent over each link every few minutes [77]. An example of network management is to detect and prevent malicious attacks in a large Internet service provider networks. A data stream classifier is required to classify in real time different kinds of attacks, such as denial-of-service (DOS), unauthorized access from a remote machine (R2L), unauthorized access to local super-user privileges (U2R), surveillance and other probing attacks.

- *Social network streams*: Online social networks (OSNs) have become increasingly popular; for example, Facebook, Twitter, and LinkedIn have millions of active

users. Such networks generate tremendous amounts of online data streams in the form of text, multimedia data, linkages and interactions. There is much research on social network mining. For example, stream clustering methods are used to detect communities, and monitor their evolution in social networks. They can explain how communities emerge, expand, shrink, and evolve in a social network. Moreover, stream classification algorithm helps to classify different types of users, or categorize discussion topics in social networks.

**Software:** There is some useful, open-source software for data stream mining research.

- WEKA[1]: WEKA is a well-known data mining software within the academic environment. WEKA includes a collection of learning algorithms such as data preprocessing, classification, regression, clustering, association rules, and visualization.

- Massive Online Analysis (MOA)[2]: MOA is based on the WEKA framework that is designed for data stream mining. It includes many online learning algorithms for evolving data streams; for example, very fast decision tree [11], and ensemble learning [18]. Moreover, MOA provides data stream generators, such as SEA concepts, STAGGER, and rotating hyper-plane.

- RapidMiner[3]: RapidMiner is another open-source software for data mining. RapidMiner is more comprehensive than WEKA as it includes all algorithms in WEKA and other advanced algorithms. Moreover, it is more intuitive as it is able define a mining process as a series of operators and it provides more visualization tools.

[1]http://www.cs.waikato.ac.nz/ml/weka
[2]moa.cs.waikato.ac.nz
[3]http://rapid-i.com

## 2.2 Overview

We define a data stream $\mathbf{X}$ as an infinite series of data objects or samples: $\mathbf{X} = (x_1, x_2, ..., x_t, ...)$. Each data object $x_i$ has a label $y_i \in \mathbf{Y} = \{y_1, y_2, \ldots, y_c\}$. Classification is the process of finding a general model from past data to apply to new data. Classification is performed in two steps: Learning step and testing step. In the learning step, the system tries to learn a model from a collection of data objects, called the training set. In the testing step, the model is used to assign a class label for unlabeled data objects in the testing set. There are many classification techniques in the literature, such as the decision tree [56–58], Bayesian classification [59, 60], rule-based classification [61, 62], neural networks [63], support vector machines [64], $k$-nearest-neighbor [65] and ensemble classifiers [66].

### 2.2.1 Constraints & a general model

Streaming time series and data streams share intrinsic characteristics, such as possibly infinite volume, chronological order and dynamical changes. For example, the amount of heartbeat data of a patient over a long period of time can be huge. Its underlying distribution can be changed over time, since the health condition of the patient may become better or worse. Google processes more than 100 million searches daily, each of which is attached to a time stamp; and these searches are changed according to different hot topics at different times. Since streaming time series and data streams have several common characteristics, we henceforth define *"streaming data"* as a general case of both streaming time series and data streams.

Table 2.2.1 shows comparisons between traditional data mining and streaming data mining. In traditional data mining, it is possible to scan a dataset many times, and execute without limited time and memory constraints. On the other hand, streaming

Table 2.1: Comparisons between traditional data mining and streaming data mining.

| | Traditional Data Mining | Streaming Data Mining |
|---|---|---|
| Number of passes | multiple | single |
| Time | unlimited | real-time |
| Memory | unlimited | bounded |
| Number of concepts | one | multiple |
| Result | accurate | approximate |

data mining may produce *approximate* results but has to satisfy constraints, such as *single-pass, real-time response, bounded memory,* and *concept-drift detection*:

- *Single-pass:* Unlike traditional data mining that may read static datasets repetitively, each sample in streaming data is examined at most once and cannot be backtracked.
- *Real-time response:* Many streaming data applications such as stock market prediction require real-time response. The amount of time for processing the data must be small to derive conclusions.
- *Bounded memory:* The amount of arriving data is extremely large or potentially infinite. As we may only compute and store a small summary of the streaming data and possibly throw away the rest of the data, approximate results are acceptable.
- *Concept-drift detection:* Concept drifts refer to the situation when the discovered patterns (or the underlying data distribution) change over time.

**Definition 1** *A **concept** or a data source is defined as a set of the prior probabilities for the classes $P(y_i)$ and the class-conditional density functions (pdf) $P(X|y_i), i = 1, \ldots, c$ (definition from [78]):*

$$S = \{(P(y_1), P(X|y_1)); \ldots; (P(y_c), P(X|y_c))\}.$$

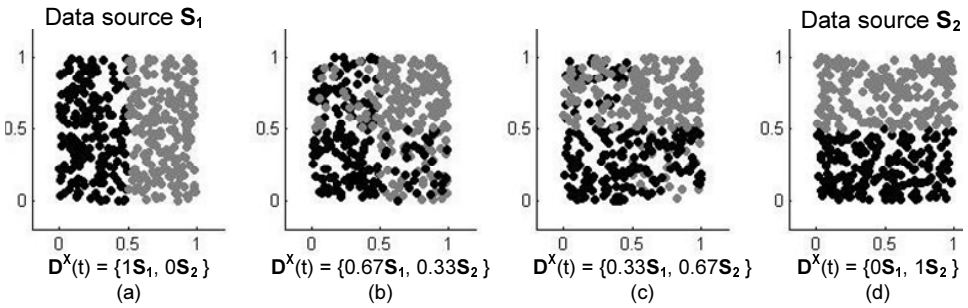



Figure 2.1: Example of a gradual change from a data source $S_1$ to a data source $S_2$ [1]. Class $y_1$ is depicted with grey dots, and class $y_2$ with black dots.

Traditional data mining only works with a single concept, which means the distributions of training dataset and testing dataset are the same. Meanwhile, data streams are dynamic and have many concepts. An example is the problem of preventing malicious attacks in network monitoring. The descriptions of the two class "normal" and "attack" evolve over time. Moreover, attackers are active and always try to devise new attacking methods thereby changing the definition of class "attack".

Suppose data stream $\mathbf{X}$ consists of a set of $k$ data sources $S_i$ with known distributions, $i = 1, \ldots, k$. At a time stamp $t$, we have one or more "active" data sources. Let $w_i(t) \in [0, 1]$ are the influence of data source $S_i$ at a time stamp $t$, where $\sum_{i=1}^{k} w_i(t) = 1$. The data distribution of the data stream $X$ at a time stamp $t$ is characterized as follow:

$$D^X(t) = \{w_1(t)S_1, \ldots, w_k(t)S_k\}$$

**Definition 2** *Suppose a data stream* $\mathbf{X}$ *consists of a set of* $k$ *data sources* $S_i$ *with influence* $w_i$*. The underlying data distribution of data stream* $\mathbf{X}$ *is* $D^X(t) = \{w_1(t)S_1, \ldots, w_k(t)S_k\}$*, where* $w_i(t) \in [0, 1]$ *and* $\sum_{i=1}^{k} w_i(t) = 1$*. If for any two time stamps* $t_1$ *and* $t_2$ *that* $D^X(t_1) \neq D^X(t_2)$*, we say there is a* ***concept drift****.*

The term *concept drift* indicates the change of the influence of data sources over time. Figure 2.1 illustrates an example of a concept drift, where the data distribution of a data stream $\mathbf{X}$ gradually changes from a data source $S_1$ to a data source $S_2$.

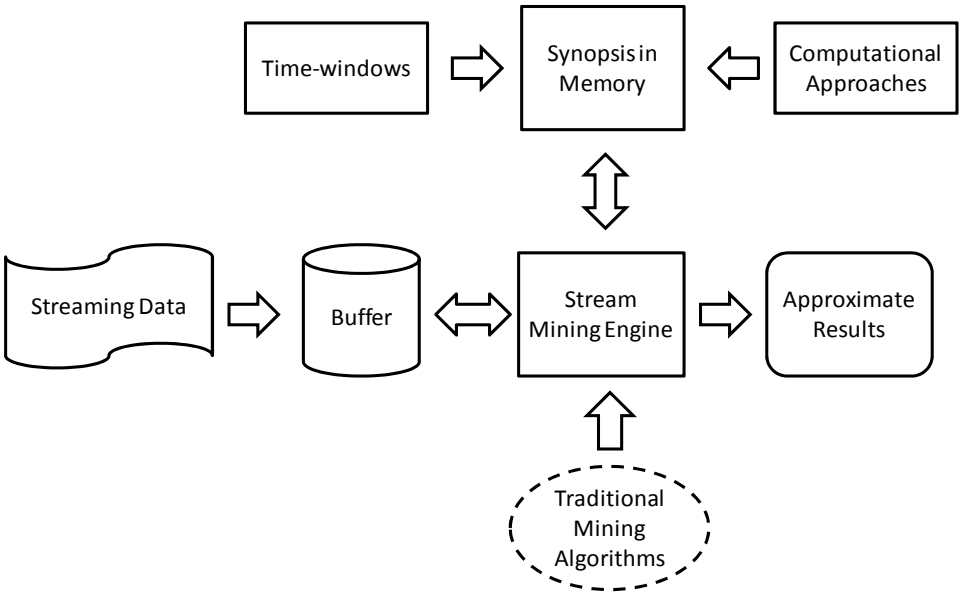



Figure 2.2: A general model for streaming data mining.

To help readers understand an overview of streaming data mining, we propose a general model of streaming data algorithms in Figure 2.2. When streaming data comes, a buffer is used to store the most recent data. The stream mining engine reads the buffer to create a synopsis of the data in memory. In order to maintain the synopsis, the system may apply different time-window and computational approaches. When certain criteria are triggered, for example, a user's request after a certain time lapse, the stream mining engine will process the synopsis and output approximate results. In general, most streaming data algorithms are derived and adapted from traditional mining algorithms.

### 2.2.2 Time Window

As streaming data is potentially infinite, it is possible to only able to process a portion of the entire data. This interesting portion is defined as a time-window of data objects. $W[i,j] = (x_i, x_{i+1}, ..., x_j)$, where $i < j$. There are different types of time-windows: landmark window, sliding window, fading window and tilted-time window.

**Landmark window**

In the landmark window, we are interested in the entire streaming data from starting time instant 1 to the current time instant $t_c$; the window is $W[1, t_c]$. Using the landmark window, all transactions in the window are equally important; there is no difference between past and present data. However, as streaming data evolves continuously, the model built with old data objects may become inconsistent with the new ones. In order to emphasize recent data, one may apply the sliding window, tilted window, or fading window variants.

**Sliding window**

In the sliding window variant $W[t_c - w + 1, t_c]$, we are only interested in the $w$ most recent transactions; the others are eliminated. The mining result is dependent on the size of the window $w$. If $w$ is too large and there is a concept drift, the window possibly contains outdated information and the accuracy of the model decreases. If $w$ is small, the window may have deficient data, the model over-fits and suffers from large variances. Previous work considers a fixed value for the size of the sliding window specified by users or an experimental value. Recently, there are proposals for flexible sliding windows where the size of the window changes according to the accuracy of the model [79, 80]. When the accuracy is high, the window extends; and when the accuracy is low, the window shrinks.

**Fading window**

In the fading window variant, each data object is assigned a different weight according to its arrival time so that new transactions receive higher weights than old ones [33, 35, 81]. Using the fading window, we reduce the effect (importance) of old and outdated transactions on the mining results. A decreasing exponential function $f(\triangle t) = \lambda^{\triangle t}$ $(0 < \lambda < 1)$ is usually used in a fading model. In this function, $\triangle t$ is the age of a data object that is equal to time difference between the current time and its arrival time. The fading window needs to choose a suitable fading parameter $\lambda$, which is typically set in the range [0.99, 1] in real applications [33, 35, 81].

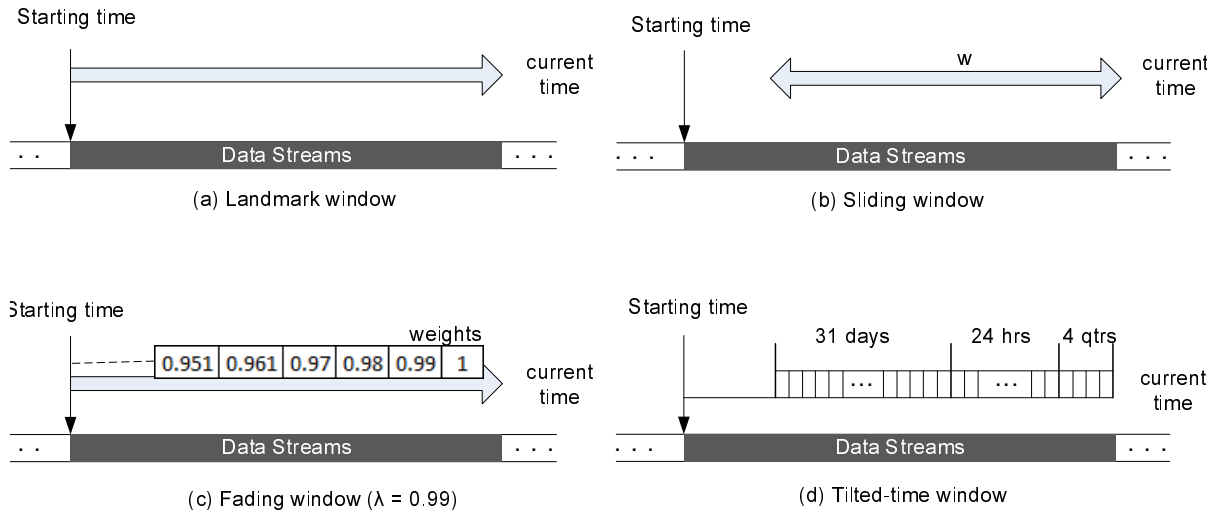



Figure 2.3: Examples of time-windows.

### Tilted-time window

The tilted-time window variant has characteristics somewhere between the fading window and sliding window variants [5, 28]. It applies different levels of granularity with regard to the recency of data. One is more interested in recent data at fine scale than long-term data from the past at coarse scale. Tilted-time window approximately stores the entire dataset and provides a nice tradeoff between storage requirements and accuracy. However, the model may become unstable after running for a long time. For example, the tree structure in FP-Stream [48] will become very large over time, and the process of updating and scanning over the tree may degrade its performance. Similarly, the micro-structures in On-Demand Classification [82] will become larger and larger and this may give rise to the problem of low-purity clustering with large micro-clusters [30].

Figure 2.3 shows examples of four different time-windows. For fading window, $\lambda$ is set to 0.99; the weights of data objects are decreased. For tilted-time window, we store the 4 most recent quarters of an hour, then the last 24 hours, and last 31 days.

### 2.2.3 Computational Approaches

Besides various time-window variants, there are computational approaches to process the streaming data.

**Incremental Learning**

Incremental learning is a computational approach for streaming data [12, 22, 32, 37–39, 67, 83, 84]. In this approach, the model incrementally evolves to adapt to incoming data. There are two ways to update the model: by-window and by-data-instance. Street *et al.* deployed an ensemble of classifiers for streaming data [67]. It evaluates a window of incoming data and adapts the model by adjusting the weight of each classifier or replacing an old classifier with an updated one. The incremental approach has the advantage of providing mining results instantly, but it requires much computational resources.

**Two-phase Learning**

Two-phase learning, also known as online-offline learning, is another computational approach in streaming data [13, 14, 16, 28, 33, 35, 36, 81, 82, 85]. The basic idea is to divide the mining process into two phases. In the first phase (online phase), a synopsis of data is updated in a real-time manner. In the second phase (offline phase), the mining process is performed on the stored synopsis whenever a user sends a request. The two-phase learning approach is able to process streaming data at very high speed. However, its limitation is that users must wait until the mining results are available.

## 2.3 Traditional Classification

**Decision Tree**:

The decision tree classifier is a popular data classification method. It aims to construct a decision tree where each interior node represents one attribute, and an edge to its child node corresponds to the possible values of its attribute. Each leaf stands for a group of

data objects whose attribute values satisfy the specified ranges represented by the path from the root to the leaf.

Most decision trees are recursively constructed in a top-down manner with the greedy approach [56–58]. There are usually three basic steps to construct a decision tree: (i) select the best attribute to split, (ii) create a node and label it with the splitting attribute, and (iii) loop until certain conditions are met, such as all the training data belong to the same class or entropy threshold is exceeded.

Decision tree algorithms usually differ in how they select the splitting attributes and prune the tree when it becomes too large. There are heuristics to find the "best" splitting attributes, such as information gain in ID3 [57], gain ratio in C4.5 [58] (ID3's successor), and the Gini index [56]. Tree pruning methods are used to prevent data over-fitting by removing redundant, least reliable branches. Thus, the pruned tree becomes smaller and simpler, and may classify testing data more quickly and accurately. Pruning techniques can be applied; for example, pre-pruning to halt the tree's construction early; post-pruning to remove sub-trees from a "full grown" tree; and pessimistic pruning that uses error rate to decide when to stop the sub-tree pruning.

Decision trees are simple to understand and interpret. They are able to handle both numerical and categorical data, and perform well with large data. However, decision trees suffer from local optimal problems due to the use of heuristics; it may create over-complex trees that do not generalize the data properly.

**Bayesian Classification**:

Bayesian classifiers are statistical classifiers that are based on Bayes' theorem to predict class membership probabilities [59, 60]. Naive Bayes, an instance of Bayesian classifiers, is well-known for its simplicity and low computational cost. It assumes the class conditional independence described as follows:

Given a data object $x$ with $d$ attributes $x = (a_1, a_2, \ldots, a_d)$, $x$ is assigned to the class having the highest posterior probability $P(c_i|x)$ computed as follows:

$$P(c_i|x) = \frac{P(x|c_i)P(c_i)}{P(x)} = \frac{P(c_i)}{P(x)}P(x|c_i) \quad (2.1)$$

$$= \frac{P(c_i)}{P(x)}\prod_{k=1}^{d} P(a_k|c_i),\ c_i \in \{c_1, c_2, c_3, \ldots, c_m\}, \quad (2.2)$$

where the values $P(a_k|c_i)$ and $P(c_i)$ are estimated from the training data. Naive Bayes works in an incremental manner in that it simply increases the relevant counts, and makes predictions based on posterior probabilities whenever it receives a new data object.

**Neural Networks**:

Neural networks consist of a set of input/output units where each connection has a modifiable weight. These weights are adjusted during the learning phase to predict the correct label of testing data. The central idea of neural network is to extract a nonlinear function by using linear combinations of the inputs as derived features. Therefore, the crucial problem of neural networks is to set the weights based on training data and desired output. Backpropagation algorithm is a common method of training neural networks that minimizes the mean squared error between the network's prediction and the actual value.

A *multilayer feed-forward neural network* (FFNN) that is based on the backpropagation algorithm has an input layer, one or more hidden layers and an output layer [63]. All weights and biases in the feed-forward neural network are initialized with small random numbers. Then, each training instance passes through the network and produces a predicted value. The error between the predicted values and actual values is propagated backward by updating the weights and biases accordingly. Neural networks are robust to noise, produce highly accurate results and can be parallelized easily. However, they suffer long training times and require a number of parameters that are best determined empirically, such as the initialized values of weights and biases, the number of hidden layers, and a learning rate. Poor result interpretation is another disadvantage of neural networks; this is typically mitigated by network pruning techniques.

**Support Vector Machine** (SVM):

The Support Vector Machines concept was first introduced by Vapnik *et al.* in 1992 [64]. It has become a principled and powerful classifier outperforming most other systems in many applications. SVM finds the maximum marginal hyperplane (MMH) to best separate training data of different classes. Training objects falling on the MMH are called *support vectors*. The problem of finding the MMH is known as a convex optimization and is solved by using Lagrangian formulation.

When data are linearly inseparable, SVM transforms data into a higher dimension, and finds a feasible linear separation. To reduce computational cost in the transformed space, *kernel tricks* are used to compute the equivalent dot products in the original input space without caring about the transforming function. SVM usually requires a long training time; however, it is highly accurate and less prone to over-fitting.

$k$**-Nearest-Neighbor Classifier** ($k$-NN):

$k$-NN classifier is a lazy learner that does not construct a general model. The $k$-NN classifier does less work during the training step and does more work during the testing step. It indexes training data and simply assigns testing data to the dominant class of their $k$ nearest neighbors. The distance measure between two data instances can be diverse: Euclidean distance, Manhattan distance, Jaccard coefficient, etc. Although this algorithm is simple and robust to noises, it requires much space to store all training data and is computationally expensive, as it needs to compute and compare distances from testing data to all others. Hence, it should be speeded up by applying efficient storage structures and pre-sorting techniques. Liangxiao *et al.* introduced a good survey of improving $k$-NN [65].

**Ensemble Classifiers**:

Many classifiers have been proposed for inducing models from data. Each classifier has its own strengths and weaknesses. In theory, there is no specific algorithm that

works well for all kinds of datasets. Developing a new algorithm that outperforms those existing classifiers for most types of datasets is really hard or even infeasible. However, an easier way is integrating existing classifiers to create a more powerful classifier. Ensemble learning, a process to construct accurate classifiers from an ensemble of weak classifiers, has attracted extensive research in the past decade. In general, these methods vary in the way they construct various classifiers and combine their predictions. The *first* step of constructing a group of classifiers can be differentiated according to the dependencies among classifiers. The independent approach trains classifiers randomly and can be easily parallelized, for example, bagging [86], random subspace [87], and random forest [88]. The dependent approach constructs a new classifier while taking advantage of knowledge obtained during the construction of past classifiers, such as AdaBoost [66], AdaBoost.M2 [89], and Gradient boosting [90]. In the *second* step of combining the classifiers' predictions, majority voting is one intuitive method to choose the dominant decision [86–88]. As majority voting cannot guarantee that the voting result will be better than the best individual one, the weighting method is introduced which assigns competent classifiers higher weights, such as performance weighting [19, 66, 89, 90], Naive Bayes weighting [60], and entropy weighting [91]. Various ensemble classifiers methods have been proposed, and they are principally different from the above two steps. Here, we only present three basic ensemble methods: Bagging, Boosting and AdaBoost.

- *Bagging*: Bagging (bootstrap aggregating) was introduced by Breiman [86]. In Bagging, each classifier is trained on a different bootstrap replicate of the training set. A bootstrap replication is generated by random sampling with replacement from the original dataset. Bagging uses voting method to combine its members' classification results.

- *Boosting*: Boosting aggregates many weak learners into a strong ensemble by filtering the training data [92]. Data objects that are misclassified or disagree on

previous classifiers are treated as training objects for the next classifier. The original boosting works as follows:

  - Select a random sample without replacement and train a learner $C_1$.
  - Train weak learner $C_2$ with filtered examples, half of which are misclassified by $C_1$.
  - Train weak learner $C_3$ only with examples that $C_1$ and $C_2$ disagree on
  - Combine the predictions of $C_1$, $C_2$ and $C_3$ by majority voting.

- *AdaBoost*: AdaBoost, short for Adaptive Boosting, is the most well-known and successful boosting algorithm in practice [66]. It adapts to the errors of weak classifiers and set the weight for each classifier instead of majority voting. Adaboost re-weighs the samples in a manner that wrongly classified samples retain their weights while correctly classified samples relatively decrease their weights by $\frac{\epsilon}{1-\epsilon}$, where $\epsilon$ is the error rate of the previous classifier. To combine classification results, Adaboost sets weights to classifier members according to their error rates, $\alpha = log\frac{\epsilon}{1-\epsilon}$. AdaBoost algorithm reduces error on the training set to zero with a number of iterations. Currently, it is still unclear about AdaBoost's capability to generalize testing data, which is not included in the training data.

Moreover, traditional classifiers have advantages and limitations in Table 2.2. For example, decision trees are easy to understand, robust to noise, efficient, and can remove redundant attributes; however, it suffers an over-fitting problem when the tree has many levels and the classification rules become difficult to interpret. The lazy-learner can be implemented easily, but it consumes much memory and is susceptible to high-dimensional data streams. An ensemble is highly accurate and easy to implement; however, it is mostly based on heuristics and is not based on strong theoretical foundation.

Table 2.2: Advantages and limitations of classification algorithms

| **Algorithm** | **Advantages** | **Limitations** |
|---|---|---|
| Decision Tree | Easy to understand<br>Robust to noises<br>Low computational cost, even for very large training datasets<br>Can deal with redundant attributes. | Suffer an over-fitting problem when the tree has many levels and the rules become difficult to comprehend. |
| Bayesian Classifier | Simple, efficient, effective and robust to noisy data | Independence assumption is often violated in the real world. |
| Neural Network | Well-generalizing capability.<br>Can solve dynamic or non-linear problem. | Inability to interpret the learned model (black-box).<br>High complexity. |
| SVM | Provide a unique solution.<br>Can solve non-linear problem with implicit kernel.<br>Work well even when training sample are bias. | Depend on the choice of the kernel.<br>High complexity.<br>Difficult to design multi-class SVM classifiers. |
| Lazy Learner: $k$-NN | Easy to implement. | Large storage requirements.<br>High complexity.<br>Highly susceptible to high-dimensional data. |
| Ensemble | High accuracy.<br>Easy to implement. | Based on heuristics and lack of solid theory. |

## 2.4 Time Series Classification

In time series classification, the most difficult challenge is that there is no explicit feature in time series data. Most of the classifiers, for example, Naive Bayes, Decision Trees, SVMs, and neural networks, require input data as vectors of features. Therefore, it is necessary to transform a time series into a set of features. Moreover, the dimensionality

of features should not be high to make the computation cost affordable. There are two types of time series classification, including *whole-series classification* and *motif-based classification*. In the first category, a whole time series is considered as a vector of features, and classification models are performed directly. This approach is suitable for short time series and requires dimension reduction techniques for long time series. The motif-based classification is usually done in two steps: *i)* motifs (or frequent occurred subsequences) are extracted, then *ii)* each time series is represented as a vector of features using motifs so that a traditional classifier can be applied.

### 2.4.1 Whole-series classification

Time series classification is a well-examined data mining task due to its vast practical applications. There are a lot of papers on time series classification, including decision tree (*TClass* [93]), neural networks (*PREMONNS* [94]), Bayesian networks (*mixed-state DBN* [95]) and support vector machines (*Zeus* [96]). Kadous *et al.* introduced two-step classifier *TClass* for multi-variate time series [93]. In the first step, global features of time series are extracted such as its mean, minima, maxima and trend. The second step is a clustering of the features; then centroids of these clusters are used as inputs for a decision tree classifier. Eads *et al.* proposed a hybrid classifier *Zeus* that employs evolutionary computation for feature extraction, and a support vector machine for classification with the selected features [96]. Recent research has shown that the simple nearest neighbor classifier (1-NN) is one of the best-performing time series classification techniques [6–8, 97]. Moreover, the performance of 1-NN classifiers is critically dependent on the choice of the distance measure. The Euclidean distance is widely used for simple time series classification. Given two time series $s$ and $s'$, the Euclidean distance between them is:

$$distance(s, s') = \sqrt{\sum_{i=1}^{L}(s[i] - s'[i])^2}$$

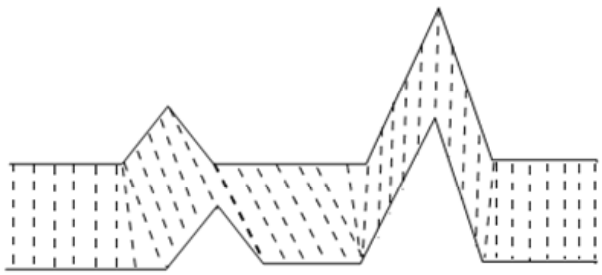


Figure 2.4: Dynamic Time Warping.

Keogh *et al.* [97] shows that 1NN classifier with Euclidean distance can achieve competitive accuracy, compared to other complex distance measures. However, the Euclidean distance requires two sets of time series to have the same length and is sensitive to distortions in the time dimension. Dynamic time warping distance (DTW) is a classical distance measure that can overcome this problem. DTW differs from Euclidean distance by finding an optimal match between two sets of time series data. DTW can align two time series datasets, allow the "stretching" and "shrinking" of parts of the time series datasets along the temporal axis to find their minimum distance. Figure 2.4 illustrates the idea of DTW.

Dynamic programming can be used to compute DTW; however, it has quadratic time complexity and is costly for large datasets. To improve DTW computation, Ratanamahatana and Keogh eliminate unnecessary calculation by limiting the warping window size [98]. This idea is later developed to a lower-bounding of DTW, called LB_Keogh [99]. Furthermore, the nearest neighbor classifier is simple and does not require extensive parameter tuning, it needs to store and search the entire dataset that results in high time and space complexities. There are some attempts to speed up the nearest neighbor classifier. Dimension reduction techniques are also used to accelerate time series classification. For example, symbolic-based methods reduce the length of time series by aggregating consecutive values into a single symbol [43]. Another speed-up technique is "instance selection" (also known as numerosity/prototype selection) that reduces the size of train-

ing dataset by keeping only the most informative instances. Ranking instances based on label matching with their nearest neighbors, Xi *et al.* proposed *FastAWARD* algorithm, which discards a large fraction of the training data to improve the performance of the 1-NN classifier and adjusts the warping window dynamically at the same time [7]. Similarly, Buza *et al.* proposed the *INSIGHT* framework for instance selection based on coverage graphs and hubness [8]. Recently, Bagnall *et al.* highlight the importance of data transformation in time series classification and propose a transformation ensemble classifier (TEC) for time series classification [100]. Their classifier that have constructed on different transformations of time series, including power power spectrum (PS), autocorrelation function (ACF) and principal components (PCA) can achieve significant accuracy over classifiers constructed in the time domain.

### 2.4.2 Motif Discovery

Motif discovery in time series is an important data mining task that has received considerable attention from the research community. Generally, there are two types of motif discovery: exact motif discovery and approximate motif discovery.

*Exact motif discovery:*

Lin *et al.* first proposed the problem of finding $k$-motifs [3, 101]. Given a predefined length $w$ and a positive value $R$, a subsequence $M$ beginning at $q$ is a match subsequence of a subsequence $C$ beginning at $p$ if their Euclidean distance is smaller than $R$: $D(C, M) < R$. Moreover, $M$ is considered as a trivial match of $C$ if there does not exist a middle subsequence $M'$ beginning at $q'$, such that $D(C, M') > R$ and either $q < q' < p$ or $p < q' < q$. The $k$-motif problem has been generalized to find the top $k$ significant motifs that has the highest count of non-trivial matches. Several algorithms have been proposed to solve this problem [40, 41, 102].

Mueen and Keogh proposed an online algorithm for finding the top motif (the nearest pair of non-overlapping sequences) of streaming time series [103]. The algorithm

maintains a list of subsequences, each of which has a sorted list of the nearest neighbors (N-list) and a sorted list of the reverse nearest neighbors (RNN-list). The algorithm follows the sliding window setting. When a new subsequence enters the window, the oldest subsequence is discarded, and the list is updated. They applied several techniques to reduce memory space and time complexity, for example: N-list of a subsequence only holds subsequences that arrived earlier than it, N-list can be stored in the strict increasing order of the time-stamp, and an early termination technique is used to reduce time to create N-list.

Lam *et al.* extend the above algorithm for the top-k motifs discovery [104]. Similarly, this algorithm maintains a list of subsequences with their nearest neighbors in a sliding window. However, for each subsequence, the algorithm stores a list of forward top-k nearest neighbors, each of which has a later time-stamp than it. Thereby, a query of finding top-k motifs is performed by first sorting all the pairs in the list increasingly according to Euclidean distance. Furthermore, they only store those time-stamps that form promising pairs to save disk space and to accelerate the algorithm.

Since the above algorithms differently define top-k motifs as the top-k nearest pairs of non-overlapping subsequences, they cannot capture the whole structure of a time series. For example, if a time series has three subsequences with a similar shape $(A_1, A_2, A_3)$ and two subsequences with another shape $(B_1, B_2)$, a top-3 motif algorithm may output only all pairs of the first three subsequence $(< A_1A_2 >, < A_1A_3 >, < A_2A_3 >)$.

*Approximate motif discovery:*

Due to the quadratic complexity of exact motif discovery, many time series representations have been proposed to reduce the dimensionality of the original data, including Discrete Fourier Transform (DFT) [105], Discrete Wavelet Transform [106], Piecewise Linear Approximation (PLA) and Piecewise Aggregate Approximation (PAA) [107]. Keogh and Kasetty claimed that there was not much difference in terms of indexing power among

these representations [97]. Nevertheless, each representation has different strengths and weaknesses. For example, DFT is not good for bursty time series; wavelet has low complexity with the multi-resolution property, but is only suitable for time series whose length is an integer power of two, and PLA is possibly computable with provable error bounds; but its visualization is not very smooth, and thus it is difficult to reconstruct the time series [108].

Lin *et al.* proposed a SAX (Symbolic Aggregate approXimation) method for dimensionality reduction [3]. The basic idea of SAX is to use a sliding window to segment a time series into continuous subsequences, which are then converted into lists of symbols based on the PAA [107] representation. SAX is a popular approach and has been proven to be very useful in many diverse applications such as biological analysis [109, 110], hazardous weather prediction [111], biometric recognition [112], and motion data analysis [113,114]. A SAX word is represented by a tuple $SAX(T, w, a)$ consisting of a time series $T$, a number of symbols $w$, and an alphabet size $a$. For example, if $w$ is 3 and $a$ is 2, there are eight possible SAX words of a time series $T$:

$$SAX(T,3,2) \in \left\{ \begin{array}{lll} \{0,0,0\}, & \{0,0,1\}, & \{0,1,0\}, \\ \{0,1,1\}, & \{1,0,0\}, & \{1,0,1\}, \\ \{1,1,0\}, & \{1,1,1\} & \end{array} \right\}$$

SAX is fast and effective on dimensionality reduction. Moreover, SAX also supports a MINDIST function that returns a lower bound of Euclidean distance between the two original time series. Lin *et al.* later introduced a visualization toolkit VizTree for SAX [42,43]. Due to its practicality, SAX made a huge impact in industry and academia with many extensions and applications. Chiu *et al.* used a random projection technique to overcome SAX's limitations on poor scalability and the inability to discover motifs in the presence of noise [40]. However, the original representation of SAX is ambiguous. For example, given a SAX word $\{0, 0, 1\}$, we cannot be sure about its alphabet size (although we know it is at least 2). To avoid this ambiguity, Shieh and Keogh proposed iSAX, an

extension of SAX, to support multi-resolution symbolic representation [44]. Each iSAX word further includes a cardinality of an alphabet size, $iSAX(T, 3, 2) = \{0^2, 0^2, 1^2\}$. Using an index structure like R-tree [115], iSAX can index massive time series, and support fast approximate search for similar motifs by increasing or decreasing the iSAX resolution. Camerra *et al.* introduced an index structure for very large time series datasets with efficient mechanisms for bulk loading and node splitting [45]. The author also demonstrated experiments of indexing up to one billion different time series datasets. However, the above algorithms assume that the length of discovered motifs is known in advance. Despite its simplicity, this assumption suffers a major problem that only motifs with such length can be discovered. However, meaningful motifs may exist with various lengths throughout a time series as seen in the previous section.

Nunthaid *et al.* iterated the SAX algorithm many times with different values of $w$ to discover motifs with different lengths. This approach is not optimal and inefficient, especially with large datasets. Li *et al.* proposed a grammar-based algorithm to discover approximate variable-length motifs [46]. A time series is first discretized by using the SAX algorithm. A string compression algorithm, *Sequitur*, is then used to extract grammar rules from the sequence of discrete symbols [116]. The algorithm is quite efficient as it is inherited from the SAX and *Sequitur* algorithms. Since the *Sequitur* algorithm focuses on compressing a sequence as much as possible, it may skip useful motifs that are useless for the compression process. Therefore, discovered motifs are not complete. Tang *et al.* [10] first segmented a time series into a list of equal-length subsequences. They computed the Euclidean distance between every two subsequences and formed a similarity matrix. To find longer motifs, the algorithm identifies the matching pairs of subsequences of the motifs, which results in up-right moving tendencies in the similarity matrix. However, this algorithm is inapplicable to streaming time series as it requires much memory and many passes through the time series.

### 2.4.3 Motif-based Classification

Dzeroski *et al.* introduced a concept of predictive clustering trees ($PCT$) and used it to classify time series gene expression data [117]. A clustering tree is a decision tree, whose each node represents one cluster. Predictive clustering is an approach uses clustering results to make predictions. A new instance is assigned a cluster and is received a prediction from that cluster. The authors first applied $k$-mean clustering algorithm with a qualitative distance, then extracted motifs of each cluster to construct a predictive clustering tree based on pairwise distances among clusters.

In bioinformatics, there are many motif-based approaches for protein classification. They often use the occurrence count of each motif in a time series as a similarity measure between the motif and the series. These motifs are later used as input for advanced classification techniques. For example, Ferreira *et al.* found rigid gap motif of a certain minimum length. Combining them with the Bayes theorem, they proposed multi-class sequence classifier (MCSC) [118]. Similar, Kunik *et al.* reported good results for motif-based SVM classification (named MEX) of enzymes data [119]. Since frequent pattern mining is similar to motif discovery, some ideas from this field have been adapted to motif discovery in time series. For example, Buza *et at.* introduced a new class of motifs, generalized semi-continuous motifs (GSC-motifs) and adapted the downward closure property to discover GSC-motifs. Motif discovery process starts with short ones and grows to larger ones step by step. They applied tree data structure to increase the algorithm's efficiency [120].

Ye *et al.* introduced time series shapelets that are time series subsequences, which are maximally representative of a class [2]. A variant of the traditional decision tree is constructed to classify time series. All possible subsequences with their Euclidean distances to each training time series are considered as attributes. A tree node is created in a similar way that the subsequence with the highest information gain is chosen, and the dataset

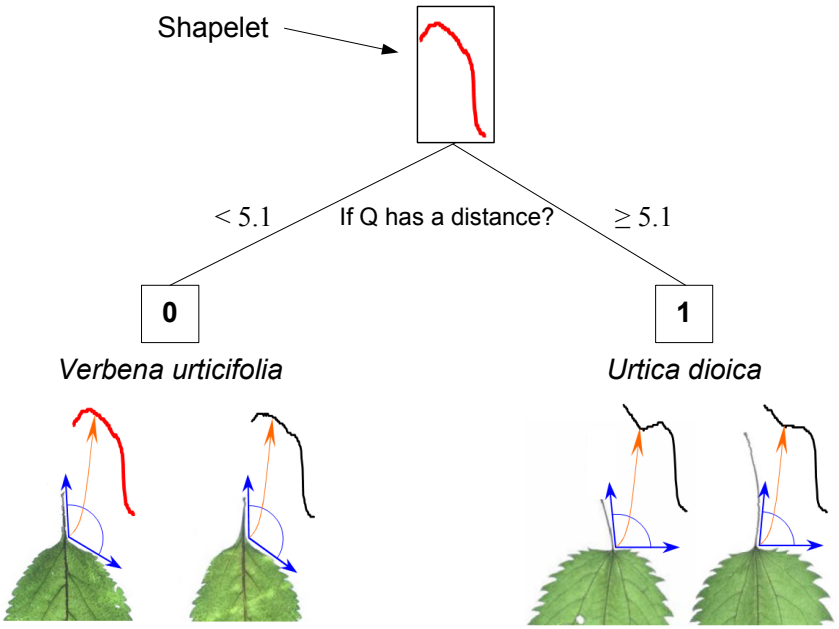



Figure 2.5: Example of a shapelet-based decision tree [2].

is divided into two smaller datasets based on its optimal split distance. Figure 2.5 shows an example of a decision tree that used shapelets to classify two species of leaves, *Urtica dioica* and *Verbena urticifolia*. The discovered shapelet indicates differences between the two species, where a stem of a Verbena urticifolia leaf has a larger angle than a stem of a Urtica dioica leaf. The optimal split distance is empirically determined as 5.1 . If a distance between a testing leaf and the shapelet is less than 5.1, the leaf is classified as Verbena urticifolia; else, it is classified as Urtica dioica.

To speed up the process of finding shapelets, the algorithm uses two simple and effective techniques: subsequence distance early abandon and admissible entropy pruning. As the Shapelet algorithm does not scale up well with large datasets, Mueen *et al.* proposed another approach termed Logical-Shapelet, which uses caching techniques for better performance and enhances shapelet representation with conjunction and disjunction operations [9]. Although the shapelet-based approaches are effective and can learn the inherent structure of the data, they are off-line classifiers and thus, cannot be used

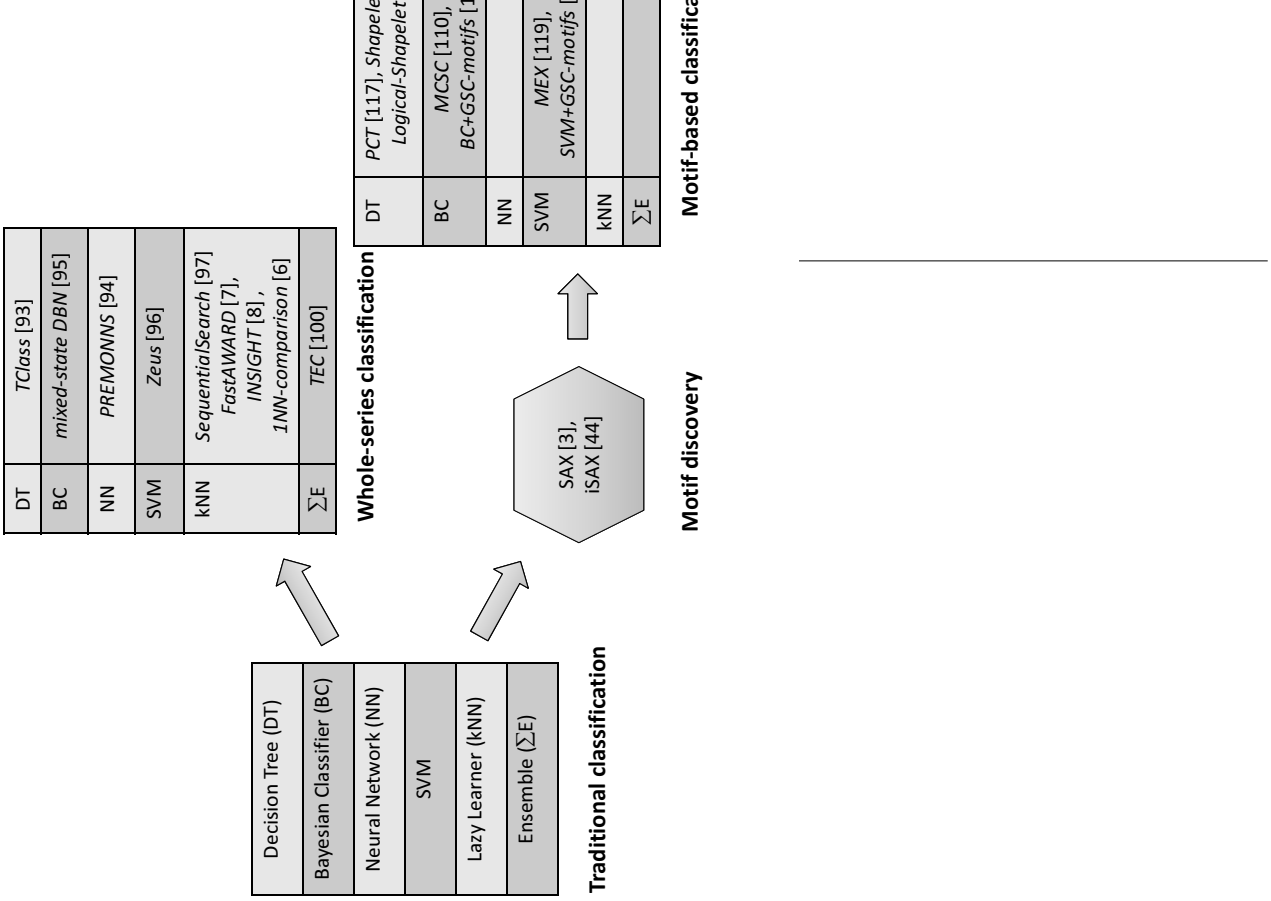



Figure 2.6: The relationships between traditional classification methods and time series classification methods.

for streaming time series.

### 2.4.4 Overall analysis

Figure 2.6 illustrates the relationships between traditional classifiers and time series classifiers. Traditional classifiers are on the left while two types of time series classification are on the right. Since there is no explicit feature in time series data, the whole-series classification uses the whole-series or extracts its global statistics as inputs for traditional classifiers. However, this approach may suffer computation cost with large time series dataset. The second approach first discovers motifs that are relevant to different time series classes. Then, traditional classifiers are constructed based on these motifs.

We observe that the nearest neighbor classifier and decision tree are the two most popular methods in time series classification. The nearest neighbor is simple and easy to implement; however, it requires advanced indexing techniques to speed up and appropri-

ate distance measures to achieve good results. Decision tree is better than the nearest neighbor by providing interpretable results. Nevertheless, current decision tree classifiers need to obtain all time series instances when they construct classification models. Hence, it still has unexplored potential for the design of algorithms for streaming time series where observations continually arrive.

**Limitations of state-of-the-art work and thesis contributions:**

Although several time series classifiers have been proposed, there is insufficient research on streaming time series classification. Most of the time series classifiers are not able to work in a streaming manner since they have to read the training dataset repeatedly, require long training time, and cannot update their models with continuously arriving data. Compared to whole-series classification, motif-based classification is light-weight and more suitable for working with streaming time series. However, state-of-the-art motif-based classifiers can only find motifs with a predefined length, which greatly affects their performance and practicality. To overcome these limitations, we introduce the definition of *closed motifs* and propose a novel time series classifier that is based on closed motifs in Chapter 3. Our proposed algorithm has been shown to be more accurate than state-of-the-art time series classifiers on several large datasets in diverse domains such as video surveillance, sensor networks, and biometrics.

## 2.5 Data Stream Classification

Data stream classification follows the general model proposed in Figure 2.2. While inheriting traditional classification, most data stream classifiers employ different computational approaches and time windows to handle high-speed data streams. For simplicity, we introduce various data stream classifiers with the same category of traditional classifiers in Section 2.3. We further outline their mutual relationships as well as present an overall analysis of these algorithms.

**Decision Tree**:

The Hoeffding tree is a decision tree classifier for data streams [11]. Traditional decision trees need to scan the training data many times to select the splitting attribute. However, this requirement is infeasible in the data stream environment. To overcome this limitation, the Hoeffding bound is used to choose an optimal splitting attribute within a certain amount of arriving data objects. Given $N$ independent observations of a random variable $r$ with range $R$ and computed mean $\overline{r}$, the Hoeffding bound guarantees that the true mean of $r$ is at least $\overline{r} - \epsilon$ with probability $1 - \delta$, where $\delta$ is a user-specified parameter.

$$P(E[r] \geq (\overline{r} - \epsilon)) \geq 1 - \delta,\ \epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2N}}$$

Let $G(X_i)$ be a heuristic measure to select a splitting attribute. After receiving $N$ observations, $X_a$ and $X_b$ are the best and the second best splitting attribute. In case of Hoeffding tree, the variable $r$ is now considered as $r = \Delta G = G(X_a) - G(X_b)$. If $\overline{r} = \Delta\overline{G} = \overline{G}(X_a) - \overline{G}(X_b) > \epsilon$, where $\epsilon$ is computed from the above equation, we say that this difference is larger than $(\overline{r} - \epsilon) > 0$ with confidence $1 - \delta$. Then, the attribute $X_a$ is selected to build the tree.

The Hoeffding tree algorithm is an incremental algorithm, which satisfies the single-pass constraint of data stream mining. For each new arriving data, Hoeffding tree algorithm use Hoeffding bounds to check whether the best splitting attribute is confident enough to create the next level tree node.

The Hoeffding tree algorithm has high accuracy and works well with large datasets. However, it is not able to handle concept drifts in data streams as no node can be changed once created. CVFDT (Concept-adapt Very Fast Decision Tree learner) is an extension of the Hoeffding tree to address concept drifts in data streams [12]. CVFDT maintains sufficient statistics at every tree node to monitor the validity of its previous decisions.

When data come, it continually updates the statistics stored in tree nodes. Using a sliding window, CVFDT removes the effect of outdated data by decreasing the corresponding statistics at the tree nodes. It periodically scans the tree nodes to detect concept drifts. If concept drift appears, CVFDT concurrently grows alternative branches with the new best attribute and prunes the old branches with alternative branches if it becomes less accurate.

**Bayesian Classification**:

Seidl *et al.* proposed a novel index-based classifier called Bayes tree [13]. Adapted from the R*-tree, Bayes tree generates a hierarchical Gaussian-mixture tree to represent the entire dataset. Each tree node contains statistics of belonging data objects, including a minimum bounding rectangle, the number of data objects, linear sum, and quadratic sum of all data objects.

To solve a multi-labeled classification problem, a single Bayes tree is constructed for each class. For each testing data object $\mathbf{x}$, the algorithm tries to find a set of prefix-closed nodes $E_i$, called frontier, in every Bayes tree. The probability that $\mathbf{x}$ belongs to class $c_i$ is computed as follows:

$$P(c_i|x) = (\sum_{e_s \in E_i} \frac{n_{e_s}}{n} g(x, \mu_{e_s}, \sigma_{e_s})) * P(c_i)/P(x),$$

where $e_s$ is a tree node in the frontier set $E_i$; $n_{e_s}, \mu_{e_s}$ and $\sigma_{e_s}$ are respectively the number of data object, the center and the deviation of tree node $e_s$. The testing object $x$ is labeled with the class having the maximum probability. The Bayes tree is an anytime classifier that can provide a decision after a very short initialization, and later provide more accurate decisions when more time is available by selecting the more precise frontier.

The Bayes tree is later extended to the MC-tree [14] by Kranen *et al.*. The MC-tree combines all classes in a multi-Gaussian tree and needs only one step to refine all class models simultaneously rather than $|C|$ steps in Bayes tree, where $|C|$ is the number

of classes. Moreover, MC-Tree optimizes tree construction by using multi-dimensional scaling (MDS) [121] to transform the data space. It is claimed to outperform Bayes tree with higher accuracy of up to 15%.

**Neural Network**:

Leite *et al.* propose an evolving granular neural network (eGNN) supported by granule-based learning algorithms to classify data streams [84]. There are two phases in an eGNN. In the first phase, an eGNN use T-S neurons to construct information granules of incoming data. Then, the neural network is built on the information granules rather than the original data in the second phase. A granule associated with a class label is defined by triangular membership functions, which are later evolved to accommodate new data. The weights are decreased by a decay constant; this process helps to reduce the degree of importance of outdated granules. Basically, an eGNN uses class exemplars in the form of information granules to perform classification tasks. When testing data comes, a max-neuron selects the best-fit granules and assigns their labels as the prediction labels of the testing data. The eGNN is able to tackle classification problems in continuously changing environments. However, the requirement of long training time is still a cost that limits the ability of an eGNN to work with a massive dataset. Therefore, only small datasets are used to evaluate the performance of an eGNN in the experiment section of this paper. The authors later extend this work to a more general and efficient semi-supervised approaches [122] to work with partially labeled datasets.

**Support Vector Machines (SVMs):**

SVMs have shown its prominent performance in many machine learning problems with static datasets. However, it is very expensive to use SMVs in large scale application due to its time complexity $O(N^3)$ and memory complexity $O(N^2)$, where $N$ is the number of data objects. To work with a very large dataset, Tsang *et al.* proposed the Core Vector Machine (CVM) algorithm that uses Minimum Enclosing Ball (MEB) to reduce

its complexity [15]. A MEB is a hyper-sphere that represents the set of data objects inside it. The algorithm first finds a representative MEB set that is a good approximation of the original dataset. Then, the optimization problem of finding the maximum margin is directly performed on this MEB set.

Rai *et al.* propose StreamSVM [16], an extension of CVM with a single scan, to work with data streams. In StreamSVM, a MEB has a flexible radius that is increased whenever a new training is added. The algorithm is competitive by providing an approximation result to the optimal one. However, StreamSVM is still unable to detect concept drifts in data streams.

**k-Nearest-Neighbor Classifier**:

On-Demand-Stream is a $k$-NN data stream classifier that extends the CluStream method [82]. It inherits most of the good features in CluStream such as the micro-cluster structure, the tilted-time window, and the online-offline approach. A micro-cluster in On-Demand-Stream is extended with a class label, and it only takes data objects with the same class label. Its offline classification process starts to find the best window of data objects, called the best time horizon. These micro-clusters in the best time horizon are extracted using the addition property of micro-clusters. On-Demand-Stream perform 1-NN classification by assigning a testing data object to the label of the closest micro-clusters.

Inspired by M-tree [123], Zhang *et al.* [17] propose a Lazy-tree structure to index micro clusters (which are called exemplars in the paper). This helps to significantly reduce $k$-NN's classification time from $O(N)$ to $O(log(N))$, where $N$ is the total number of exemplars in the Lazy-tree. The tree consists of three main operations: search, insertion and deletion operations. The insertion and deletion operations add new nodes and remove outdated nodes in a way that guarantees that the tree is balanced. The search operation is used to classify testing data. A *branch-and-bound* technique based on the triangle

inequality filters unnecessary checking exemplars; therefore, it minimizes the number of comparison and accelerates the searching operation.

**Ensemble Classifiers**:

Bagging and Boosting have shown their superior through extensive experiments on traditional dataset. Therefore, many researchers have tried to adapt them to work on data streams.

- *Online Bagging & Boosting*: Oza *et al.* proposed the *Online Bagging & Boosting*, which is one of the first work of adapting traditional bagging and boosting [18]. From a statistical view, each training data object appears $k$ times in training datasets of classifier members with the probability $P(k) = \binom{N}{k}\left(\frac{1}{N}\right)^k\left(1-\frac{1}{N}\right)^{N-k}$, where $k$ is the size of training set and $N$ is the size of dataset. On data streams, we can assume that the number of data objects is unlimited, $N \to \infty$. Therefore, the probability $P(k)$ tends to a $Poisson(1)$ distribution, where $Poisson(1) = exp(-1)/k!$. Within this observation, Oza *et al.* proposed Online Bagging to assign each data object a weight according to $Poisson(1)$ distribution, which is considered as a replacement sampling method. In Online Boosting, the weights of coming data objects and classifier members are adjusted according to the error rates of classifier members at the current time.

- *Weighted Ensemble Classifiers*: With the observation that the expiration of old data should rely on data distribution instead of their arrival time, Wang *et al.* proposed an accurate-weighted ensemble (AWE) classifier with weighting method for mining concept-drifting in data streams [19]. The algorithm constructs and maintains a fix $k$ number of classifiers, which can be C4.5, RIPPER, or naive Bayesian. Processing in a batch-mode manner, it use each new chunk of coming data objects to train a new classifier. Then, the ensemble is formed by selecting the $k$ most accurate classifiers, and the weight of each classifier is set according to its accuracy. This method is stated to be better than a single data stream classifier, such as VFDT, CVFDT. However, it is quite sensitive to the chunk size and the number of classifier members $k$.

In real applications, data streams may contain noise where data instances may be mislabeled or have erroneous values. Zhang *et al.* proposed an *aggregated ensemble* algorithm to tackle the problem of learning from noisy data streams [20, 21]. This approach is a combination of horizontal and vertical ensemble frameworks. The horizontal framework builds a different classifier on each data chunk, while the vertical framework builds different classifiers on the up-to-date data chunk with different learning algorithms. The horizontal framework is robust to noise and can reuse historical information; however, it is unsuitable with sudden drifts, where the concepts of data stream change dramatically. On the other hand, the vertical frame work can produce good results even in case of sudden drifts; nevertheless, it is sensitive to noise. Building classifiers on different data chunks using different learning algorithms, the aggregated ensemble constitutes a Classifier Matrix. The average weighting method is used on this matrix to predict the label for testing data. The author theoretically proved that the aggregate ensemble has less or equal mean squared error of the vertical and horizontal frameworks in average. However, this framework suffers from high time complexity.

- *Adapted One-vs-All Decision Trees (OVA)* [22]: OVA is a recent ensemble method for data streams. It learns $k$ binary CVFDT classifiers, and each classifier is trained to classify instances between a specified class and all remaining classes. To classify a new data object, each classifier is run and the one with the highest confidence is returned. The confident of the classification is set as the proportion of the dominant class at the leaf where the testing object has reached. To achieve a high accuracy, OVA aims to construct an ensemble of CVFDT classifiers with a low error correlation and high diversity. Moreover, OVA quickly adapts to concept drifts as it only needs to update two component classifiers related to the evolving class, and can work well with imbalanced data streams.

- *Meta-knowledge Ensemble* [23]: The high complexity of ensemble learning limits its applicability to time-critical data stream applications in the real world. Zhang *et al.*

proposed a meta-knowledge ensemble algorithm that selects the best classifier for testing data. An Ensemble-tree (E-tree) is constructed to organize base classifiers, each of which occupies a weight and a closed space in the whole data space. Similar to R-tree [115], the E-tree has three key operations, including search, insertion and deletion operations. Hence, it is height-balanced and guarantees a logarithmic time complexity for prediction. The insert operation is used to integrate a new classifier into the ensemble. When the number of classifiers in a node exceeds a predefined value, the node is split into two nodes with a principle that the covering area of the two nodes should be minimized. The deletion operation removes outdated classifier when E-tree reaches its capacity, the tree may need to be re-organize to guarantee its balance. The search operation is used to classify a testing instance $x$. Classifiers whose close space contains $x$ are invoked, a weighted voting method is applied to decide a class label for $x$.

### 2.5.1 Overall analysis

After presenting many data stream classifiers, we introduce a way to summarize all related issues and give readers a broader view of data stream classification. Figure 2.7 shows the relationship between traditional and data stream classifiers. Traditional classifiers are on the left; data stream classifiers are on the right. To categorize data stream classifier, the computational approaches and time windows, are in the middle.

Similarly, we observe that data stream classifiers are extended from traditional classifiers and they apply different computational approaches and time windows. For example, VFDT is an extension of the decision trees for data streams. VFDT uses Hoeffding bound to create a tree node when having a sufficient amount of data. It follows the incremental learning approach and land-mark window. CVFDT is an enhanced version of VFDT that can adapt to concept-drift by constructing alternative trees. Bayes tree is an extension of Bayesian classifier with the two-phase learning approach and land-mark window. MC-Tree improves the Bayes tree with multi-label nodes. eGNN is a neural network algorithm

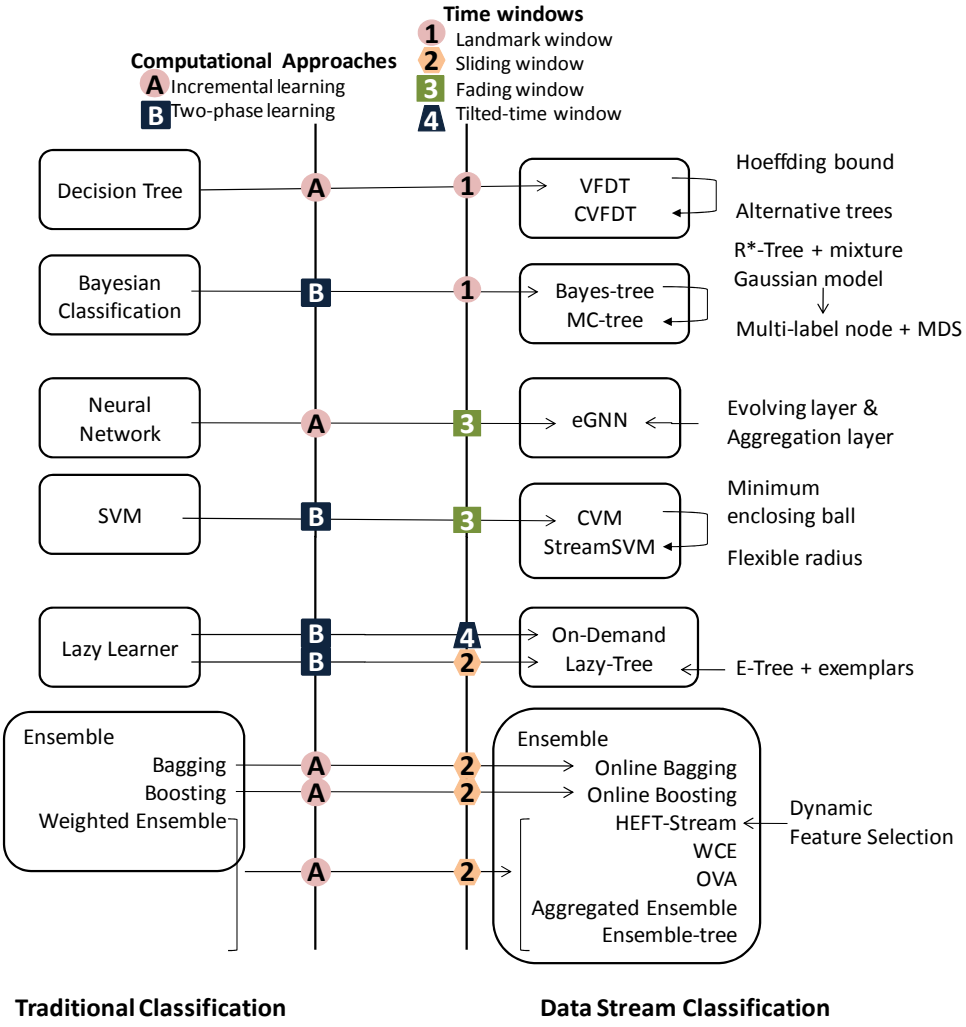



Figure 2.7: The relationships between traditional classification methods and stream classification methods.

that designed to work for data streams. CVM, derived from SVM classifier, follows the two-phase learning approach and fading window. StreamSVM extends CVM by making the radius of minimum enclosing balls flexible. On-Demand, an extension of $k$-NN classifier, applies the two-phase learning approach and tilted-time window. Lazy-Tree is another improved version of $k$-NN classifier with the two-phase learning approach and sliding window. Many ensemble classifiers are designed to work for data streams. Online Bagging & Boosting are extensions of traditional Bagging & Boosting with incremental learning approach and sliding window. There are many weighted ensemble classifiers with

Table 2.3: Capabilities of data stream classification algorithms.

| Algorithm | Bounded Memory | Single-pass | Real-time Response | Concept-drift Adaptation | Concept-drift Classification | High-dimensional Data |
|---|---|---|---|---|---|---|
| VFDT [11] | √ | √ | √ | | | √ |
| CVFDT [12] | √ | √ | √ | √ | | √ |
| Bayes tree [13] | √ | √ | √ | √ | | |
| MC-tree [14] | √ | √ | √ | √ | | |
| eGNN [84] | √ | √ | | √ | | |
| CVM [15] | √ | √ | | √ | | |
| StreamSVM [16] | √ | √ | | √ | | |
| On-Demand [82] | √ | √ | √ | √ | | |
| Lazy-Tree [17] | √ | √ | √ | √ | | |
| Online Bagging & Boosting [18] | √ | √ | √ | | | |
| AWE [19] | √ | √ | √ | √ | | √ |
| OVA [22] | √ | √ | √ | √ | | √ |
| Aggregated Ensemble [21] | √ | √ | √ | √ | | |
| Ensemble-tree [23] | √ | √ | √ | √ | | √ |

different weighting strategies. For example, aggregated ensemble uses average weighting method (voting); while AWE, OVA, and Ensemble-tree apply weighting method based on the accuracy of classifier members. They follow the incremental learning approach, and can be loosely considered as applying a sliding window as they typically remove the least accurate classifier member.

Moreover, we evaluate capabilities of data stream classifiers in Table 2.5.1. Although many classifiers have been proposed, they are typically unable to satisfy all constraints of data stream mining at the same time, which are discussed in Section 2.2. We observe that all classifiers satisfy the bounded-memory and single pass constraints. eGNN, CVM,

and StreamSVM cannot respond in a real-time manner, as they require much time for training process. For the concept-drift constraint, we establish two levels of how the algorithms respond to concept-drifts. The first level, *concept-drift adaptation*, means that an algorithm may update with new concepts and remove outdated concepts. Most algorithms deploy the time windows to update fresh information. Therefore, they satisfy this criterion. However, those applying the landmark window do not satisfy. The second level, *concept-drift classification*, indicates that an algorithm may detect and adapt properly to different types of concept-drifts. Unfortunately, there is no algorithm that can satisfy this constraint. This is also one of our research work, which will be discussed in Chapter 4.

We also assess whether these classifiers can work for high-dimensional data streams. VFDT, CVFDT, AWE, OVA, and Ensemble-tree can work for high-dimensional data streams, as they deploy the decision tree as their primitive classifiers. When a classifier follows a computational approach and a time-window, it will have some specific tradeoffs, which is discussed in above Section 2.2. Moreover, data stream classifiers also inherit common advantages and limitations of traditional classifier in Table 2.2.

**Limitations of state-of-the-art work and thesis contributions:**

Although various data stream classifiers have been proposed, they have several limitations. Firstly, most data stream classifiers do not work well with high dimensional data, where all data objects appear to be sparse and dissimilar. Dimension reduction techniques, which extract relevant and non-redundant features, are necessary for accelerating the learning process. An overview of feature selection techniques will be presented in the next section. Secondly, none of above classifiers can properly adapt to different types of concept-drifts (indicated as concept-drift classification ability in Table 2.5.1). Furthermore, data stream algorithms are required to monitor feature evolution since the importance of a feature evolves over time. To address above challenges, we will introduce

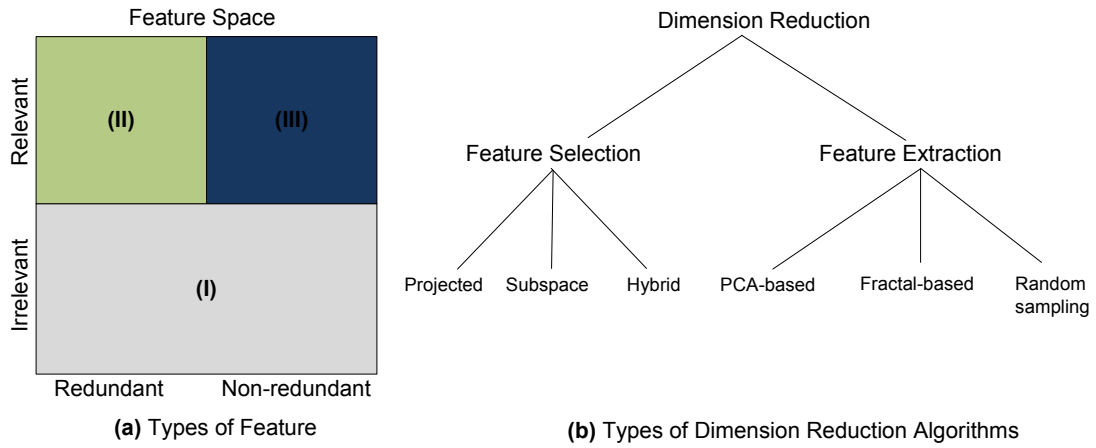



Figure 2.8: Overview of Feature Selection

a framework to incorporate feature selection into a heterogeneous ensemble in Chapter 4. Our algorithm not only work well with in high dimensional data streams, but also is able to detect and properly adapt to feature evolution.

In the next section, we continue to review other research on feature selection and semi-supervised learning, which are related to our research.

## 2.6 Other Related Research

### 2.6.1 Dynamic Feature Space

In high-dimensional dataset, not all data attributes (features) are important to the learning process. Figure 2.8(a) shows three common types of features: (i) irrelevant features, (ii) relevant but redundant features, and (iii) relevant and non-redundant features. The critical task of dimension reduction techniques is to extract the set of relevant and non-redundant features so that the learning process is more meaningful and faster. Moreover, to reduce the number of features, an algorithm may select a subset of the full feature space (called feature selection), or derive a new feature by combining existing features (called feature extraction). Feature selection is preferable to feature extraction, as it preserves the meaning of selected features while the latter usually creates hard-to-understand fea-

Table 2.4: Advantages and limitations of feature selection models.

| **Approach** | **Advantages** | **Limitations** |
|---|---|---|
| **Filter** | Fast<br>Scalable<br>Independce of the classifier | Ignores interaction with the classifier |
| **Wrapper** | Interact with the classifier<br>Can find the most "useful" features toward the classifier | High computational cost<br>Risk of over-fitting<br>Classifier dependent selection |
| **Hybrid**<br>(Filter+ Wrapper) | Interact with the classifier<br>Better computational complexity than wrapper methods<br>Are prone to over-fitting | Classifier dependent selection |

tures. Moreover, feature selection has a limited feature search space $|2^d|$, where $d$ is the number of features. Feature extraction has unlimited search space; however, its performance is usually better than feature selection. Figure 2.8(b) illustrates some approaches for each type of dimension reduction algorithms. For example, feature selection includes projected, subspace, and hybrid approaches. Feature extraction consists of PCA-based, fractal-based, and random sampling approaches [124].

In the literature, feature selection techniques can be classified into three categories: filter, wrapper and embedded models [125]. The filter model applies an independent measure to evaluate a feature subset; thus, it only relies on the general characteristics of data. The wrapper model runs together with a learning algorithm and uses its performance to evaluate a feature subset. A hybrid model takes advantage of the above two models. Advantages and limitations of the three models are elaborated in Table 2.4. Furthermore, the importance of a feature evolves in data streams and is restricted to a certain period of time. Features that are previously considered as informative may become irrelevant, and vice-versa, those rejected features may become important features in the future. Thus, *dynamic feature selection* techniques need to monitor the evolution of features.

After investigating many clustering and classification algorithms, we observe that only a few algorithms work with high-dimensional data streams, and, they have many limitations. For clustering, HPStream [29] applies the projected approach, and only removes irrelevant features but no redundant features. It requires a predefined value of the average dimensional degree, and is just suitable for spherical-shape cluster. Similarly, IncPreDeCon [126] only takes out irrelevant features and suffers from high complexity. For classification, some classifiers work with high-dimensional data streams as they are simply derived from or deployed decision trees as primitive classifiers [11, 12, 19, 22, 23]. In this scenario, they are considered to be deploying a filter model that uses the decision tree's informative measures (e.g., information gain) to select relevant features. Hence, these algorithms strictly use decision trees as primitive classifiers, and are not able to remove redundant features. Therefore, the problem of dynamic feature selection in data streams is open and requires further research. A dimension reduction technique that follows the hybrid model, captures features' evolution dynamically, and removes both irrelevant and redundant features will be ideal.

### 2.6.2 Semi-supervised Learning

Semi-supervised learning has been proposed to cope with partially labeled data. Although many semi-supervised learning algorithms have been developed for traditional data [127], there is still insufficient work for data streams. Semi-supervised learning can be categorized into two approaches [127]. The first approach is graph-based semi-supervised learning in which labeled and unlabeled samples are connected by a graph and edges represent similarity between samples [128, 129]. The goal is to estimate a smooth label function $f$ on the graph so that it is close to the given labels of vertices. However, it is difficult to apply the graph-based approach for data stream as the cost for managing and retrieving a large graph is high. The second approach, semi-supervised Support Vector

Machines (S3VMs) [130–132], incorporates unlabeled instances into learning by defining a hinge loss function that favors unlabeled instances outside of the margin and assigns them weights to alleviate the imbalance problem. However, this is a highly non-convex optimization problem which is difficult to solve.

Recently, Masud *et al.* proposed a semi-supervised algorithm for data streams, called SmSCluster [133]. The algorithm utilizes a cluster-impurity measurement to construct an ensemble of $k$-NN classifiers. When a classifier has no knowledge of a certain class, the algorithm copies the knowledge from another classifier with an injection procedure in order to reduce the mis-classification rate. However, SmSCluster suffers from high computational complexity as it performs clustering based on the $k$-means algorithm [134, 135]. The difficulty in choosing the optimal $k$ value is an inherent problem of $k$-means clustering and this is aggravated by the evolving nature of data streams which inevitably results in more dynamic clusters being formed. In 2007, Aggrawal *et al.* proposed a summarization paradigm for the clustering and classification of data streams [136]. This paradigm can be considered as a generalization of CluStream clustering [28] and On-Demand Classification [82]. Unfortunately, the classification and clustering algorithms run separately and there is no mutual relationship between them. The algorithm needs time to process the offline operations and cannot provide timely results. Moreover, the use of a pyramidal time window makes it unstable; the micro-structures become larger and larger over time and this degrades the model's performance [30].

**Limitations of state-of-the-art work and thesis contributions:**

Recent semi-supervised algorithms attempt to exploit clustering for their classification. However, as discussed above, they have several limitations, such as choosing a simple clustering method [133] and no mutual relationship [28]. There is a demand for exploring a new research problem, called *concurrent mining*, where the knowledge gained by one mining task may also be useful to other mining tasks and vice versus. Furthermore,

in many real applications, labeled training instance are rare since it is time-consuming and requires domain experts to label instances. Unfortunately, most semi-supervised classifiers do not work well with very sparsely labeled datasets. Active learning is able to improve semi-supervised learning by querying a small number of unlabeled instances for labels. In Chapter 5, we address these limitations by proposing a concurrent algorithm that performs clustering and classification at the same time. Our algorithm is further equipped with a novel active learning technique so that it can work well with very sparsely labeled datasets.

## 2.7 Summary

In this chapter, we have examined classification problem of the two most popular type of sequential data: streaming time series and data streams. We have analyzed various algorithms of tradition classification, time series classification, data streams classification, feature selection, and semi-supervised classification. By figuring out limitations of the state-of-the-art classifiers, we have risen several research problems, including efficient classification for streaming time series, dynamic feature space problem, and concurrent mining problem for data streams. In the next chapters, we will elaborate more these research problems and propose algorithms to solve them.

*"Data is not information, information is not knowledge, knowledge is not understanding, understanding is not wisdom."*

Clifford Stoll Astronomer

*"Success consists of going from failure to failure without loss of enthusiasm."*

Winston Churchill (1874-1965) British prime minister

# 3

# Closed Motifs for Streaming Time Series Classification

## 3.1 Introduction

With the current rapid development of information technology, large amounts of data are being collected daily from both scientific and business applications, such as electrocardiograms in biomedicine, sensor data of aircraft and stock market movements. Such data that are continually observed results in massive amounts of chronological data, termed streaming time series [137]. To discover knowledge from streaming time series, scientists are challenged by its characteristics: unbounded size, high dimensionality and dynamic nature. Motif discovery, which aims to find frequent patterns or subsequences (called *motifs*), is one of the most important time series mining tasks. It is an essential preprocessing procedure which is widely used for other mining tasks. For example, motif discovery is the first step to extract association rules for time series data. They are referred to by different names, such as "primitive shapes" [138], "frequent patterns" [139],

"chords" [140], and "shapelets" [2]. Motifs are considered as representatives for each class in several time series classification algorithms [2, 141]. They are also used for time series clustering and summarization [142] and anomaly detection [143]. Moreover, motif discovery has been successfully applied in many domains, such as biology [109, 144], motion capture [113, 145].

Since the hidden patterns are previously unknown, discovering motifs with different lengths is a tedious task. To reduce the computational requirements, most existing work limits the search space with the assumption that the length of motifs $w$ must be predefined. Many effective algorithms have been proposed to find $w$-length motifs [40–44]. However, these algorithms are very sensitive to the choice of $w$. If the value of $w$ is too small, only subsequences of the hidden patterns can be found and a large number of redundant motifs will be produced; if the value of $w$ is too large, patterns may not be found because patterns following by different adjacency subsequences (tails) are considered unmatched [10]. Even if the value of $w$ is properly configured, only motifs with length $w$ can be discovered. For example, an analyst wants to discover motifs having the number of occurrence greater than 2 for electrocardiogram dataset in Figure 3.1. The value of $w$ is small resulting in many redundant motifs found in Figure 3.1(a). On the other hand, there is no discovered motif in the case of a large $w$ in Figure 3.1(b).

Although there are some efforts to find motifs of different lengths, there are several limitations. Nunthanid *et al.* proposed running the $k$-motif algorithm several times for different values of $w$ which is inefficient [146]. Tang *et al.* recorded a similarity matrix for all subsequences over time. Scanning the whole similarity matrix, they search for trends in the matrix that point to the existence of longer motifs [10]. However, this method requires much memory and many passes through the time series which makes it inapplicable to streaming time series where data streams can only be read once.

The above observations motivate us to find *closed motifs* having appropriate lengths more efficiently, for example Figure 3.1(c). We term a motif to be *closed*, if there is no

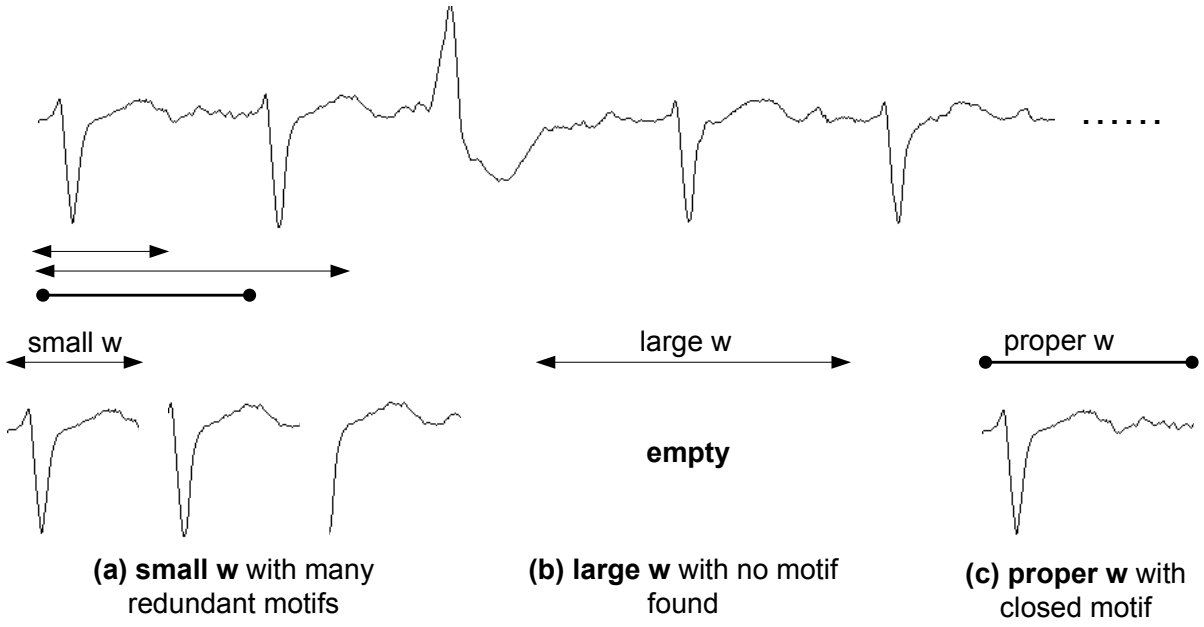



Figure 3.1: Example of motif discovery for an electrocardiogram dataset with different values of $w$.

motif with longer length having the same number of occurrences. It is akin to closed itemsets [147] in association rule mining which are shown to have practical use in many applications, such as market basket analysis [148], web usage mining [149], and software bug mining [150].

Time series classification has attracted a lot of research interest in the last decade. Although many algorithms have been developed, the nearest neighbor classifier has been shown to be one of the most robust time series classification techniques [6–8]. While the nearest classifier is simple and easy to implement, its time and space complexities are relatively high. Ye *et al.* defined a novel time series shapelet and proposed a decision tree based classifier for time series classification [2]. The algorithm is later extended to Logical-Shapelet which augments shapelet representation with logical operations and uses caching techniques for better performance [9]. These approaches have been experimentally demonstrated to be more accurate and interpretable than the nearest neighbor approaches. However, the shapelet algorithms require a long training time and can only

run in an off-line manner, where all training data must be obtained in advance.

Here is a summary of our contributions to address the challenges of finding motifs of different lengths and classifying streaming time series:

- We introduce a new concept named *closed motifs* and propose a novel algorithm to discover closed motifs.
- We index a streaming time series by constructing and updating a flexible suffix tree. Using the downward closure property, we traverse the tree in a depth-first manner to discover closed motifs.
- We develop a user-friendly toolkit for visualizing closed motifs named ARC-VIEW. ARC-VIEW is more practical than the VizTree toolkit [43] in visualizing large suffix trees with a large alphabet size. We further enhanced it with an advanced ranking method to discover the most distinctive motifs.
- We further propose a nearest neighbor classifier that is based on closed motifs. Although the algorithm only needs to store a small number of distinctive closed motifs, it achieves greater accuracy than the state-of-the-art time series classifier Logical-Shapelet [9].

## 3.2 Preliminaries

In this section, we introduce notations and relevant definitions, and then propose a novel algorithm to discover closed motifs in time series.

**Definition 3** ***A time series*** $T = (t_1, t_2, \ldots, t_n)$ *is a finite, real-valued sequence of a variable's observations, where* $n$ *is the length of* $T$*.*

Due to high computational needs, time series mining algorithms usually work on a subsequence or a subsection of the original time series [3, 41, 43, 97].

**Definition 4** *Given a time series $T$ of length $n$, a* ***subsequence*** $S = t_i, \ldots, t_{i+m-1}$ *is a sampling of $m \leq n$ contiguous positions of $T$, such that $1 \leq i \leq n - m + 1$ ( definition from [40]).*

In many real applications, a time series dataset can be collected in a streaming manner; e.g. data generated by a sensor network, or data collected from the stock market. In this scenario, it is called streaming time series [137].

**Definition 5** *A* ***streaming time series*** *$T$ is an infinite time series, where observations may arrive at arbitrary intervals.*

### 3.2.1 SAX representation

The SAX [3] (Symbolic Aggregate approXimation) representation symbolizes a time series $T$ segregated into a sequence of $w$ symbols (a word) w.r.t. an alphabet size $a$. A SAX representation is written as a tuple $SAX(T, w, a)$. The SAX algorithm has three main steps:

(i) First, the time series $T$ of length $n$ is normalized and then divided into $w$ consecutive segments, each of which has an equal length of $\lceil \frac{n}{w} \rceil$. Piecewise Aggregate Approximation (PAA) of $T$ is created with average values of these segments.

(ii) Given the alphabet size $a$, a list of $(a - 1)$ breakpoints $B = \{\beta_1, \beta_2, \beta_3, \ldots \beta_{a-1}\}$ is calculated so that each area from $\beta_i$ to $\beta_j$ under the Gaussian curve equals $1/a$. These breakpoints can be computed by using simple statistical calculations. Then, each interval $[\beta_i, \beta_j)$ is mapped to a symbol.

(iii) Finally, the PAA coefficients are converted to symbols according to which breakpoint interval they belong to. Figure 3.2 shows examples of the SAX algorithm.

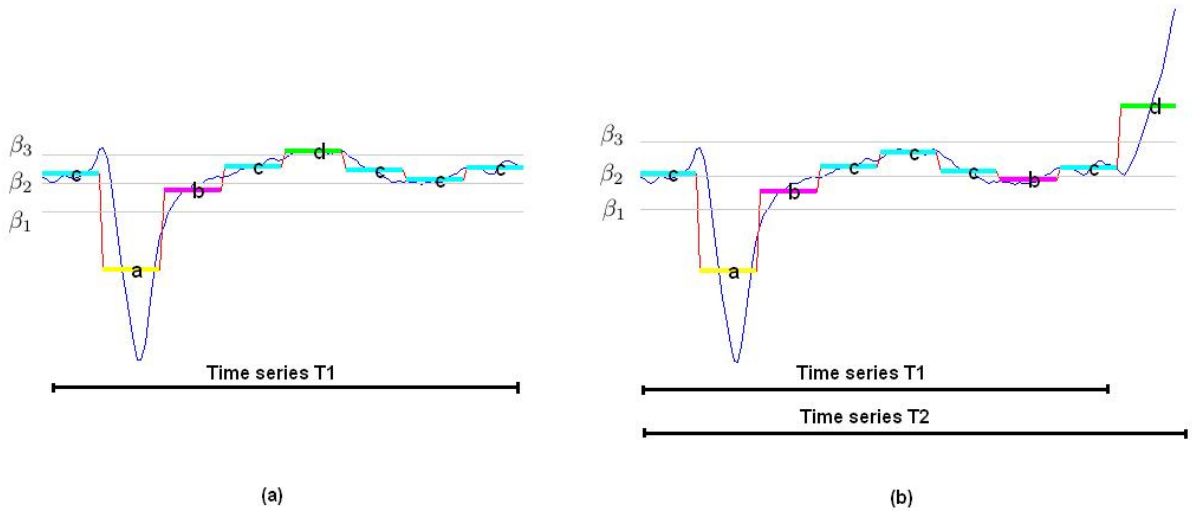



Figure 3.2: (a) SAX representation of time series $T_1$ with $n = 128$, $w = 8$ and $a$ =4. (b) SAX representation of time series $T_2$ with $n = 144$, $w = 9$ and $a$ =4. Images are generated by MATLAB with source code provided by SAX authors [3].

When the length of a time series $n$ is large, a sliding window of length $w$ can be used to extract all possible subsequences of length $w$. Then, each subsequence is mapped onto a word using the SAX algorithm. The support of a SAX word is defined as the count of its *non-trivial matches* [3] that are not nearby and have the same SAX representation. Figure 3.3 shows examples of trivial and non-trivial matches.

**Definition 6** *Given two subsequences $S_1$ and $S_2$ of a time series $T$ having the same SAX word, $S_2$ is considered as a* ***non-trivial match*** *to $S_1$ if there exist a subsequence $M$ between $S_1$ and $S_2$ so that $M$ has a different SAX word.*

**Definition 7** *A subsequence $S$ is frequent if the support of its SAX word is equal or greater than a minimum occurrence threshold $\sigma$, $occurrence(S) \geq \sigma$. We shall term such a subsequence $S$ a* ***motif****.*

We define a *downward closure* property as follows that is similar to that of frequent pattern mining:

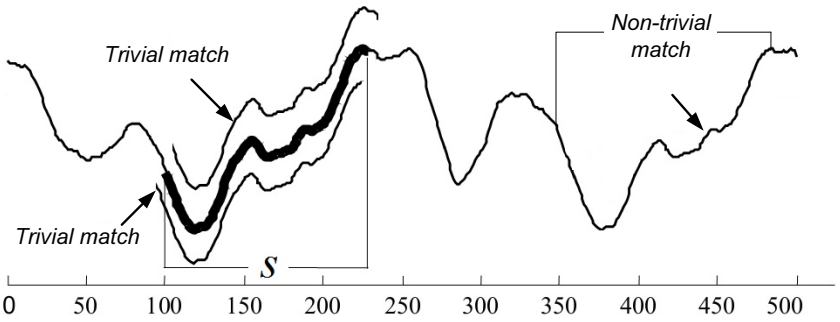



Figure 3.3: Trivial matches are immediately to the left and right of a subsequence $S$, while non-trivial match is apart from $S$.

**Theorem 1** *Given a subsequence $S$, the support of any subsequence of $S$ is equal or greater than the support of $S$.*

Theorem 1 can be proved easily, as every time $S$ appears, its subsequence also appears. Moreover, we can derive:

**Corollary 1** *If a subsequence $S$ is a motif, all subsequences of $S$ are also motifs.*

**Definition 8** *When $S'$ is a subsequence of $S$, we say that $S$ is a **parent-sequence** of $S'$.*

**Definition 9** *A motif $S'$ is a **closed motif** if it has no parent-sequence $S$ that has the same number of occurrences.*

As mentioned above, motif discovery results can be overwhelmed with a lot of redundant motifs. Hence, we are only interested in finding *closed* motifs.

Although SAX is a simple and effective motif discovery algorithm, it has a limitation that SAX representations of two subsequences may be totally different in the event that one subsequence is a parent-sequence of another. This makes it difficult to check whether a motif is closed because we need to make sure that it has a higher occurrence than all of its parent-sequences. For example, Figure 3.2 shows the SAX representations of two

subsequences $T_1$ and $T_2$. Although $T_2$ is a parent-sequence of $T_1$; their SAX words are different, (**cabcdccc**)$\neq$ (**cabcccbc**)d. To check whether $T_1$ is a closed motif, we need to know $T_2$ and its support. Since their SAX words are different, it is hard to find exactly a SAX word of $T_2$ as well as its support.

## 3.3 Our algorithm

In the previous section, we highlighted the limitations of SAX in situations where two subsequences may have different data distributions because of having different breakpoints. Here, we present our approach to overcome this limitation by modifying SAX with global breakpoints. First, we read a sufficient amount of the time series data, and calculate its mean and standard deviation. These values are then used to compute the global breakpoints to symbolize time series. When a new data instance comes, the mean, standard deviation and global breakpoints are updated accordingly. Since these global breakpoints are sensitive to outliers, it is advisable to perform data pre-processing to remove outliers with extreme values. Our algorithm is able to handle data streams if the global breakpoints are reasonably stable across the life of the data stream. This assumption makes the SAX representations of a subsequence and its extending subsequence consistent; therefore, it limits the search space and reduces the complexity of the closed motif discovery problem.

### 3.3.1 Suffix Tree Construction

Given a streaming time series $X$, we construct a dynamic tree structure $closedTree$ to index all subsequences with different lengths. In $closedTree$, each tree node represents a specified subsequence and stores its weight as well. Details of constructing $closedTree$ are given in Algorithm 1. Using our modified version of SAX, we first symbolize $X$ into a streaming string $\widehat{\mathbf{X}} = [\widehat{s_1}, \widehat{s_2}, \ldots, \widehat{s_m}, \ldots]$ by using the global breakpoints. We initialize

**Algorithm 1** Construct *closedTree*

**Input:** A streaming time series $\mathbf{X} = [\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_n}, \ldots]$.
The minimum length of motifs, $l_{min}$, the minimum occurrence threshold $\sigma$.
**Output:** An indexing tree structure *closedTree*.

```
X is symbolized to a streaming string X̂ = [ŝ_1, ŝ_2, ..., ŝ_m, ...]
Initialize two words w = ŝ_1 ŝ_2 ... ŝ_{l_min} and w_tail = ŝ_2 ... ŝ_{l_min}
while X has more symbols ŝ_m do
    if a node of the word (w, s_m) exists then
        Concatenate s_m to w and w_tail
    else
        Update weights of all nodes in the paths from the tree's root to Equation 3.2.
        if w & (w_tail, s_m) are frequent then
            Create a new node for (w, s_m), and set its weight according to Equation 3.1.
        end if
        Remove the first symbol of w and w_tail
        Remove trivial matches of w
    end if
end while
```

a word $w$ with a minimum length $l_{min}$ and extract its tail word $w_{tail}$ by excluding the first symbol $\widehat{s_1}$. The word $w$ can be considered as a sliding window with a flexible length. The word $w$ is extended as long as possible based on its potential to become a motif. Given an upcoming symbol $s_m$ of the streaming time series $X$, if a node for the corresponding word $(w, s_m)$ exists (that means the word $(w, s_m)$ has been already indexed), we concatenate $s_m$ to $w$ and $w_{tail}$. Otherwise, we traverse the path from the tree's root to $w$ and update weights of all nodes according to Equation 3.2. Based on the Corollary 1, we derive a necessary condition to index the word $(w, s_m)$ that is when all of its subsequences are motifs. Since all subsequences of $(w, s_m)$ are exactly $w$, $(w_{tail}, s_m)$ and all their subsequences, we only need to check whether two words $w$ and $(w_{tail}, s_m)$ are frequent. If they are frequent, we create a new node for the word $(w, s_m)$, and use a probabilistic model to estimate its weight.

For simplicity, we rewrite the word $(w, s_m) = (s_1, s_2, \ldots, s_{k-1}, s_k)$. Moreover, we can easily derive $w = (s_1, s_2, \ldots, s_{k-1})$, $(w_{tail}, s_m) = (s_2, \ldots, s_{k-1}, s_k)$. We denote

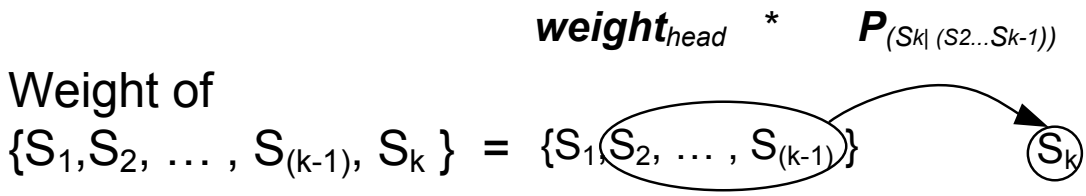



Figure 3.4: Illustration of the probabilistic model to estimate weights of concatenating words.

weight of a node by its subscript; for example, $weight_{(s_1,s_2,\ldots,s_{k-1})}$ is weight of the node $(s_1, s_2, \ldots, s_{k-1})$. The weight of the word $(w, s_m)$ is approximated as:

$$weight_{(s_1,\ldots,s_k)} \approx weight_{(s_1,\ldots,s_{k-1})} * \frac{weight_{(s_2,\ldots,s_k)}}{weight_{(s_2,\ldots,s_{k-1})}} \tag{3.1}$$

Next, we take out the first symbol of the words $w$ and $w_{tail}$, remove trivial matches of the word $w$ and continue in the next loop.

Figure 3.4 illustrates the probabilistic model to estimate the weight of the word $(w, s_m)$. Its proofs are as follows:

**Proof:** We use Bayes' theorem, $P(A, B) = P(A|B)P(B)$, as a basis to derive this estimation. Suppose $A = s_k$ and $B = (s_1, \ldots, s_{k-1})$, we have:

$$P(s_1, \ldots, s_k) = P(s_k|(s_1, \ldots, s_{k-1})) * P(s_1, \ldots, s_{k-1})$$

We estimate $P(s_k|(s_1, \ldots, s_{k-1}))$ to $P(s_k|(s_2, \ldots, s_{k-1}))$. Then,

$$\begin{aligned} P(s_1, \ldots, s_k) &\approx P(s_k|(s_2, \ldots, s_{k-1})) * P(s_1, \ldots, s_{k-1}) \\ &\approx \frac{P(s_2, \ldots, s_k)}{P(s_2, \ldots, s_{k-1})} * P(s_1, \ldots, s_{k-1}) \end{aligned}$$

Since $P(s_1, \ldots, s_k) = \frac{weight_{(s_1,\ldots,s_k)}}{N}$, with N is a current length of the time series. We

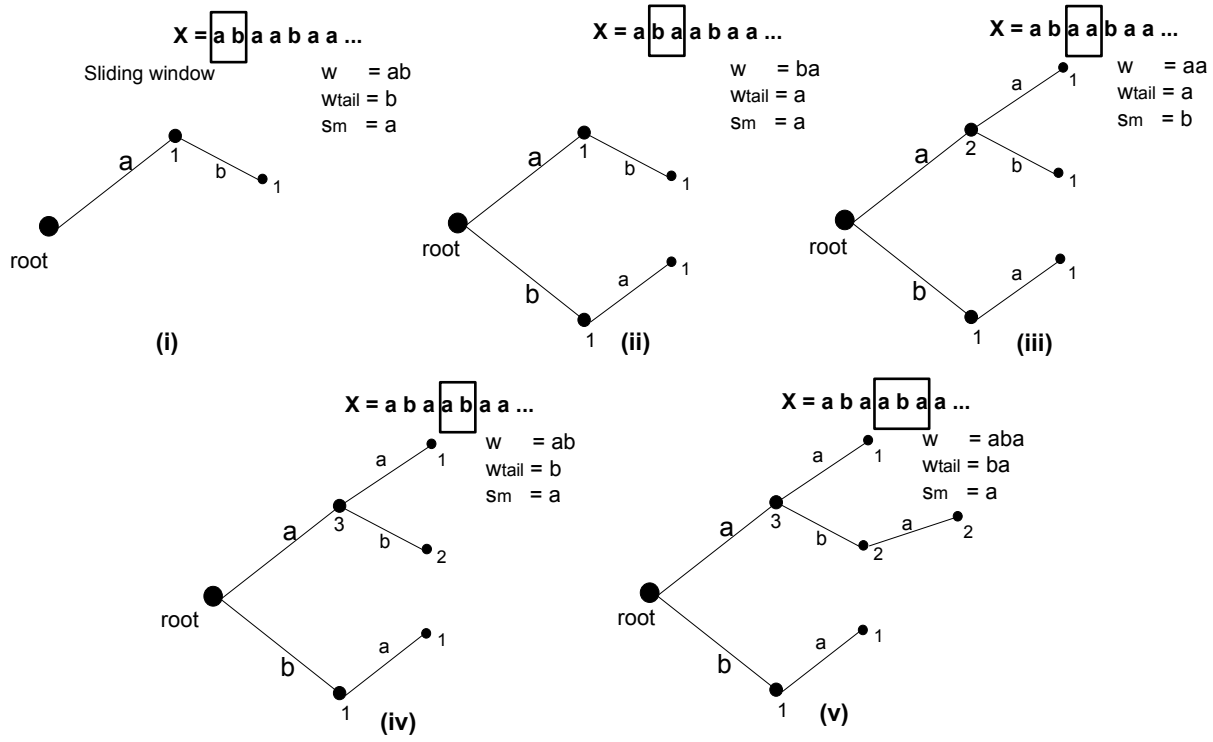



Figure 3.5: Examples of constructing *closedTree* with an alphabet size $a$ =2 and occurrence threshold $\alpha$ =2.

have:

$$\frac{weight_{(s_1,\ldots,s_k)}}{N} \approx \frac{weight_{(s_2,\ldots,s_k)}/N}{weight_{(s_2,\ldots,s_{k-1})}/N} * \frac{weight_{(s_1,\ldots,s_{k-1})}}{N}$$

$$\Leftrightarrow weight_{(s_1,\ldots,s_k)} \approx \frac{weight_{(s_2,\ldots,s_k)}}{weight_{(s_2,\ldots,s_{k-1})}} * weight_{(s_1,\ldots,s_{k-1})}$$

Figure 3.5 shows examples of constructing the *closedTree*. The top of Figure 3.5 illustrates a streaming time series $X$ arriving continuously. The time series $X$ is symbolized into many symbols by SAX with an alphabet of size 2. Each tree node represents a word constructed by following a path from the tree's root to this node. The number below a

node denotes its weight. At the initialization stage in Figure 3.5(i), the sliding window is **ab** and its according tree node is created with weight 1. Then, the window moves to the right, we have $w = ba$, $w_{tail} = a$, and $s_m = a$. Since the node $(w, s_m) = baa$ does not exist, we create a new node **ba** with weight 1 in Figure 3.5(ii). Similarly, a node **aa** is created, and weights of all nodes in the path from the root to this node are incremented by 1 in Figure 3.5(iii). And, we have $w = ab$, $w_{tail} = b$, and $s_m = a$ in the next window. The weight of node **ab** is incremented by 1. Since the nodes $w = ab$ and $(w_{tail}, s_m) = ba$ are frequent, a new node $(w, s_m) =$ **aba** is created. We need to also estimate its weight based on Equation 3.1.

$$weight_{aba} = weight_{ab} * \frac{weight_{ba}}{weight_b} = 2 * \frac{1}{1} = 2.$$

It is noteworthy that although the window length is 2, the $closedMotif$ algorithm can discover motifs with longer lengths. For example, the motif $aba$ with length 3 is found in this example.

#### 3.3.1.1 Fading Model

Motif discovery algorithms for streaming time series need to update their results continuously. Moreover, the importance of motifs may change over time; motifs that are important in the past may become uninteresting in the present. For example, stock market motifs that are discovered during an economic crisis will become obsolete when the economy gradually recovers. Therefore, we apply a fading model to capture the dynamic nature of motifs.

For each observation $t$, we assign a weight, which decreases according to its age. The age is defined as the time gap between the current time and the arriving time. Particularly, the weight of $t$ is set according to a decreasing exponential function, $weight(t) = \lambda^{\Delta t}$, where $(0 < \lambda \leq 1)$ is a fading parameter and $\Delta t$ is the age of $t$.

Similarly, we update the weight (importance) of a motif $S$ as follows:

$$weight(S) = weight(S) * \lambda^{\Delta t} + 1, \tag{3.2}$$

where $\Delta t$ is the time gap between the current time and the last occurrence time of the motif. The line 7 in Algorithm 1 shows how we apply the fading model while constructing the *closedTree*.

**Theorem 2** *Given a fading parameter $\lambda$, the total weight of a time series $T$ of length $n$ is* $\frac{1-\lambda^{(n+1)}}{1-\lambda}$.

**Proof:** Given a time series $T = t_1, t_2, \ldots, t_n$, according to the fading model, an observation $t_i$ has a weight of $weight(t) = \lambda^{\Delta t} = \lambda^{(n-i)}$. Hence, the total weight of $T$ is:

$$\begin{aligned} weight(T) &= \sum_{i=0}^{n} weight(t_i) = \sum_{i=0}^{n} \lambda^{(n-i)} \\ &= \sum_{i=0}^{n} \lambda^{i} = \frac{1-\lambda^{(n+1)}}{1-\lambda} \end{aligned}$$

The total weight of a time series is subsequently used to calculate the *coverage* of a motif.

### 3.3.2 Closed motifs discovery

We traverse the *closedTree* to collect all closed motifs in a depth-first manner. Starting from the root of *closedTree*, we recursively call Algorithm 2. Firstly, an empty list of closed motifs *return_list* is initialized. The node $v$ is an input of the algorithm. If the weight of $v$ is less than an occurrence threshold value $\sigma$; then $v$ is not a motif, and an empty list is returned. To determine whether $v$ is a closed motif, we need to check that $v$ has no parent-sequence that has the same weight. The parent-sequences can be extended from both sides of $v$. For the right side, we check for all child nodes $e$ of $v$. If there is any child node $e$ having the same weight of $v$, the node $v$ is not a closed motif. For the left side, we create a parent-sequence by appending each character to the SAX word of the node $v$. If the node of the parent-sequence exists and has the same weight

**Algorithm 2** $closedMotif$(Node $v$)

**Input:** An indexing tree structure $closedTree$ and occurrence threshold value $\sigma$
**Output:** A list of closed motifs.

```
1: Initialize return_list = {}
2: if v.weight < σ then
3:    Return an empty list
4: end if
5: vIsClosed = true
6: for all child nodes e of v do
7:    if e.weight == v.weight then
8:       vIsClosed = false
9:    end if
10: end for
11: for all character c in the alphabet do
12:    w = c + v.getWord()
13:    Node e = searchNode(w)
14:    if e.weight == v.weight then
15:       vIsClosed = false
16:    end if
17: end for
18: if vIsClosed then
19:    Add v to return_list
20: end if
21: for all child nodes e of v do
22:    Recursively call closedMotif(e)
23:    Append its result to return_list
24: end for
```

of $v$, the node $v$ is not a closed motif. Otherwise, $v$ is a closed motif and is added to $return\_list$. Finally, we call Algorithm 2 for each child node $v$ recursively, and append its result to $return\_list$.

### 3.3.3 ARC-VIEW: a visualization toolkit for closed motifs

For visualization of closed motifs in streaming time series, we created a toolkit called *ARC-VIEW*, which is partly inspired by a visualization tool for time series named VizTree [42]. The VizTree draws a suffix tree where each branch has a specified SAX word that is created by traversing along the branch. Each branch, together with a weight,

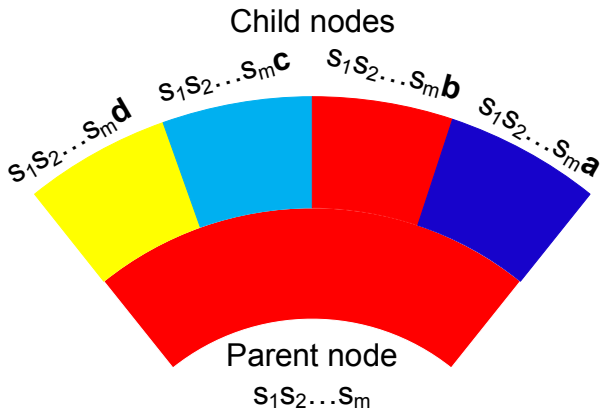



Figure 3.6: An example of ARC-VIEW with alphabet {a,b,c,d}. A parent node is represented by an arc, and its child nodes are depicted by sub-arcs filled with corresponding colors {blue, red, cyan, and yellow}.

represents motifs having the same SAX word. Although VizTree can discover motifs and detect anomalies, it has some limitations. With a large alphabet size, VizTree creates visually complicated figures that are very hard to interpret. Moreover, it can only show motifs with a predefined length. That makes SAX impossible to be used with streaming time series, where motifs have different lengths and need to be updated continuously.

ARC-VIEW not only overcomes the above limitations, but also allows users to monitor closed motifs in real time. Furthermore, ARC-VIEW provides a user-friendly interface that helps users to easily identify and view discovered motifs in an intuitive way. In ARC-VIEW, each node of the *closedTree* is represented by an arc with a specified angle, width and center. If the node has child nodes, the arc is further extended to many equal-angled sub-arcs, which have the same width and center. Moreover, these sub-arcs are represented by different colors. Figure 3.6 illustrates an example of ARC-VIEW with an alphabet size of 4. The parent node $s_1s_2\ldots s_m$ has four child nodes. Then, its child nodes are visualized by sub-arcs filled with corresponding colors. In this example, the alphabet $\{a,b,c,d\}$ is coded by colors $\{blue,\ red,\ cyan,\ and\ yellow\}$.

Figure 3.7 shows the screenshot of the ARC-VIEW toolkit. It has a main frame, ranking frame and classification frame, which are numbered as 1, 2 and 3 respectively.

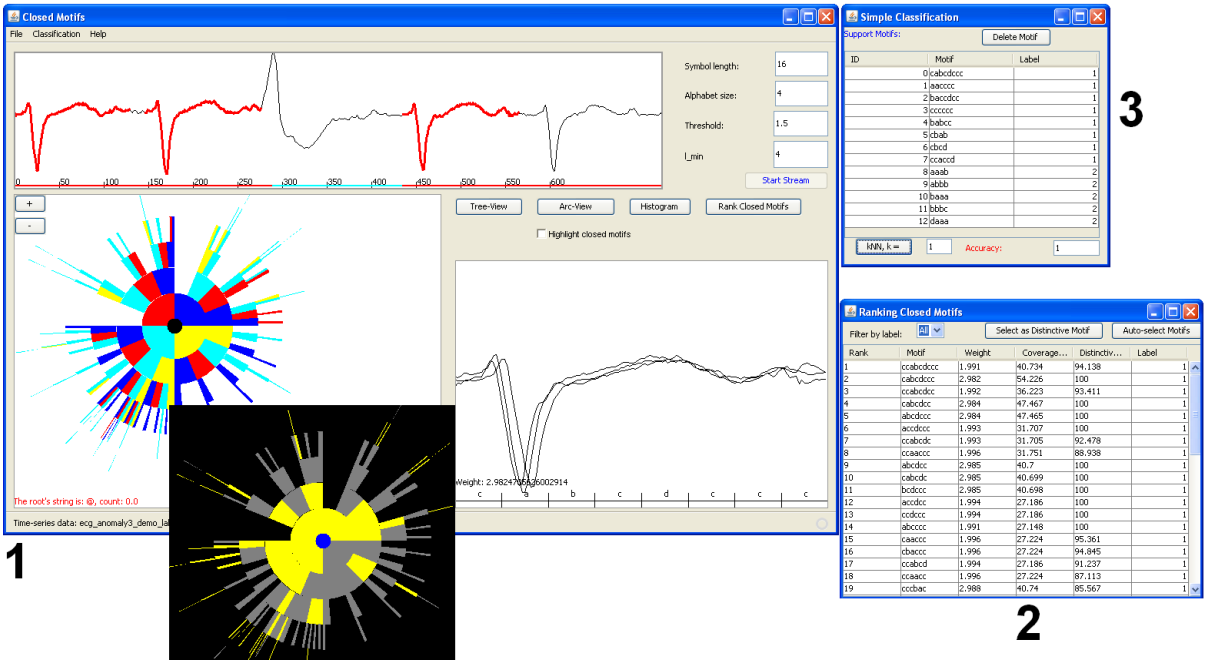



Figure 3.7: A screenshot of ARC-VIEW with Trace dataset [4]. ARC-VIEW toolkit has three frames, including a main frame, ranking frame and classification frame, which are numbered 1, 2 and 3 respectively.

When the program is executed, the main frame is loaded with three blank panels. Users can change the default parameter settings at the top-right corner. The parameters include symbol length, alphabet size, occurrence threshold value $\sigma$ and $l_{min}$ value. When users select an input time series by using a dropdown menu, the top-left panel of the main frame shows a preview of the time series. Then, users can click on buttons to view the *closedTree* in two modes: tree-view or arc-view. The bottom-left panel visualizes the *closedTree* in the arc-view mode. In this panel, users can zoom in, zoom out and drag to obtain a better view. When users click on a specified arc, motifs belonging to this arc will appear at the bottom-right panel. This panel also shows the weights of those motifs and their SAX words.

As closed motifs are important, ARC-VIEW offers two convenient ways to view closed motifs. Firstly, users can select the check-box to highlight closed motifs. With a black

background, those closed motifs are highlighted in yellow, and non-closed motifs are distinguished easily with brown color. Figure 3.7 demonstrates how closed motifs are highlighted when the check-box is selected. Secondly, ARC-VIEW ranks a list of closed motifs by their distinctive power, which is related to its purity and coverage as defined below.

Coverage of a motif is calculated as a multiple of its length and frequency divided by the total weight of the time series, which is defined in Theorem 2. It indicates how often these closed motifs appear in the whole time series. In the classification problem, time series data comes with many labels; therefore, a motif may cover observations from many classes. The purity of a motif is the largest percentage of the dominant class covered by the motif. The **distinctive power** of a motif is first prioritized by its purity; motifs with higher purity are more distinctive. If two motifs have the same purity, then we rank them according to their coverage, i.e., the more distinctive motif has larger coverage.

The ranking frame, numbered 2 in Figure 3.7, shows an example of ranking by distinctive power. The first motif *ccabcdccc* has purity of 94.138% as it covers observations from the classes 1 and 2. The fading parameter $\lambda$ is 0.99998, and the current view of the time series consists of 720 observations. Hence, the motif's coverage is calculated as follows:

$$\begin{aligned} Coverage(aaadeeeee) &= \frac{length \times weight}{totalweight} \times 100\% \\ &= \frac{9 \times 16 \times 1.9915}{\frac{1-0.99998^{(720+1)}}{1-0.99998}} \times 100\% \\ &\approx 40.734\%. \end{aligned}$$

Furthermore, in the classification frame, numbered 3 in Figure 3.7, we implemented a simple classification algorithm to demonstrate the capability of distinctive power in a classification problem, which is explained in the next section.

ARC-VIEW also supports the processing of streaming time series. When data from a streaming time series arrives continuously at a large volume, the *closedTree* is updated according to Algorithm 1. In this scenario, the top-left panel of the main frame only shows the most recent segment of the streaming time series. When users click on "Start Stream" button, new data is processed and the recent segment and *closedTree* are updated consequently. The arc-view of the *closedTree* and the table of closed motifs are also updated accordingly. This utility helps users to quickly identify whether any new closed motif appears.

### 3.3.4 Closed Motifs for Classification

In the previous section, the ARC-VIEW toolkit has demonstrated its capability in visualizing time series data and finding closed motifs. We believe that closed motifs can be successfully applied to many time series data mining problems, including clustering, rule discovery and anomaly detection. However, we will focus only on the classification problem.

We propose the nearest neighbor (1-NN) classifier that utilizes the set of discovered closed motifs to improve speed and accuracy. Recent research has shown that the 1-NN algorithm is effective and very difficult to match in most time series problems [6–8]. However, 1-NN algorithm cannot manage large-scale classification problems as it requires much computational and memory resources. To overcome these limitations, we aim to extract a short list of the most distinctive closed motifs for each class. Our algorithm only needs to compare a testing time series to this list; therefore, it can quickly solve a classification problem with accurate results. First, we need to define a similarity measure between two time series.

**Definition 10** *Distance between two time series of the same length,* $T = (t_1, t_2, \ldots, t_n)$ *and* $P = (p_1, p_2, \ldots, p_n)$*, is a symmetrical function and returns a nonnegative value, i.e.,* $Dist(T, P) = Dist(P, T) \geq 0$.

For example, Euclidean distance can be used to measure the distance between two time series, $ED(T,P) = \sqrt{\sum_{i=1}^{n}(t_i - p_i)^2}$.

Since a subsequence $S$ is usually shorter than the time series $T$, we need to define the distance between a time series $T$ and a subsequence $S$.

**Definition 11** *Subsequence distance from the time series* $T = (t_1, t_2, \ldots, t_n)$ *and the subsequence* $S = (s_1, s_2, \ldots, s_l)$ *is the minimum distance between a subsequence* $S'$ *of the time series* $T$ *and the subsequence* $S$, $SubDist(T,S) = min(Dist(S,S')), S' \subset T$ *(definition from [2]).*

The subsequence distance is actually the distance between $S$ and its best-matched subsequence $S'$ of $T$, as shown in Figure 3.8.

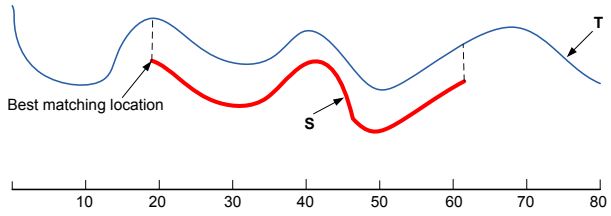



Figure 3.8: Illustration for subsequence distance.

There are some research work that use a distance measure MINDIST [43] to implement the $k$ nearest neighbor. However, the MINDIST is the lower bound of the distance between two original time series which may contain some errors. Therefore, this approach is not good enough to produce accurate results. To overcome this limitation, we define a mean subsequence of a SAX word as follows:

**Definition 12** *Given a SAX word having* $m$ *non-trivial matches of a length* $l$, $S^k = \{s_1^k, s_2^k, \ldots, s_l^k\}$ $(1 \leq k \leq m)$, *its mean subsequence* $\bar{S} = \{\bar{s}_1, \bar{s}_2, \ldots, \bar{s}_l\}$ *is defined:*

$$\bar{s}_i = \frac{1}{m}\sum_{k=1}^{m} s_i^k.$$

We integrate each SAX word with a mean subsequence that can be used later to measure the distance between itself and a testing time series. Figure 3.9 illustrates a node in the *closedTree* with its mean subsequence. The node *cabc* with weight 3.98 consists of a mean subsequence (indicated by a thick line) of four subsequences belonging to it. When a tree node $N$ receives a new subsequence $S_{new}$, its mean subsequence $\bar{S}$ can be easily updated as follows:

$$\bar{S} = \frac{\bar{S} * weight(N) + S_{new}}{weight(N) + 1}$$

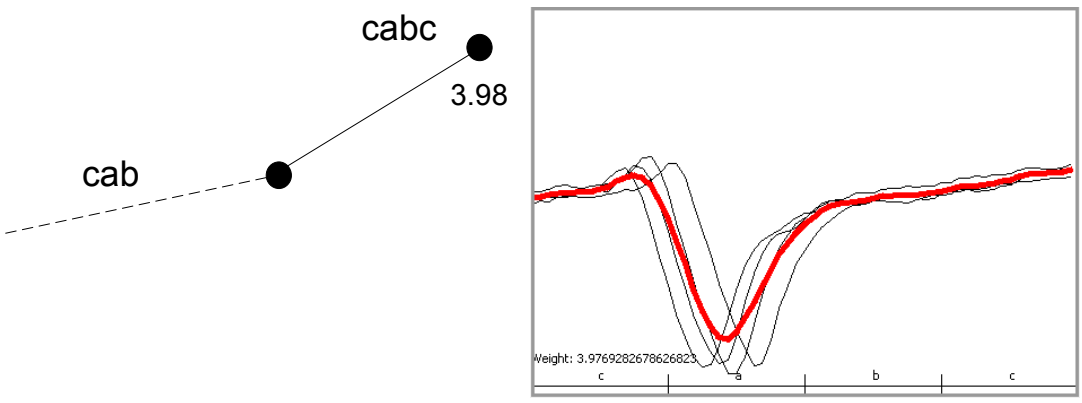



Figure 3.9: Illustration for a mean subsequence of a tree node with a thick line.

Details of the classifier, 1-NN *closedMotif*, are given in Algorithm 3. During the learning procedure, the suffix tree is constructed, and a list of closed motifs is extracted. The 1-NN *closedMotif* gets this list as input. Since the number of closed motifs is still large, we are only interested in the list of the top $m$ distinctive motifs for each class (line 1). For each motif in the list, we compute the subsequence distance between its mean subsequence and a testing time series $T$. Then, we select a motif with the shortest distance and assign its label to the time series $T$.

**Algorithm 3** 1-NN $closedMotif$ (Time series $T$, $m$)

**Input:** A list of closed motifs, $list$
Maximum number of motifs for each class, $m$
**Output:** Label of time series $T$

1: Select top $m$ distinctive closed motifs for each class and put into a list $short\_list$
2: **for all** closed motif in $short\_list$ **do**
3: Get its mean subsequence $meanSub$.
4: Compute subsequence distance between $T$ and $meanSub$, $SubDist(T, meanSub)$.
5: **end for**
6: Select a motif with the shortest distance and assign its label to time series $T$.

## 3.4 Experiments and Analysis

### 3.4.1 Experimental Setup

To validate the effectiveness of our $closedMotif$ algorithm, we perform extensive experiments on both synthetic and real time series datasets. The datasets are taken from the publicly available UCR time series Data Mining Archive [4] and PhysioBank [151]. Experiments were conducted on a Windows PC with a Pentium D 3GHz Intel processor and 2GB memory. This section consists of two parts:

- For motif discovery, we compare our approach with the SAX algorithm as this is one of the most cited motif discovery approaches (more than 200 citations up to date).

- For time series classification, we compare our algorithm with *Logical-Shapelet*, a recent prominent classifier for time time series [9]. According to the author's recommendation, default values of the Logical-Shapelet are set as follows: $minLength = 10$ and $stepSize = 1$.

The default parameter settings for our algorithm are set as follows: alphabet size $a$ is 15, occurrence threshold value $\sigma$ is 1.5, the maximum number of motifs for each class $m$ is 30, and initial window is 64 with 4 symbols of length 16. The fading parameter $\lambda$ in our algorithm is 0.99998.

### 3.4.2 Motif Discovery

We examine *closedMotif* and SAX in many aspects including discovering complete and varying-length motifs, and their scalability. To allow fair and meaningful comparisons, common parameters of *closedMotif* and SAX are given the same values.

#### 3.4.2.1 Complete motif discovery:

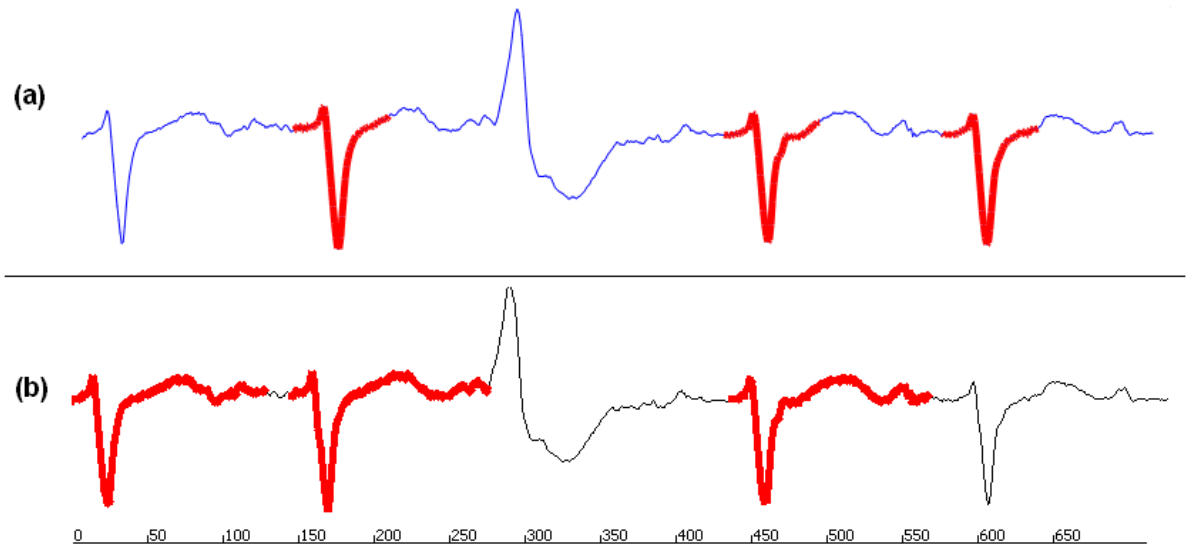



Figure 3.10: Comparison on ECG dataset with $a$=4 (a) Motifs discovered by SAX (shown by Viz-Tree). (b) Closed motifs discovered by *closedMotif* (shown by ARC-VIEW).

*Electrocardiograms* (ECGs) are the interpretations of the electrical activity of the heart over a period of time. In this experiment, the same dataset in Figure 3.1 is used to illustrate *closedMotif*'s ability in discovering complete motifs. This time series dataset is obtained from the MIT-BIH Noise Stress Test Dataset, PhysioBank [151]. Figure 3.10 clearly shows that resultant motifs discovered by *closedMotif* and SAX are different. In Figure 3.10(a), SAX algorithm only produces incomplete motifs, which are subsequences of the hidden patterns. Our proposed algorithm is superior to SAX as it discovers closed motifs, which allows us to obtain a complete heartbeat cycle.

#### 3.4.2.2 Discovery of varying-length motifs:

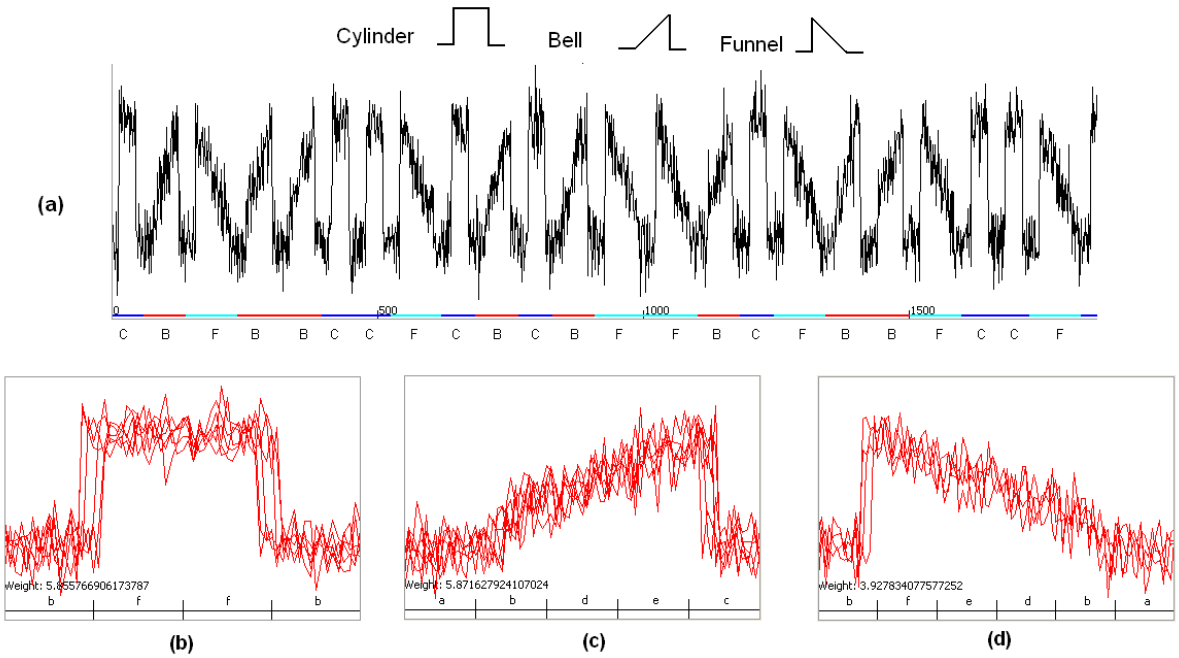



Figure 3.11: (a) Cylinder-Bell-Funnel dataset with $a$=6. (b) Discovered cylinder motifs of length 64. (c) Discovered bell motifs of length 80. (d) Discovered funnel motifs of length 96.

*Cylinder-Bell-Funnel* (CBF) is a popular synthetic dataset, which is widely used in time series analysis. Three are three different shapes (cylinder, bell, and funnel), which are randomly inserted into the time series. In order to test our algorithm's capability of finding motifs of various lengths, we generated the three shapes with different lengths: cylinder shape with length 64, bell shape with length 80, and funnel shape with length 96. First, we run the k-motif algorithm three times with window sizes 64, 80, and 96. As a result, the k-motif algorithm can only find motifs having the same size of the window; it can only find cylinder motifs with window size 64, bell motifs with window size 80 and funnel motifs with window size 96. Then, we run our closedMotif algorithm with an initial window size 64. In contrast, our algorithm can find all the CBF motifs within one

pass only. Figures 3.11(b), (c), (d) shows the discovered cylinder, bell, and funnel motifs respectively.

#### 3.4.2.3 Scalability with Streaming Time Series

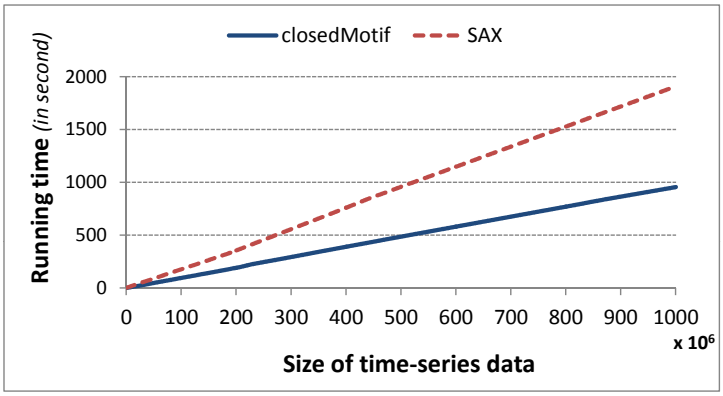



Figure 3.12: Running time v.s. data size

We further perform our proposed algorithm in a streaming mode and evaluate its scalability with large datasets. We choose the random walk time series, as it is one of the most experimented datasets in motif discovery $T(n+1) = T(n) + \mathbb{N}(0,1)$, where $\mathbb{N}(0,1)$ is a random Gaussian generator [40, 97, 152, 153]. We generate a massive streaming time series with a size of up to *1 billion* observations. For the SAX algorithm, we set the length of each word to 10. To compare $closedMotif$ and the SAX algorithm, we record their running time every one million instances. In Figure 3.12, when the size of time series data increases, the running time of $closedMotif$ increases linearly. Although the SAX algorithm has a similar linear time complexity, it can be observed that our algorithm is much faster; $closedMotif$ only needs to read a next SAX symbol to find the longest motif, while SAX needs to calculate every SAX word for each time window. Furthermore, $closedMotif$ can discover more important motifs than SAX as it has the capability to discover closed motifs with different lengths.

### 3.4.3 Time Series Classification

In time series classification, *closedMotif* extracts closed motifs that are most distinctive for each class. Then, it applies a variant of the nearest neighbor algorithm to perform classification. We denote our classifier (described in Algorithm 3) as *1-NN closedMotif*. We compare it with 1-NN Euclidean Distance classification (*1-NN ED*) and Logical-Shapelet [9]. The *1-NN ED* classifier stores the whole training dataset. It computes the Euclidean distance between a testing time series and each time series in the training set. The testing time series is assigned the same label of the training time series with the shortest distance. We choose the 1-NN ED classifier as it is a competitive baseline algorithm [43]. We also select Logical-Shapelet since it is a state-of-the-art classifier with good performance. We show three case studies in three different domains in the following sections.

#### 3.4.3.1 Video Surveillance

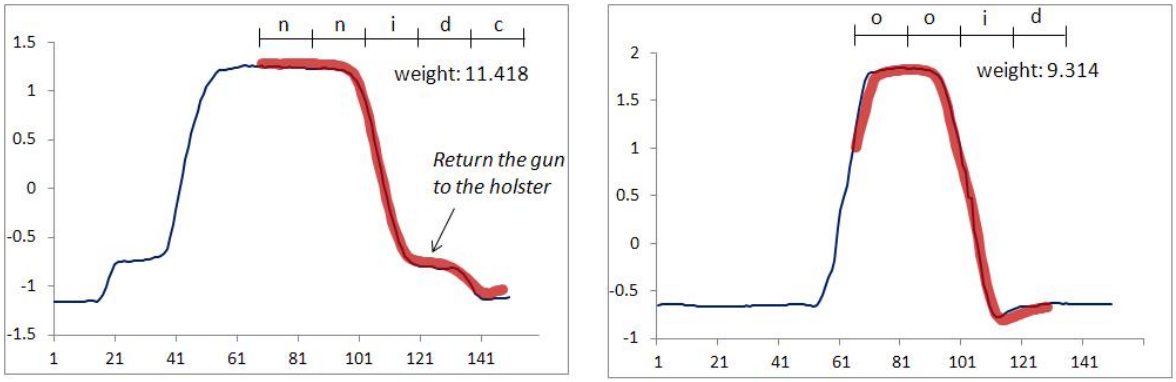



Figure 3.13: The Gun-Point dataset with the most representative motifs of each class.

The Gun-Point dataset that comes from the video surveillance domain is one of the most studied datasets in time series classification [2, 6, 7] . It tracks the right hand of actors, and records its vertical position in two cases. In the *Gun-Draw* case, the actors

repeatedly draw a gun from a hip-mounted holster, aim at a target and put the gun back to the holster. In the *Point* case, the actors repeat the same motion without a gun and they use their index fingers to point to a target instead. The dataset has 40/150 training/testing time series with a length of 150. The accuracy of *1-NN closedMotif* is 94.00%, which outperforms Logical-Shapelet (93.30% accuracy) and 1-NN ED (91.30% accuracy). Moreover, the closedMotif algorithm also provides more information about the difference between two classes. Figure 3.13 shows the most distinctive motifs of two classes. When the actor has a gun, his hand stays on the holster's position for a while before putting the gun back. In Figure 3.13(a), *nnidc* motif with weight 11.418 is most distinctive for the Gun class. In contrast, *ooid* motif with weight 9.314 in Figure 3.13(b) shows that the actor's hand moves freely without the gun.

#### 3.4.3.2 Sensor Network Data

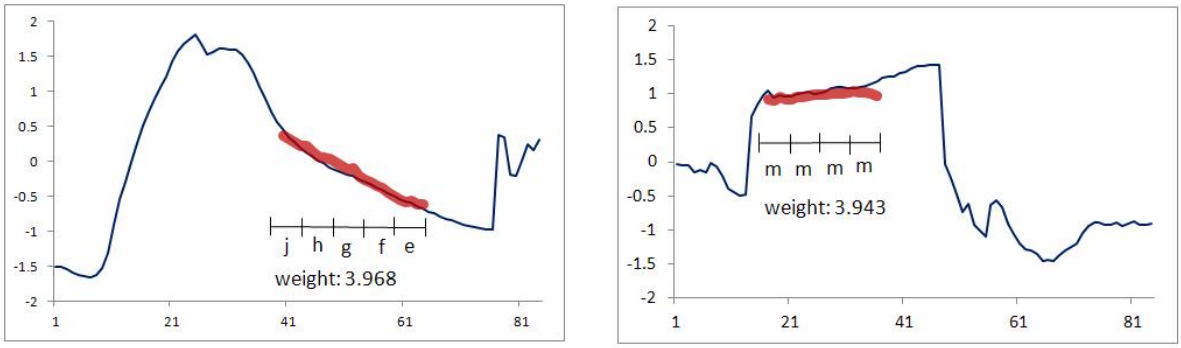
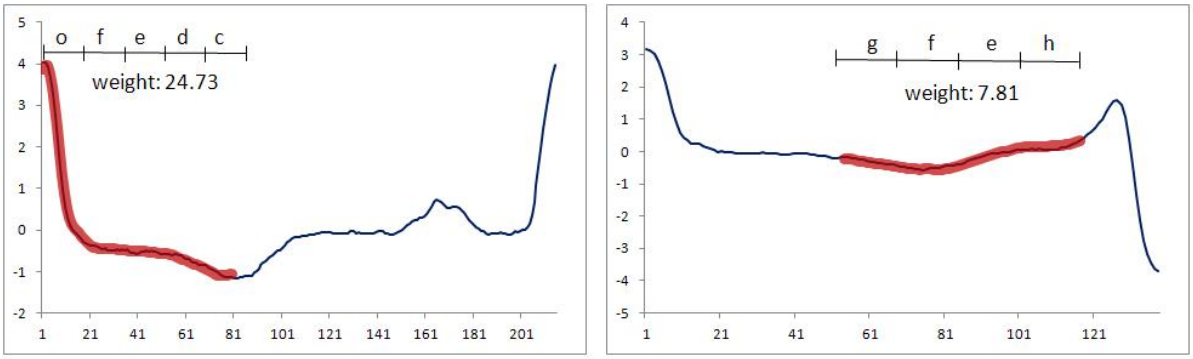



Figure 3.14: The MoteStrain dataset with the most representative motifs of each class.

Sensor networks have attracted a lot of research interest with a broad range of applications. The MoteStrain dataset is collected from wireless sensors (Berkeley motes), which were deployed to monitor the environment inside the Intel Berkeley Research Lab [4,154]. The dataset consists of 20 training time series and 1252 testing time series, each with a length of 84. We examine the usefulness of closed motifs for the classification of data

from two different sensors. With an accuracy of 89.38%, *1-NN closedMotif* outperforms Logical-Shapelet (83.23% accuracy), and 1-NN ED (87.90% accuracy). Figure 3.14 shows the most distinctive motifs of the two sensors. We can see that there is a downward motif $jhgfe$ with weight 3.968 that is representative for the first sensor in Figure 3.14(a) and a flat motif $mmmm$ with weight 3.943 that is distinctive for the second sensor in Figure 3.14(b).

#### 3.4.3.3 Biomedical Data




(a) Normal ECG (b) Arrhythmia ECG

Figure 3.15: The TwoLeadECG dataset with the most representative motifs of each class.

Heart disease has become one of leading single causes of death and thus, accurate diagnoses can be life-saving. Electrocardiograms (ECGs) that record heartbeats in relation to time are routinely used for initial screening and diagnosis. We perform experiments with the arrhythmia ECG data in the Physionet MIT-BIH Arrhythmia database [151]. The data contains two-channel ambulatory ECGs sampled at 360 Hz per channel with a 11-bit resolution. Each channel is independently annotated by two or more cardiologists; disagreements were resolved to obtain the computer-readable reference annotations for each beat. We collect the first ECG channel of patient 233 for this study. All the above classifiers are deployed to distinguish between normal ECG and arrhythmia ECG. Our *1-NN closedMotif* achieved the best accuracy of 88.50% while Logical-Shapelet achieved

Table 3.1: Comparisons of accuracies of different algorithms on various datasets.

| Dataset | #Classes | Size of training set | Size of testing set | Time series length | 1-NN ED | Logical-Shapelet | 1-NN closedMotif | Symbol length | Alphabet size |
|---|---|---|---|---|---|---|---|---|---|
| Gun-Point | 2 | 50 | 150 | 150 | 91.30% | 93.30% | **94.00%** | 16 | 15 |
| MoteStrain | 2 | 20 | 1252 | 84 | 87.90% | 83.23% | **89.54%** | 5 | 15 |
| ArrhythmiaECG | 2 | 100 | 113 | 285 | 85.50% | 76.11% | **85.50%** | 16 | 15 |
| CBF | 3 | 30 | 900 | 128 | 85.00% | 88.56% | **99.67%** | 16 | 15 |
| TwoLeadECG | 2 | 23 | 1139 | 82 | 74.70% | 85.60% | **89.03%** | 4 | 15 |
| DiatomSizeReduction | 4 | 16 | 306 | 345 | 93.5%5 | 75.16% | **96.41%** | 32 | 15 |
| FaceFour | 4 | 24 | 88 | 350 | 78.40% | 90.91% | **95.45%** | 20 | 15 |
| Trace | 4 | 100 | 100 | 275 | 76.00% | **100.00%** | 88.00% | 16 | 15 |
| **Average** | | | | | 84.44% | 86.61% | **92.58%** | | |

76.11%. Though 1-NN ED attained the same accuracy of *1-NN closedMotif*, it required a longer running time. Figure 3.15 shows that there are two distinctive motifs for normal ECG and arrhythmia ECG.

Table 3.1 summarizes accuracies of the above time series classifiers on various datasets. We further validate our algorithm with five more datasets: CBF, TwoLeadECG, DiatomSizeReduction, FaceFour, and Trace dataset [4]. We observe that *1-NN closedMotif* algorithm is more accurate than Logical-Shapelet in 7/8 datasets and 1-NN ED in 8/8 datasets. The average accuracy of *1-NN closedMotif* is 92.58%, which is higher than the average accuracies of Logical-Shapelet and 1-NN ED by 5.97% and 8.14% respectively.

#### 3.4.3.4 Sensitivity of $m$

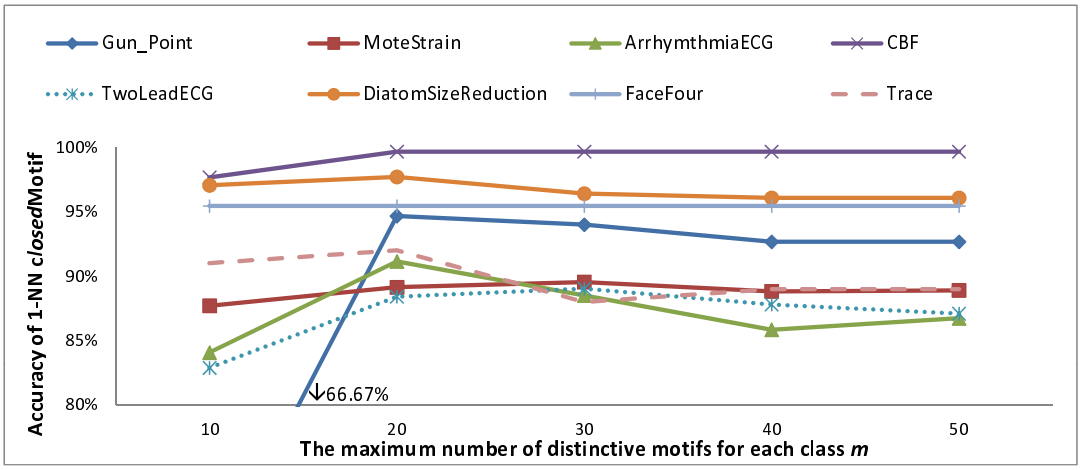



Figure 3.16: Sensitivity of the maximum numbers of distinctive motifs for each class $m$.

In this section, we conduct several experiments to evaluate the sensitivity of the maximum numbers of distinctive motifs for each class, $m$. We keep other parameters unchanged and increase the value of $m$ from 10 to 50 with each step of 10. Experimental results are shown in Figure 3.16. At the beginning, when the value of $m$ raises, the accuracy of *1-NN closedMotif* increases since it has more distinctive motifs to compare to the testing time series. However, its accuracy slightly decrease when the value of $m$ is too large. This is because *1-NN closedMotif* contains "contaminated" motifs that may not purely belong a certain class. We suggest that the value of $m$ should be set between 20 to 30 to achieve good accuracy.

## 3.5 Summary

Motivated by the importance of discovering motifs with appropriate lengths, we have introduced the definition of *closed motifs* and proposed a novel time series classifier that is based on closed motifs. We first build a synopsis of streaming time series data in the form of a suffix tree, which can be constructed and updated in real-time with only a single pass. This property allows the algorithm to process massive time series at high data rates. Then, applying the downward closure property, we traverse the tree to discover closed motifs. We further implement our algorithm together with a user-friendly toolkit ARC-VIEW, which is more practical than VizTree [43] in visualizing high-resolution motifs. Moreover, the toolkit is equipped with highlighting functions that help users easily monitor and evaluate discovered motifs. Extensive experiments with both synthetic and real time series datasets have shown that our $closedMotif$ algorithm is able to discover closed motifs with different lengths and is scalable for massive streaming time series datasets. Moreover, we demonstrate the usefulness of closed motifs in time series classification. The nearest neighbor classifier that is constructed from the most distinctive

closed motifs outperforms prominent classifiers in terms of accuracy and speed for many datasets in various domains.

In the next chapter, we continue to examine a more difficult problem: classification for high-dimensional data streams, where datasets are multivariate, huge and unbounded. We will propose an algorithm to solve the problem of dynamic feature space, which was discussed earlier in Chapter 2.

*"We're entering a new world in which data may be more important than software."*

Tim O'Reilly(1954-) Founder of O'Reilly Media

*"Action is the foundational key to all success."*

Pablo Picasso (1881-1973) Artist

# 4

# Heterogeneous Ensemble for Feature Drifts in Data Streams

## 4.1 Introduction

In the previous chapter, we have addressed the classification problem of streaming time series by proposing an efficient and accurate closed-motif-based classifier. Here, we will tackle the classification of high-dimensional data streams with more challenging characteristics. With rapid technological advancement, *data streams* are commonly generated in many real-life applications, such as stock markets, online stores and sensor networks. In order to discover knowledge from data streams, scientists have to confront the following challenges: (1) tremendous volumes of data; (2) dynamic changes of the discovered patterns, which is commonly referred to as *concept drifts*; and (3) real-time response. Concept drifts are categorized into two types: gradual drifts with moderate changes and sudden drifts with severe changes. There are two common approaches of existing classification models for data streams: online incremental learning and ensemble learning.

Incremental learning trains a single classifier and updates it with newly arriving data. For example, Domingos and Hulten [11] proposed a very fast Hoeffding tree learner (VFDT) for data streams. The VFDT was later extended to CVFDT [12], which can handle the concept drifting streams by constructing alternative nodes to replace the outdated nodes when concept drifts occur. Incremental learning is quite efficient but it cannot adapt to sudden drifts. Ensemble learning, which aims to combine multiple classifiers for boosting classification accuracy, has attracted a lot of research due to its simplicity and good performance. It manages concept drifts using one of the following adaptive approaches: (1) majority vote or weighting combination [67], (2) real-time updates the individual classifiers [18, 68], and (3) changing the ensemble structure by replacing outdated classifiers [19, 22]. However, it has high computational complexity as there are many time-consuming processes, eg. generating new classifiers, and updating classifiers.

In this chapter, we address the above problems by presenting a novel framework to integrate feature selection techniques and ensemble learning for data streams. To alleviate ensemble updating, we propose a new concept of *"feature drifts"* and use it to optimize the updating process. With a gradual drift, each classifier member is updated in a real-time manner. When a feature drift occurs, which represents a significant change in the underlying distribution of the dataset, we train a new classifier to replace an outdated classifier in the ensemble.

Moreover, feature selection helps to enhance ensemble learning. It not only improves the accuracy of classifier members by selecting the most relevant features and removing irrelevant and redundant features, but also reduces the complexity of the ensemble significantly as only a small subset of feature space is processed. Finally, we propose a heterogeneous ensemble where different types of classifier members are well selected to maximize the diversity of the ensemble [155, 156]. Figure 4.1 gives an overview of our

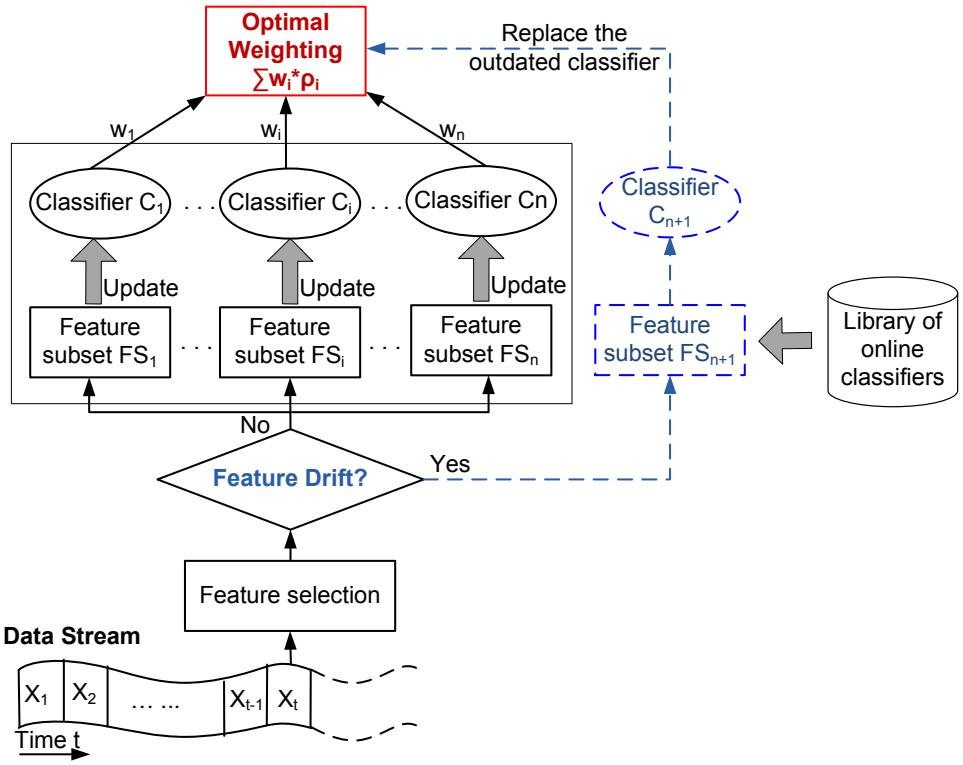



Figure 4.1: Ensemble Learning of Feature Drifts.

framework. The ensemble consists of many classifiers, each of which has its own feature subset. If there is a feature drift, the ensemble is updated with a new classifier together with a new feature subset; otherwise, each classifier is updated accordingly. To aggregate the classification results of classifier members, we assign each classier a weight w.r.t its performance. This weighting method is proven to minimize the expected added error of the ensemble.

In summary, the following are contributions of our framework:

- We enhance ensemble learning with feature selection which helps to lessen computational complexity and increase accuracy.
- We propose the definition of feature drifts and explore relationships between feature drifts and concept drifts.

- We significantly increase the accuracy of the ensemble by designing a heterogeneous ensemble with well-chosen member classifiers and an optimal weighting scheme.

## 4.2 Proposed Framework

In this section, we will propose a novel framework for integrating feature selection and heterogeneous ensembles. Our framework not only adapts to different kinds of concept drifts, but also has low complexity due to its dynamic updating scheme and the support of feature selection techniques.

We assume infinite data streams $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}, \ldots]$ as input in the framework, where $\mathbf{x_t} = [\mathbf{f_t^1}, \mathbf{f_t^2}, \ldots, \mathbf{f_t^d}]^\mathbf{T}$ is a $d$-dimensional vector arriving at time $t$. We assume the data streams have $c$ different class labels. For any data vector $\mathbf{x_t}$, it has a class label $\mathbf{y_t} \in \{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_c}\}$. Generally when the dimension $d$ is large, there is often only a small set of key features that is critical for building accurate models for classification.

Data streams tend to evolve over time and so do the key features correspondingly. For ease of discussion, we restate the definitions of *concepts* and *concept drifts*, as well as present a definition of *feature drifts*:

**Definition 1** *A **concept** or a data source is defined as a set of the prior probabilities for the classes $P(y_i)$ and the class-conditional density functions (pdf) $P(X|y_i), i = 1, \ldots, c$ (definition from [78]):*

$$S = \{(P(y_1), P(X|y_1)); \ldots; (P(y_c), P(X|y_c))\}.$$

**Definition 2** *Suppose a data stream $\mathbf{X}$ consists of a set of $k$ data sources $S_i$ with influence $w_i$. The underlying data distribution of data stream $\mathbf{X}$ is $D^X(t) = \{w_1(t)S_1, \ldots, w_k(t)S_k\}$, where $w_i(t) \in [0, 1]$ and $\sum_{i=1}^{k} w_i(t) = 1$. If for any two time stamps $t_1$ and $t_2$ that $D^X(t_1) \neq D^X(t_2)$, we say there is a **concept drift**.*

**Definition 13** *Given a feature space $\mathcal{F}$, at time point t, we can always select the most discriminative subset $\widehat{\mathcal{F}}_t \subseteq \mathcal{F}$. If for any two time points i and j $\widehat{F}_i \neq \widehat{F}_j$, we say that there is a* ***feature drift****.*

Next, we explore the relationship between concept drifts and feature drifts.

**Lemma 1** *Concept drifts give rise to feature drifts.*

**Proof:** Assume that certain feature selection techniques evaluate the discrimination of a feature subset $\mathcal{F}$ at time $t$ by a function:

$$\Psi(\mathcal{F}_t, t) = \Psi(P(f^i|y_j), t), \quad f^i \in \mathcal{F}_t \subseteq \{f^1, \ldots, f^p\}, y_j \in \{y_1, \ldots, y_c\}$$

And, there is a feature drift in $[t_1, t_2], (t_1 < t_2)$,

$$\begin{cases} \hat{\mathcal{F}}_{t_1} = argmax_{\mathcal{F}_i \subseteq \mathcal{F}} \Psi(\mathcal{F}_i, t_1) \\ \hat{\mathcal{F}}_{t_2} = argmax_{\mathcal{F}_i \subseteq \mathcal{F}} \Psi(\mathcal{F}_i, t_2) \\ \hat{\mathcal{F}}_{t_1} \neq \hat{\mathcal{F}}_{t_2} \end{cases}$$

We can always find a feature $f^a \in \mathcal{F}$ and a class $y_b$ so that $P(f^a|y_b, t_1) \neq P(f^a|y_b, t_2)$. Else, $P(f^i|y_j, t_1) = P(f^i|y_j, t_2), \forall(i, j)$, then $\hat{\mathcal{F}}_{t_1} = \hat{\mathcal{F}}_{t_2}$. Since the data stream $X$ consists of the feature $f_a$ ; therefore, $P(X|y_b, t_1) \neq P(X|y_b, t_2)$. That means at least two concepts have changed their influence; hence, $D^X(t_1) \neq D^X(t_2)$. This denotes a concept drift in the time interval $[t_1, t_2]$.

**Lemma 2** *Concept drifts may not lead to feature drifts.*

**Proof:** For example, given data streams $\mathbf{X}$ within a time period $[t_1, t_2]$ we assume the prior probability of a class $i$ and $j$, $P(y_i)$ and $P(y_j)$, are changed; but their sum $(P(y_i) + P(y_j))$ and other probabilities remain the same. In this scenario, there is a concept drift but no feature drift.

Figure 4.2 shows examples of a concept drift and a feature drift. In Figure 4.2(a), there is a concept drift of changing between two data sources in the discriminate feature

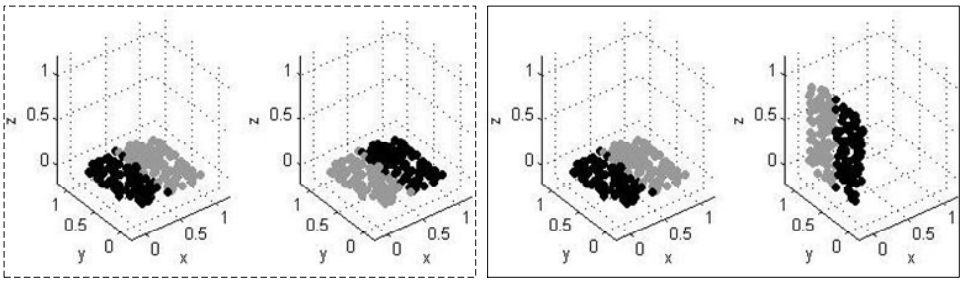



Figure 4.2: (a) A concept drift of changing two data sources in the discriminate feature subset $(x, y)$. (b) A feature drift, where the discriminate feature subset changes from $(x, y)$ to $(y, z)$. Class $y_1$ is depicted with grey dots, and class $y_2$ with black dots.

subset $(x, y)$, but no feature drift. Figure 4.2(b) illustrates a feature drift, where the discriminate feature subset changes from $(x, y)$ to $(y, z)$. This feature drift also implies a concept drift.

Combining Lemmas 1 and 2, we can conclude that:

**Theorem 4.1** *Feature drifts occur at a slower rate than concept drifts.*

Feature drifts, which are observed in high dimensional data streams, occur no faster than concept drifts. As shown in the overview of our framework Figure 4.1, we need to modify feature selection techniques to detect feature drifts. The key idea is using feature selection to accelerate ensemble learning and steer its updating process. Moreover, feature selection techniques not only remove irrelevant and redundant features, but also accelerate the learning process by reducing the dimensionality of the data. Thus, the overall performance of the ensemble is improved in terms of accuracy, time and space complexities.

First, we select online classifiers as classifier members as they can be incrementally updated with new data. Then, we construct a heterogeneous ensemble with a capability to adapt to both concept and feature drifts. With gradual drifts, we only need to update the classifier members. With feature drifts, we adjust the ensemble by replacing the

outdated classifier with a new one. We further deploy a weighting technique to minimize the cumulative error of the heterogeneous ensemble.

### 4.2.1 Feature Selection Block

Feature selection selects a subset of $q$ features from the original $d$ features ($q \leq d$) so that the feature space is optimally reduced. Generally, we expect the feature selection process to remove irrelevant and redundant features. We decide to use FCBF (Fast Correlation-Based Filter) as it is simple, fast and effective [69]. FCBF is a multivariate feature selection method where the class relevance and the dependency between each feature pair are taken into account. Based on information theory, FCBF uses symmetrical uncertainty to calculate dependencies of features and the class relevance. Starting with the full feature set, FCBF heuristically applies a backward selection technique with a sequential search strategy to remove irrelevant and redundant features. The algorithm stops when there are no features left to eliminate.

Symmetrical Uncertainty ($SU$) uses entropy and conditional entropy values to calculate dependencies of features. If $X$, $Y$ are random variables, $X$ receives value $x_i$ with probability $P(x_i)$, $Y$ receives value $y_j$ with probability $P(y_j)$; the symmetrical uncertainty between $X$ and $Y$ is:

$$SU(X,Y) = 2\left[\frac{H(X) - H(X|Y)}{H(X) + H(Y)}\right] = 2\left[\frac{I(X,Y)}{H(X) + H(Y)}\right] \tag{4.1}$$

where $I(X,Y)$ is the mutual information between $X$ and $Y$, $H(X)$ and $H(Y)$ are the entropies of $X$ and $Y$ respectively; the higher the $SU(X,Y)$ value, the more dependent $X$ and $Y$ are.

We improve FCBF with a sliding window so that it has low time and space complexities. Incoming data is stored in a buffer (window) with a predefined size. Next, the matrix of symmetrical uncertainty values is computed to select the most relevant

**Algorithm 4** Ensemble Learning

**Input:** A series of infinite streaming data $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}, \ldots]$, where $\mathbf{x_t}$ is a $d$ dimensional vector $[f_t^1, f_t^2, \ldots, f_t^p]^T$ with a class label $y_t$ arriving at time $t$, $y_i \in \{y_1, y_2, \ldots, y_c\}$.
A set of $l$ different classifier types, $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_l\}$.
The maximum size of the ensemble, $k \times l$.
**Output:** A heterogenous ensemble $\mathcal{E}$.

```
Initialize the ensemble E with k classifiers of each model in M, denoted as C_1, C_2, ..., C_{k×l}.
while X has more instance do
    if chunk is not full then
        Add x_i to chunk.
    else
        Perform FCBF to get the relevant and non-redundant feature subset φ_i.
        if φ_i ≠ φ_{i−1} then
            Find the best accurate classifier C_best in the ensemble and get its type.
            Build a new classifier C_new having the same type with C_best, and associated with the feature subset φ_i.
            Remove a classifier with the worst accuracy from E.
            Add the new created classifier C_new into E.
        end if
        for each classifier in the ensemble C do
            for each instance x in chunk do
                Set m according to Poisson(1)
                Update m times each classifier member with x.
            end for
        end for
    end if
end while
```

feature subset. The process is performed in a sliding window fashion, and the selected feature subsets are monitored to detect feature drifts. When two consecutive subsets are different, we postulate that a feature drift has occurred.

## 4.2.2 Ensemble Block

### 4.2.2.1 Heterogeneous Ensemble

When constructing an ensemble learner, the diversity among member classifiers is expected to be the key contributor to the accuracy of the ensemble. Furthermore, a

heterogeneous ensemble that consists of different classifier types usually attains high diversity [155, 157]. Motivated by this observation, we construct a small heterogeneous ensemble rather than a big homogeneous ensemble with a large number of same type classifiers, which will compromise speed.

As mentioned, we aim to select online classifiers so that the ensemble can properly adapt to different types of concept drifts. Here, CVFDT [12] and Online Naive Bayes (OnlineNB) are chosen as the basic classifier types, but the framework can work with any classification algorithm. The OnlineNB is an online version of the Naive Bayes classifier. When a training instance $(x_t, y_t)$ comes, OnlineNB updates the corresponding prior and likelihood probabilities, $P(y_t)$ & $P(F_i = f_i^t|y_t)$. To classify a testing instance, it applies Bayes' theorem to select a class having the maximum posterior probability as follows:

$$OnlineNB(x_t) = argmax_{y_j} P(y_j) \prod_{i=1}^{n} P(F_i = f_i^t|y_j) \tag{4.2}$$

Details of the heterogeneous ensemble's learning process are given in Algorithm 4. Given a data stream $X$ and a predefined set $\mathcal{M}$ of different classifier types, we initialize the ensemble with $k$ classifiers of each type in $\mathcal{M}$. Next, data streams are processed in a sliding window mode and data instances are grouped into predefined-size chunks. When a new chunk arrives, we apply a feature selection technique to find the most discriminative feature subset. If the subset is different from the previous one, there is a feature drift. We would then need to construct a new classifier with the selected feature subset. We add it to the ensemble, and remove the worst classifier if the ensemble is full; the new classifier will be of the same type as the best classifier member (lines 7-12). Finally, we employ online bagging [18] for updating classifier members to reduce the variance of the ensemble (lines 13-18).

#### 4.2.2.2 Ensemble Classification

**Algorithm 5** Real Time Classification

**Input:** A new testing unlabeled instance $x_t$.
An ensemble $\mathcal{E}$ of $N$ classifiers, denoted as $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N$. Every classifier $\mathcal{C}_i$ is associated with a feature subset $\varphi_\mathbf{i}$.
Given a testing instance, every classifier $\mathcal{C}_i$ outputs a probability distribution vector $\rho_i = [\rho_{i1}, \rho_{i2}, \ldots, \rho_{ic}], (\sum_{j=1}^{c} \rho_{ij} = 1,\ i \in \{1, 2, \ldots, N\})$.
A padding value, $\alpha = 0.001$.
**Input:** A probability distribution vector of $x_t$.

1: **for** every classifier $\mathcal{C}_i$ in the ensemble $\mathcal{E}$ **do**
2: Project the arriving testing instance $\mathbf{x}_t$ onto a low dimensional feature space $\varphi_\mathbf{i}$ and get $\widehat{\mathbf{x}}_t$.
3: Compute the probability distribution vector $\rho_i = \mathcal{C}_i(\widehat{\mathbf{x}}_t)$.
4: Get the aggregated error of classifier $\mathcal{C}_i$, $err_i$, and calculate the weight following the Equation 4.5, $w_i = 1/(err_i + \alpha)$.
5: **end for**
6: Aggregate all probability distribution vectors as follows:

$$\mathcal{E}(\rho) = \sum_{i=1}^{z} w_i * \rho_i$$

7: Normalize and return vector $\mathcal{E}(\rho)$.

Based on the research work of Tumer *et. al* [158] and Fumera *et. al* [159], the estimated error of the ensemble is:

$$E_{add}^{ens} = \sum_{k=1}^{N} w_k^2 E_{add}^k + \sum_{k=1}^{N} \sum_{l \neq k} w_l w_k \left[\beta^k \beta^l + \rho^{kl}(\sigma_i^k \sigma_i^l + \sigma_j^k \sigma_j^l)/s^2\right], \tag{4.3}$$

where $\beta^k$ and $\sigma^k$ are the bias and the standard deviation of the estimate error $\varepsilon^k$ of classifier $C_k$ , $\rho^{kl}$ is the correlation coefficient of the errors $\varepsilon^k$ and $\varepsilon^l$.

We assume that classifier members are unbiased and uncorrelated (i.e. $\beta^k = 0, \rho^{kl} = 0, k \neq l$). Then, we have $E_{add}^{ens} = \sum_{k=1}^{N} w_k^2 E_{add}^k$, $\sum_{k=1}^{N} w_k = 1$. To minimize the added error of the ensemble, the weights of classifier members are set as follows:

$$w_k = (E_{add}^k)^{-1} \left[\sum_{m=1}^{N} (E_{add}^m)^{-1}\right] \tag{4.4}$$

When constructing the ensemble classifier, we estimate the added error of every classifier member, $E_{add}^k$, which is its accumulated error from its creation time to the current

time. Moreover, to mitigate the extreme case of $E^m_{add} \approx 0$, we modify the Equation 4.4 as follows:

$$w_k = (E^k_{add} + \alpha)^{-1} \left[ \sum_{m=1}^{N} (E^m_{add} + \alpha)^{-1} \right], \tag{4.5}$$

where $\alpha$ is a padding value which is empirically set to 0.001.

Details of the ensemble classification process are shown in Algorithm 5. Given the ensemble $\mathcal{E}$ where each member can classify a testing instance and output the result as a probability distribution vector, we use a weighting combination scheme to get the result. First, for each classifier $\mathcal{C}_i$ we project the testing instance $x_t$ onto the subspace $\varphi_\mathbf{i}$. Then, the classifier $\mathcal{C}_i$ processes the projected instance and outputs a probability distribution vector. We attain the aggregated accuracy of $\mathcal{C}_i$ from its creation time, and calculate its optimal weight according to Equation 4.5 to minimize the expected added error of the ensemble. Finally, the ensemble's result is set as the normalized sum of the weighted classification results of the members (lines 6-7).

## 4.3 Experiments and Analysis

### 4.3.1 Experimental Setup

For our experiments, we use three synthetic datasets and three real-life datasets. The three synthetic datasets, SEA generator (SEA), Rotating Hyperplane (HYP), and LED dataset (LED), are generated from the MOA framework [160]. Concept drifts are generated by changing 10 attributes at a speed of 0.001 per instance in the HYP dataset. The three real life datasets are: network intrusion (KDD'99[1]), hand-written digit recognition (MNIST[2]), and protein crystallography diffraction (CRYST[3]) datasets. Table 4.1 shows the characteristics of the six datasets.

[1]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[2]http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets
[3]http://ajbcentral.com/CrySis/dataset.html

Table 4.1: Characteristics of datasets used for evaluation.

| Name | #Instances | #Attributes | #Classes | Noise | #Selected Features | **Ratio of FS** |
|---|---|---|---|---|---|---|
| SEA | 100,000 | 3 | 2 | 10% | 2.25 | **75%** |
| HYP | 100,000 | 10 | 2 | 5% | 6.71 | **67.13%** |
| LED | 100,000 | 24 | 3 | 10% | 15.84 | **65.98%** |
| KDD'99 | 494,022 | 34 | 5 | N/A | 2.33 | **6.87%** |
| MNIST | 60,000 | 780 | 10 | N/A | 30.77 | **3.95%** |
| CRYST | 5,500 | 1341 | 2 | N/A | 7.49 | **0.56%** |

We compare our algorithm HEFT-Stream with other prominent ensemble methods: AWE [19] and OnlineBagging [18]. We further verify the effect of the feature selection technique in the performance of HEFT-Stream by running HEFT-Stream without the feature selection, denoted as *HEFT-Stream-noFS*. It is worthy noted that HEFT-Stream-noFS is equivalent to a heterogenous ensemble of OnlineNB and CVFDT classifiers. The experiments were conducted on a Windows PC with a Pentium D 3GHz Intel processor and 2GB memory. To enable more meaningful comparisons, we use the same parameter values for all the algorithms. The number of classifier members is set to 10 for all the ensemble algorithms, and the chunk size is set to 1000. To simulate the data stream environment, we process all experiments using a practical approach, called *Interleaved-Chunk*. In this approach, data instances are read to form a data chunk. Each new data chunk is first used to test the existing model. Then it is used to update the model and it is finally discarded to save memory.

### 4.3.2 Experimental Results

As AWE and OnlineBagging are homogeneous ensembles and can only work with one classifier type, we set different classifier types for these ensembles accordingly. For AWE, we set classifier members as Naive Bayes and C4.5, which are recommended by the authors [19], and denoted as *AWE(NB)* and *AWE(C4.5)*. For OnlineBagging, we set its classifier members as OnlineNB and CVFDT, and denoted as *Bagging(OnlineNB)* and *Bagging(CVFDT)* respectively. We conduct the experiments ten times for each dataset

and summarize their average accuracy and running times in Table 4.2. Readers may visit our *website*[4] for the algorithms' implementation, more experimental results, and detailed theoretical proofs.

We observe that the AWE ensemble has the worst accuracy and the longest running time. This is because the AWE ensemble uses traditional classifiers as its members and trains them once; when there are concept drifts, these members become outdated and accuracy is degraded. Moreover, the AWE ensemble always trains a new classifier for every upcoming chunk, and this increases processing time. The OnlineBagging has better performance but it largely depends on the classifier type. For example, Bagging(OnlineNB) is more accurate and faster than Bagging(CVFDT) for the LED dataset but Bagging(OnlineNB) become less precise and slower than Bagging(CVFDT) for the KDD'99 dataset. It is also noteworthy that both AWE and OnlineBagging do not work well with high dimensional datasets, such as MNIST and CRYST.

[4]https://sites.google.com/site/heftstream

| Dataset | AWE(NB) | | AWE(C4.5) | | Bagging(OnlineNB) | | Bagging(CVFDT) | | HEFT-Stream-noFS | | HEFT-Stream | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | Time | Acc | Time | Acc | Time | Acc | Time | Acc | Time | Acc | Time |
| SEA | 88.08 | 6.61 | 88.12 | 26.08 | 87.91 | 2.9 | 89.12 | 21.47 | 88.95 | 21.46 | **89.28** | 22.65 |
| HYP | 87.94 | 19.42 | 72.72 | 27.40 | 86.92 | 8.07 | 88.90 | 40.41 | 86.95 | 39.04 | **89.18** | 12.42 |
| LED | 73.91 | 74.33 | 72.13 | 40.34 | 73.93 | 26.21 | 73.79 | 83.00 | 73.74 | 72.04 | **74.07** | 28.02 |
| KDD'99 | 95.09 | 280.33 | 94.68 | 281.33 | 92.95 | 230.3 | **97.75** | 209.6 | 96.89 | 246.47 | 96.37 | 142.0 |
| MNIST | 9.87 | 2054.0 | 78.49 | 1246.7 | 9.87 | 1286.00 | 21.13 | 1456.00 | 19.65 | 1727 | **79.36** | 439.00 |
| CRYST | 53.70 | 40.38 | 83.30 | 147.00 | 54.28 | 57.63 | 76.18 | 101.33 | 53.97 | 10.11 | **83.52** | 37.10 |
| **Average** | 68.10 | 412.51 | 81.57 | 294.80 | 67.64 | 268.53 | 74.48 | 318.65 | 70.03 | 317.69 | **85.30** | 113.53 |

Table 4.2: Comparisons of AWE(NB), AWE(C4.5), Bagging(OnlineNB), Bagging(CVFDT), HEFT-Stream without feature selection, and HEFT-Stream. Time is measured in seconds. For each dataset, the highest accuracy value is **boldfaced**, and the lowest running time is underlined.

Our approach, HEFT-Stream, addresses the above problems and achieves better performance than AWE and OnlineBagging. Without incorporated with feature selection techniques, HEFT-Stream-noFS cannot detect feature drifts and therefore cannot replace outdated classifier members. HEFT-Stream-noFS somehow performs in the middle of Bagging(OnlineNB) and Bagging(CVFDT). HEFT-Stream has the best accuracy and the lowest running time for most datasets. It continuously updates classifier members with gradual drifts, and only trains new classifiers whenever there are feature drifts or sudden drifts. This property not only enables HEFT-Stream to adapt to different types of concept drifts but also conserve computational resources. Furthermore, HEFT-Stream can dynamically change the ratio among classifier types to adapt to different types of datasets; when a particular classifier type works well with a certain dataset, its ratio in the ensemble will be increased. Finally, with integrated feature selection capability, HEFT-Stream only works with the most informative feature subsets which improves accuracy and reduces processing time. The last column of the Table 4.1 shows the ratios of the selected features to the full feature sets for all datasets. We realize that feature selection techniques are very useful for high dimensional datasets. For example, the percentages of the selected features are 3.95% for the MNIST dataset, and only 0.56% for the CRYST dataset.

### 4.3.3 Sensitivity Analysis

To assess the sensitivity of HEFT-Stream w.r.t the ensemble size, we use the KDD'99 dataset, and assign the chunk size to 1000. The number of classifier members is increased as follows: 6, 10, 20, 30, and 40. As shown in Figure 4.3, the running times are linearly increased with the ensemble size. However, the accuracy increases until the ensemble size reaches a certain value, then it goes down. A possible explanation is that in the first steps the ensemble's variance is reduced as there are more classifier members. But when

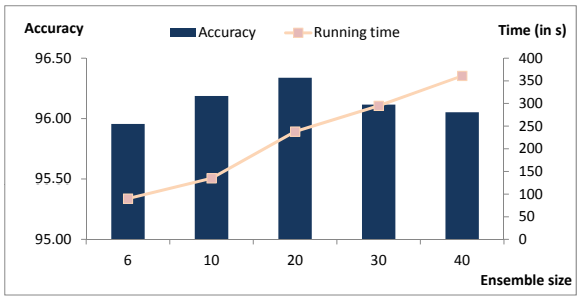



Figure 4.3: Sensitivity of ensemble sizes on KDD'99 dataset.

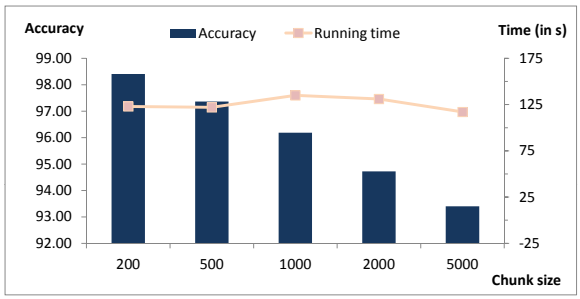



Figure 4.4: Sensitivity of chunk sizes on KDD'99 dataset.

the ensemble insists too many members, it becomes steady and slowly adapts to concept drifts.

Similarly, we evaluate the sensitivity of HEFT-Stream w.r.t the chunk size. We conduct experiments with the KDD'99 dataset and the ensemble size equals 10; the chunk size is set as follows: 200, 500, 1000, 2000, and 5000. Figure 4.4 show the accuracy values and running times with different chunk's sizes. Intuitively, the running times are similar as the ensemble has to train and test with the same number of instances. With smaller chunk size, the ensemble is more accurate as it is more frequently updated. In real applications, the chunk size should be set according to the data streams' speed and the availability of labeled data.

## 4.4 Summary

In this chapter, we proposed a general framework to integrate feature selection and heterogeneous ensemble learning for stream data classification. Feature selection helps to extract the most informative feature subset which accelerates the learning process and increases accuracy. We first apply feature selection techniques on the data streams in a sliding window manner and monitor the feature subset sequence to detect feature drifts. The heterogeneous ensemble is constructed from well-chosen online classifiers. The ratios of classifier types are dynamically adjusted to increase the ensemble's diversity, and allows

the ensemble to work well with many kinds of datasets. Moreover, the ensemble adapts to the severity of concept drifts; we update the online classifier members for gradual drifts, and replace an outdated member by a new one for sudden drifts. We have conducted extensive experiments to show that our ensemble outperforms state-of-the-art ensemble classifiers for data streams.

In the next chapter, we will examine another aspect of data stream mining to maximize gains by performing multiple mining tasks at the same time. Furthermore, we consider this issue in a practical environment where only a small portion of data is labeled.

*"Coming together is a beginning; keeping together is progress; working together is success."*

Henry Ford (1863-1947) Businessman

*"Winners never quit and quitters never win."*

Vince Lombardi (1913-1970) Coach

# 5 Concurrent Semi-supervised Learning

## 5.1 Introduction

In the previous chapter, we have successfully handled the problem of classifying high-dimensional data streams. A heterogeneous ensemble classifier that is nicely integrated with a dynamic feature selection technique is able to adapt to different types of concept drifts and works well with various kinds of datasets. In this chapter, we further explore a more complex problem: concurrent mining of very sparsely labeled data streams.

Data streams are characterized as temporally ordered, read-once-only, fast-changing, and possibly infinite, compared to static datasets studied in conventional data mining problems. Although many studies have been conducted to deal with data streams, most existing data stream algorithms focus on stand-alone mining. In many practical applications, it is desirable to concurrently perform multiple types of mining in order to better exploit data streams. For example, website administrators will be interested to use click-

stream data to classify users into specific types and cluster webpages of similar topics at the same time in order to enhance the user's experience. In addition, concurrent mining offers a potential synergy: The knowledge gained by one mining task may also be useful to other mining tasks. Unfortunately, there is currently little research on concurrent stream mining.

In order to demonstrate the potential of concurrent stream mining, we propose a stream mining algorithm called CSL-Stream (Concurrent Semi-supervised Learning of Stream Data) that concurrently performs clustering and classification. We choose to focus on classification and clustering as they share common assumptions that data objects within a cluster (class) are similar and data objects belonging to different clusters (classes) are dissimilar. Moreover, it is observed that most data objects in a cluster belong to a dominant class, and a class can be represented by many clusters. Therefore, classifiers can leverage on clustering to improve its accuracy [161]. For example, webpages within a topic are most visited by a specific type of users, and a user may be interested in many topics.

Moreover, CSL-Stream is a semi-supervised algorithm that is applicable to many real applications where only a small portion of data is labeled due to expensive labeling costs. In order to exploit a valuable limited amount of labeled data, CSL-Stream maintains a class profile vector for each node in the synopsis tree of the data streams. The profile vectors play an important role as pivots for the clustering process. The two mining tasks not only run at the same time, but also mutually improve each other's results. The clustering considers class profile vectors to attain high-purity clustering results. Conversely, the classification benefits from clustering models in achieving higher accuracy, and coping better with unlabeled data or outliers. Finally, CSL-Stream uses an *incremental learning* approach where the clustering and classification models are continuously improved to handle concept drifts and where mining results can be delivered in a timely manner. By

re-using information from historical models, CSL-Stream requires only constant time to update its learning models with acceptable memory bounds. Furthermore, when the portion of labeled instances is very small ($\simeq 1\%$), we propose a novel active learning method to select the most "informative" queries, thus improving CSL-Stream's performance.

To develop such a concurrent stream mining framework, there are three main technical challenges. First, as multiple mining tasks are involved, CSL-Stream needs to maintain a larger amount of information extracted from the data stream. Second, effective and efficient operations on the data must be developed to avoid losing information. Third, a semi-learning approach is needed to cope with unlabeled data; in real-time applications, 100% pre-labeling of data is not generally be available due to expensive labeling costs. Overcoming these challenges and achieving impressive experimental results, we hope that CSL-Stream will inspire a new research direction in understanding latent commonalities among various data mining tasks in order to determine the degree of concurrency possible among them; this requires further research to derive data mining primitives which represent the entire space of data mining tasks.

In summary, our contributions are as follows:

- We propose a new algorithm to perform concurrent semi-supervised learning on data streams. We have proven that concurrent mining achieves better results by exploiting the synergies between related mining tasks.

- We integrate the dynamic synopsis tree with class profile vectors so that CSL-Stream can work well with unlabeled data.

- We deploy an incremental approach that can provide the clustering and classification results instantly. We also introduce advanced techniques to keep the space and time complexity low with theoretical bounds and empirically evaluations.

- We further enhance CSL-Stream with an active learning method so that it can work well with very sparsely labeled datasets.

## 5.2 Proposed Method

Figure 5.1 gives an overview of our novel concurrent mining approach. CSL-Stream stores a dynamic tree structure to capture the entire multi-dimensional space as well as a statistical synopsis of the data stream. Unlike static grids in D-Stream [81] that use much memory, our dynamic tree structure requires far less storage because unnecessary nodes are pruned and sibling nodes are merged where necessary. We apply a fading model to deal with concept drifts; the properties of a tree node will *decrease* according to how long the node has not been updated.

The two mining tasks of CSL-Stream, semi-supervised clustering and semi-supervised classification, are concurrently performed. They leverage on each other in terms of improving accuracy and speed; the clustering process takes into account class labels to produce high-quality clusters; the classification process uses clustering models together with a statistical test to achieve high accuracy and low running time.

CSL-Stream also employs an *incremental learning* approach. The system only needs to be updated whenever it becomes unstable. To check the system's stability, we select labeled data during each time interval $t_p$ and test them with the classification model. If the accuracy is too low, we will incrementally update the clustering models.

### 5.2.1 Dynamic Tree Structure

We assume infinite data streams $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}, \ldots]$ as input in the framework, where $\mathbf{x_t} = [\mathbf{f_t^1}, \mathbf{f_t^2}, \ldots, \mathbf{f_t^d}]^\mathbf{T}$ is a $d$-dimensional vector arriving at a time stamp $t$. We assume one record $\mathbf{x_t}$ arrives at each time stamp $t$ and has a class label $\mathbf{y_t}$ ($0 \leq y_t \leq L; y_t = 0$ if $e$ is unlabeled). Let $\mathbf{S}$ is a $d$-dimensional hyper-space formed by the data streams $X$.

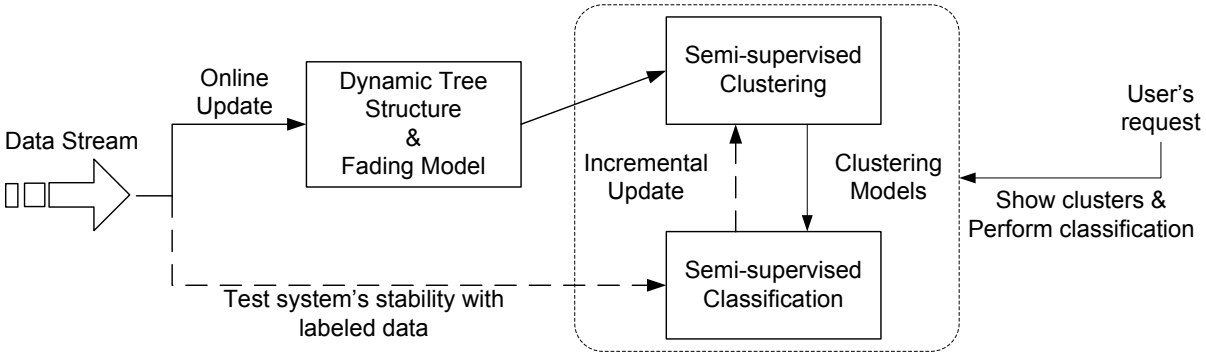



Figure 5.1: Overview of CSL-Stream

We construct a dynamic tree structure to capture the entire space $S$ with different levels of granularities as follows. Initially, a tree root at level 0 is created to hold the entire space $S$. Then for a tree node at level $h$, we partition each dimension into two equal-sized portions, and the tree node is partitioned into $2^d$ child nodes at level $h+1$. We chose 2 as it provides sufficient granularity with reasonable storage requirements. This partitioning process terminates when it reaches a pre-defined maximum tree height $H$. The tree root contains the overall information of the entire space $S$. Each tree node at height $h$ stores information of its subspace at granularity $h$. A tree node is only created if there are some instances belonging to it.

Figure 5.2 is an example of the tree structure with $d = 2$, $H = 2$. Figure 5.2(a) is a 2-dimensional data streams whose each dimension has a range [0-1]. Each data instance has a transaction identifier, two coordinates $[f_1, f_2]$, a class label (1, 2, or 0 if unlabeled), and a time stamp. The tree root is created at level $h = 0$, then it is divided into four $(2^d = 2^2 = 4)$ child nodes at level $h = 1$. Again, a tree node at level 1 is partitioned into four child nodes at level $h = 2$, and the partitioning process stops. Then, a tree node stores information of data instances belonging to its space. For example, in Figure 5.2(b), the data instance $x_2$ with coordinates $[0.6, 0.2]$ is stored in the leaf node $B$ at the bottom level $h = 2$.

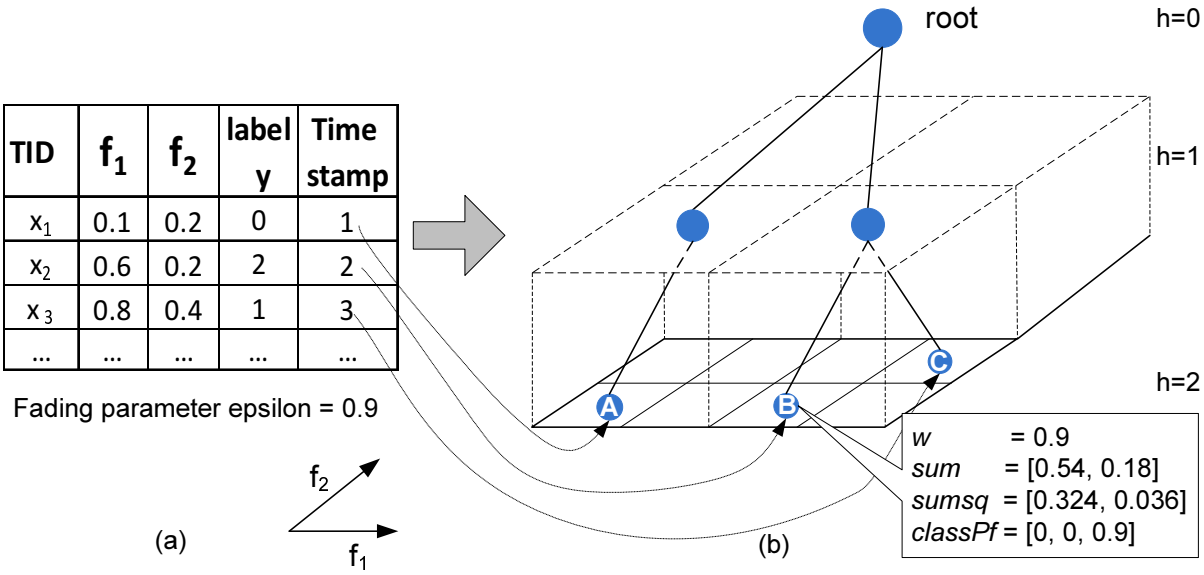



Figure 5.2: Synopsis tree for the 2-dimensional data stream with $H = 2$

#### 5.2.1.1 Node Storage

Suppose a tree node $N$ receives $m$ data instances $x_1, x_2, \ldots, x_m$. The following information will be stored in the tree node $N$:

(i) Node weight $w$ is the sum of weights of all data instances, $w = \sum_{i=1}^{m} w_{x_i}$. The weight $w_{x_i}$ of an instance $x_i$ is set to 1 at its arriving time, and then decreased over time to reflect its diminishing effect on the mining processes. Details of the weighting scheme are defined in the next section.

(ii) A $d$-dimensional vector $sum$ is the weighted sum of the coordinates of all data instances, $sum = \sum_{i=1}^{m} w_{x_i} \times x_i$.

(iii) A $d$-dimensional vector $sumsq$ is the weighted sum of the squared coordinates of all data instances, $sumsq = \sum_{i=1}^{m} w_{x_i} \times (x_i)^2$

(iv) A $(L+1)$-dimensional vector $classPf$ stores the class profile of the node. The $l^{th}$ element is the sum of weights of all data instances that have label $l$. Here, we add one more element for unlabeled instances.

By storing the above information, we are able to derive the mean $\mu = sum/w$ and variance $\sigma = \sqrt{sumsq/w - (sum/w)^2}$ to describe the statistical class boundary of a node. A change of the class profile also reflects a change in class distribution.

#### 5.2.1.2 Handling Concept Drift

Concept drift occurs in a data stream when there are changes in the underlying data distribution over time. There are two major approaches to handle concept drifts for data streams: (1) *instance selection*, and (2) *instance weighting*. The instance selection approach uses a sliding window to select recently arrived instances while ignoring older instances [162]. In the instance weighting approach, data instances can be weighted according to their ages with regard to the current concept, e.g. fading model [81].

In CSL-Stream, we apply a fading model with an exponential function to set the weight of a data instance $x$: $w_x = f(\Delta t) = \lambda^{\Delta t}$, where $0 < \lambda < 1$, and $\Delta t$ is the time gap between the current time and the arriving time of the instance $x$. Figure 5.2(b) illustrates the fading model. Let us assume that the current time is 3, and the fading parameter $\lambda$ is 0.9. The instance $x_2$ that came at time stamp 2 has a weight of $w_{x_2} = \lambda^{(3-2)} = 0.9^1 = 0.9$. As the tree node $B$ only receives a data instance $x_2$, the node $B$ stores the following information: $w = 0.9$, $sum = 0.9 \times [0.6, 0.2] = [0.54, 0.18]$, $sumsq = 0.9 \times [0.6^2, 0.2^2] = 0.9 \times [0.36, 0.04] = [0.324, 0.036]$, and $classPf = [0, 0, 0.9]$ as the instance $x_2$ has 2 as its class label .

### 5.2.2 Online Update

Given an instance $x_t$, this online-update process starts from the root and propagates throughout the tree until the maximum height $H$ is reached. At each height $h$, we update the node $N$ that contains $e$. Then, we move down to search for the child node of $N$ that contains $x_t$ and update it. If no such child node exists, we create a new node. Using a hash technique with node pointers, the complexity of searching for the correct

**Algorithm 6** Online Update

```
1: initialize tree T
2: while data stream is active do
3:    read next data point x_t with label y_t
4:    N ← T.root
5:    while N.height ≤ H do
6:       Δt ← t − C.lastUpdateTime
7:       N.w ← N.w×f(Δt)+1
8:       N.sum ← N.sum×f(Δt)+x_t
9:       N.sumsq ← N.sumsq×f(Δt)+(x_t)^2
10:      N.classPf ← N.classPf×f(Δt)
11:      N.classPf[y_t] ← N.classPf[y_t] +1
12:      N ← N.getChild(e)
13:      if N is NULL then
14:         addSubNode(N)
15:      end if
16:   end while
17:   if t = t_p then
18:      Semi-Clustering()
19:   else if (t mod t_p = 0) then
20:      PruneTree()
21:      Incremental_Update()
22:   else if (user_request = true) then
23:      Show clustering results
24:      Semi-Classification()
25:   end if
26: end while
```

child node at any tree level is $O(1)$, so the complexity of tree updating for each new data instance approaches $O(H)$, which is constant time in the stream environment.

As shown in Algorithm 6, the online-update process consists of two parts:

(i) *Top-down updating* of the tree node's information whenever a new instance arrives (lines 6-16): Firstly, the time gap is calculated. All historical information is faded with the exponential function $f(\triangle t)$. The weight and element $i$-th of class profile are incremented by 1, while the *sum* vector and *sumsq* vector are added by the coordinates and the squares of the coordinates respectively.

(ii) *Periodical pruning* of the tree to conserve memory and accelerate the mining process (line 19): For each time interval $t_p = \lfloor \log_\lambda(\alpha_L/\alpha_H) \rfloor$, which is the minimum time required for a node's density to be changed according to Lemma 4, CSL-Stream searches the tree and deletes all sparse nodes ($\alpha_L, \alpha_H$ to be defined in the next section). Next, it searches the tree to find a parent node whose all child nodes are dense. Then, all child nodes are merged and deleted, and the parent node contains the sum of their properties.

### 5.2.3 Concurrent Semi-Supervised Learning

#### 5.2.3.1 Semi-supervised Clustering

We use the node weight to define different types of nodes: dense, transitional and sparse nodes. The reachability property of dense nodes is also defined.

**Node Density**: The density of a node $N$ is a ratio of its weight to its hyper-volume (product of $d$-dimensional lengths). Suppose the volume of the entire data space $S$ is $V(S)$ and the tree node $N$ is at the height $h$, the volume of the node $N$ is $V(N) = V(S)/2^{dh}$. According to Lemma 3, the upper bound of total weight of the synopsis tree is $lim_{t\to\infty} w_{total-tree} = \frac{1}{1-\lambda}$. Thus, the average density is defined as $lim_{t\to\infty} \frac{w_{total-tree}}{V(S)} = \frac{1}{(1-\lambda)V(S)}$.

We define a node to be *dense* if its density is $\alpha_H$ times greater than the average density. Therefore, the dense condition can be written as:

$$\begin{aligned} \frac{w}{V(N)} = \frac{w}{V(S)/2^{dh}} &\geq \alpha_H * \frac{1}{(1-\lambda)V(S)} \\ w &\geq \alpha_H * \frac{V(S)}{2^{dh}} * \frac{1}{(1-\lambda)V(S)} \\ w &\geq \alpha_H * \frac{1}{2^{dh}(1-\lambda)} = D_H. \end{aligned} \tag{5.1}$$

Similarly, a node $N$ is a sparse node if its density is $\alpha_L$ times less than the average density:

$$w \leq \alpha_L * \frac{1}{2^{dh}(1-\lambda)} = D_L. \tag{5.2}$$

And a node is defined as a transitional node if the density of the node is between $D_H$ and $D_L$, i.e., $D_L < w < D_H$.

**Node Neighborhood**: Two nodes are *neighbors* if their minimum distance (single-link distance) is less than $\frac{\delta}{2^H}$, where $\delta$ is a neighbor range parameter. A node $N_p$ is *reachable* from $N_q$ if there exists a chain of nodes $N_1, \ldots, N_i, N_{i+1}, \ldots, N_m$, $N_q = N_1$ and

**Algorithm 7** Semi-Clustering

**Input:**
A list of labeled dense nodes: $LDSet$
A list of unlabeled dense nodes: $UDSet$

1: Create a cluster for each node in $LDSet$.
2: Extend these clusters sequentially with neighbor nodes in $UDSet$.
3: Merge any 2 clusters if they have same labels.
4: Perform DBSCAN clustering for the remaining nodes in $UDSet$
5: Remove clusters whose weights are less than $\epsilon$

$N_p = N_m$; such that each pair ($N_i$, $N_{i+1}$) are neighbors. A *cluster* is defined as a set of density nodes where any pair of nodes is reachable.

The semi-supervised clustering of CSL-Stream is performed according to Algorithm 7. Initially, the algorithm traverses the tree to get a list of all dense nodes. Then, this list is divided into two sets: a set of labeled dense nodes *LDSet* and a set of unlabeled dense nodes *UDSet*. Firstly, we create a cluster for each labeled node (line 1). Next, we extend these clusters step by step with their neighbor nodes. If the neighbor node does not belong to any cluster, we put it into the current cluster (line 2). If the neighbor node belongs to a cluster, we will merge these two clusters if they have the same label (line 3). Then, we continue to build clusters for the remaining nodes in *UDSet*. We perform DBSCAN [163] clustering to find the maximum set of remaining reachable unlabeled dense nodes in the *UDSet* (lines 4). Finally, we remove clusters whose weights are less than a threshold value, $\epsilon$; these clusters are considered as noise.

#### 5.2.3.2 Semi-supervised Classification

Details of the semi-supervised classification of CSL-Stream are given in Algorithm 8. The input to Algorithm 8 is a list of clusters. Each cluster in the list has a class profile vector computed by the sum of class profiles of all its member nodes. For each testing instance, we find the closest cluster and then do a statistical check to decide whether the distance between the instance and the cluster's center is acceptable. The adequate range

**Algorithm 8** Semi-Classification

**Input:** A test set: *testSet*
A list of clusters: *listCluster*
A statistical range: $\theta$

1: **while** *testSet* is not empty **do**
2: $x_i \leftarrow$ *testSet*.removeFirst()
3: Select the closet cluster, $\hat{C}$.
4: *stDv*$\leftarrow$ $\hat{C}$.getDeviation()
5: *mean*$\leftarrow$ $\hat{C}$.getMean()
6: *dist*$\leftarrow$calculateDistance($x_i$,*mean*)
7: **if** $dist < \theta * stDv$ **then**
8: $x_i$.clusterID $\leftarrow$ $\hat{C}$.getClusterID()
9: Label $y_i \leftarrow$ $\hat{C}$.getDominantClass()
10: **else**
11: Set $x_i$ as noise.
12: **end if**
13: **end while**

**Algorithm 9** Incremental Update

**Input:** A list of historical clusters: *listClusters*
Accuracy threshold: *threshold*
**Output:** A list of clusters

1: **if** $accuracy \leq threshold$ **then**
2: **for all** cluster $\hat{C}$ in *listCluster* **do**
3: $\hat{C}$.removeSparseNodes()
4: $\hat{C}$.checkConsistence()
5: **end for**
6: Traverse the tree to get a list $\beta_{dense}$ of *isolated* dense nodes.
7: Separate $\beta_{dense}$ into two sets of labeled and unlabeled nodes, $LDSet$ and $UDSet$.
8: Semi-Clustering($LDSet$,$UDSet$);
9: **end if**

is less than $\theta$ times of the cluster's standard deviation (line 7). The cluster's center and deviation are computed accordingly $\mu = \frac{sum}{w}, \sigma = \sqrt{\frac{sumsq}{w} - \left(\frac{sum}{w}\right)^2}$. If the distance is acceptable, the dominant class of the cluster will be assigned to the testing sample. Else, the instance will be considered as noise.

#### 5.2.3.3 Incremental Update

CSL-Stream is an incremental algorithm; it can detect and update its learning models incrementally whenever a concept drift occurs.

For each time interval $t_p$ (Algorithm 6, line 21), we select labeled instances and test them with the classification model to check the system's stability. If the accuracy is greater than a predefined threshold value (experiments reveal that 0.9 is an appropriate

value); the model remains stable, and we can skip the remaining steps. Else, we need to refine the learning models. CSL-Stream begins to fine-tune the historical clustering model by checking the consistency of each cluster. The algorithm removes sparse nodes, that are dense at the previous learning phase, and splits the cluster if it becomes unconnected or has low purity.

Then, CSL-Stream traverses the tree to get a list of isolated dense nodes $\beta_{dense}$ (line 6). After separating $\beta_{dense}$ into two sets of labeled and unlabeled dense nodes, it performs the semi-supervised clustering method to derive new arriving clusters. As the procedure reuses the historical clustering results, the size of $\beta_{dense}$ is relatively small. Thus, the running time of the clustering process is reduced significantly.

#### 5.2.3.4 Example

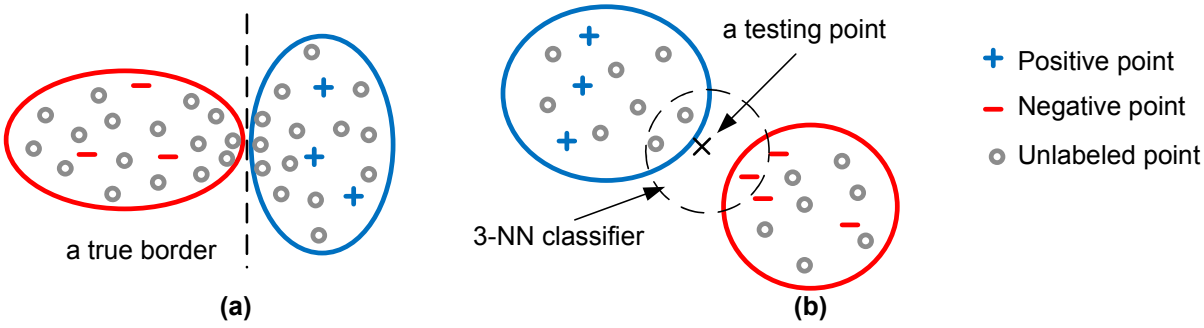



Figure 5.3: Examples of the benefits of concurrent semi-learning

Our concurrent approach maximizes computational resources and exploits advantages of the interplay between clustering and classification. CSL-Stream's semi-supervised clustering takes care of data labels by continuously updating the class profile array of each node. It ensures that two clusters are merged only if they have the same dominant class, finds the best borders among clusters. For example, in Figure 5.3 (a), two clusters with different class labels share the same border. A stand-alone clustering algorithm will probably incorrectly consider them as a single cluster. For example, D-Stream will

merge the two clusters as it views the data points at the border as connected due to their proximity to one another [81]. CSL-Stream does not merge them as they have different dominant classes, and thus CSL-Stream correctly considers them as two clusters.

By disregarding unlabeled data, a stand-alone classification method may not capture the classes' distribution. Using clustering results with unlabeled data, CSL-Stream can gain knowledge of the classes' distributions and improve its accuracy. For example, in Figure 5.3 (b), a 3-NN classifier wrongly classifies the testing point because most its neighbors are unlabeled. CSL-Stream selects the label of the nearest cluster, and correctly classifies the testing point as positive.

### 5.2.4 Enhancing Concurrent Learning with Active Learning

In many applications, totally labeled training instances are rare because it is time-consuming and difficult to get experts to label instances. Moreover, instances with incorrect labels may significantly degrade the performance of the model. Active learning or "query learning" attempts to address this problem by asking domain experts to selectively label a small number of unlabeled instances only. It aims to achieve an accurate predictive model using as few labeled instances as possible, therefore, minimizing the labeling cost. There are three main settings where queries can be generated: membership query synthesis, stream-based sampling, and pool-based sampling. In the query synthesis setting, the learner may ask for labels of any unlabeled instances in the input space. A limitation of this setting is that such arbitrary queries can be awkward to human experts, for example, active learning works poorly when a human is used to classify handwritten characters [164]. The stream-based setting is sometimes called sequential active learning, as when each unlabeled instance comes, the learner must decide whether to query it or not. A typical method is to define a region of uncertainty and to only query instances that fall within this region. This approach is computationally expensive, as the region

must be updated after each query. The pool-based sampling approach is motivated by a scenario where there is a large pool of unlabeled data and only a small set of labeled data [165]. It usually deploys a greedy method together with an *informativeness* measure to evaluate all instances in the pool.

Here, we choose to apply the pool-based sampling setting as it can provide the best queries after collecting and evaluating the entire collection of unlabeled data. We further propose an effective heuristic to select the most important query. Particularly, we need to check and generate a query with two following cases:

- *Unlabeled clusters*: A cluster is considered as an unlabeled cluster if it contains no labeled data instance. In this case, the cluster's center representing the whole cluster is the most important data instance, so we select the center to query for a label. Then, the label of the cluster is set to the suggested label.

- *Skewed labeled clusters*: A cluster's label is skewed if its labeled data instances only occupy one side of the cluster. As we are uncertain about the other side of the cluster, we choose the farthest instance from the other side to query for a label. If the cluster's label differs from the suggested label, we need to internally re-cluster this cluster.

Figure 5.4 shows an illustration of our active learning method. CSL-Stream processes the synopsis of data streams, which are represented by a dynamic tree structure. Hence, it may cause awkward problems to human experts if we simply query the center of a tree node. To overcome this problem, we attach each tree node to a representative data instance, and this instance is updated whenever a new sample arrives in the corresponding tree node. Details of our active learning method are given in Algorithm 10. As the number of queries is limited, we want to perform active learning with clusters having bigger weights first. Hence, we sort clusters according to their weights in descending

**Algorithm 10** Active Learning

**Input:** A list of clusters: *listCluster*
Maximum number of queries: max-query

```
1: Sort the list listCluster according to clusters' weights in descending order.
2: for all cluster C in the list of clusters listCluster do
3:    if number of query > max − query then
4:       Exit.
5:    else
6:       if C is unlabeled then
7:          Select the tree node N that contains the cluster's center.
8:          Get the representative point p_rep of the tree node N.
9:          Query for the label of p_rep and set it as the dominant class label of C.
10:      else if C is skewed labeled then
11:         Get the "informative" node N that is farthest from labeled nodes of C.
12:         Query for the label of the representative point p_rep of N.
13:         if p_rep.class ≠ C.getDominantClass() then
14:            Perform Semi-Clustering() for cluster C.
15:         end if
16:      end if
17:   end if
18: end for
```

order (line 1). Then, we check all the clusters and generate the most informative queries if the number is less than or equal to a maximum value (lines 2-18).

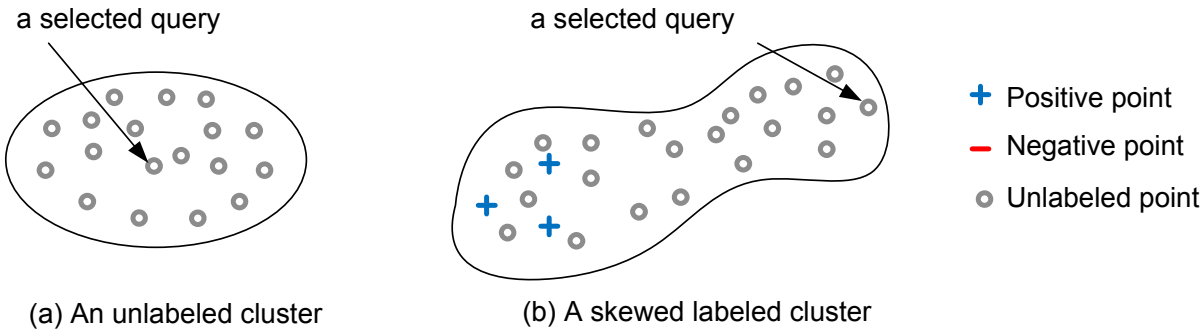



Figure 5.4: Enhancing Concurrent Learning with Active Learning

### 5.2.5 Theoretical Proofs

**Lemma 3** *The total weight of the synopsis tree is less than or equal to* $\frac{1}{1-\lambda}$.

**Proof:** The total weight of the synopsis tree is the sum of the weights of all its tree nodes. Moreover, the weight of a tree node equals to the sum of data instances belonging to the node's subspace. Hence, given a current time $t$, the tree weight is the sum of weights of data instances that have arrived during the time gap $[0, 1, \ldots, t]$. As we apply the fading model, the weight of data instance arriving at time instance $t' \in [0, t]$ is: $f(\Delta t) = \lambda^{\Delta t} = \lambda^{t-t'}$. Thus, the total weight of the synopsis tree is:

$$W_{total-tree} = \sum_{t'=0}^{t} \lambda^{t-t'} = \frac{1-\lambda^{t+1}}{1-\lambda} \leq \frac{1}{1-\lambda}.$$

Alternatively, we can derive that $lim_{t\to\infty} W_{total-tree} = \frac{1}{1-\lambda}$.

**Lemma 4** *The minimum time for a tree node to change from dense to sparse is* $t_p = \lfloor \log_\lambda(\alpha_L/\alpha_H) \rfloor$.

**Proof:** Let us assume that a tree node $N$ is sparse at a time instance $t_1$. It then becomes sparse at a time instance $t_2$, $(t_1 < t_2)$. According to the definition of a dense node in Equation 5.1, the tree node's weight at time $t_1$ is:

$$w_1 \geq \alpha_H * \frac{1}{2^{dh}(1-\lambda)}.$$

Similarly, according to the definition of a sparse node in Equation 5.2, the tree node's weight at time $t_2$ is:

$$w_2 \leq \alpha_L * \frac{1}{2^{dh}(1-\lambda)}.$$

We have:

$$\begin{aligned}
w_2 &\geq w_1 * f(t_2 - t_1) = w_1 * \lambda^{(t_2-t_1)} \\
\alpha_L * \frac{1}{2^{dh}(1-\lambda)} &\geq \alpha_H * \frac{1}{2^{dh}(1-\lambda)} * \lambda^{(t_2-t_1)} \\
\alpha_L &\geq \alpha_H * \lambda^{(t_2-t_1)} \\
\alpha_L/\alpha_H &\geq \lambda^{(t_2-t_1)} \\
t_p = \lfloor \log_\lambda(\alpha_L/\alpha_H) \rfloor &\leq (t_2 - t_1).
\end{aligned}$$

#### 5.2.5.1 Space Complexity

**Lemma 5** *If the density threshold to remove sparse node is $D_L$, then the maximum number of nodes in the synopsis tree is $N_{max-node} \leq \lceil \frac{\log D_L}{log\lambda} \rceil = O(dH)$*

**Proof:** In the worst case, we assume that every data point arrives at a different node and no merging is performed. When a node receives a data point, its weight is set to 1. We define $t_{max}$ as the time needed for this node to become sparse with its weight $\leq D_L$, and is removed. After $t_{max}$ time, whenever a data point arrives, a new tree node created, and another one is deleted. This means that $t_{max}$ is the maximum number of tree nodes. We have:

$$\lambda^{t_{max}} = D_L \quad \Rightarrow \quad t_{max} \log \lambda = \log D_L \quad \Rightarrow \quad t_{max} \leq \lceil \frac{\log D_L}{\log \lambda} \rceil.$$

We also know that: $D_L = \alpha_L * \frac{1}{2^{dh}(1-\lambda)}$. Then,

$$\Rightarrow \quad t_{max} \leq \lceil \log_\lambda \alpha_L - \log_\lambda (1-\lambda) - dH \log_\lambda 2 \rceil \quad = O(dH).$$

#### 5.2.5.2 Time Complexity

**Lemma 6** *The complexity of CSL-Stream is $O(n(dH + L))$, where $n$ is the number of instances of the data stream and $L$ is the number of classes.*

**Proof:** The proposed CSL-Stream has three main operations: online updating, clustering and classification. The online updating consists of top-down updating and periodical pruning. The top-down updating only requires a constant time to update a new data instance and its complexity is $O(nH)$. The running time of the pruning process depends on the number of tree nodes. In the previous section, we prove that the upper bound of the number of tree nodes is $O(dH)$, so the pruning process's complexity is $O(ndH)$.

The complexity of the clustering depends on its implementation. If we use R*-trees to implement the synopsis tree, the searching time for neighbor nodes is $O(log(dH))$. And the complexity of clustering operation is $O(dHlog(dH))$. The running time of the

Table 5.1: Characteristics of datasets used for evaluation.

| **Name** | **#Instances** | **#Attributes** | **#Classes** |
|---|---|---|---|
| RBF | 100,000 | 10 | 5 |
| HYP | 100,000 | 10 | 2 |
| LED | 100,000 | 24 | 3 |
| SHUTTLE | 58,000 | 9 | 7 |
| KDD'99 | 494,022 | 34 | 5 |
| COVERTYPE | 581,012 | 54 | 7 |

classification operation depends on the number of clusters, and this number is usually a multiple of the number of classes. Thus, the classification time can be expected to be $O(nL)$, where $L$ is the number of classes.

In summary, the complexity of the algorithm is: $O(nH + ndH + dHlog(dH) + nL) = O(n(dH + L))$

## 5.3 Experiments and Analysis

### 5.3.1 Experimental Setup

We use both synthetic and real datasets in our experiments to demonstrate the practicality of our approach. Three synthetic datasets, Random RBF generator (RBF), Rotating Hyperplane (HYP), and LED dataset (LED), are generated from the MOA framework [160]. Concept drifts are generated by moving 10 centroids at a rate of 0.001 per instance in the RBF dataset, and changing 10 attributes at a rate of 0.001 per instance in the HYP dataset. We also use the three largest real datasets in the UCI machine learning repository: SHUTTLE, KDD'99, and Forest Covertype [166]. Table 5.1 shows the characteristics of the six datasets.

In order to illustrate the beneficial mutual relationship between clustering and classification, we created two variants of CSL-Stream that do not exploit this relationship as follows:

- *Alone-Clustering:* This variant of CSL-Stream does not consider class labels while clustering.

- *Alone-Classification:* This variant of CSL-Stream finds the nearest tree node to the testing instances and classifies them to the dominant class of the tree node.

We compare the performance of CSL-Stream to its stand-alone variants, D-Stream [81], and SmSCluster [133]. The experiments were conducted on a Windows PC with a Pentium D 3GHz Intel processor and 2GB memory.

**Parameter Setting**

Table 5.2: Parameter Table

| Parameter | Description | Recommended Value |
|---|---|---|
| $\lambda$ | fading parameter | [0.99,1] |
| $\alpha_H$ | dense ratio threshold | 3 |
| $\alpha_L$ | sparse ratio threshold | 0.8 |
| H | the maximum trees height | [5,10] |
| $\epsilon$ | noise threshold | [1,3] |
| $\delta$ | neighborhood range | [0,2] |
| $\theta$ | statistical range | [3,6] |

Table 5.2 shows seven parameters that are used in CSL-Stream and their recommended values. The first three parameters, which do not greatly affect results, are fixed as follows: the fading factor $\lambda$ is set to 0.998 as recommended in D-Stream [81], the dense ratio threshold $\alpha_H$ is set to 3.0, and the sparse ratio threshold $\alpha_L$ is set to 0.8. For D-Stream and SmSCluster, we use the published default parameter configurations. The chunk size *chunk-size* is set to 1000 for all datasets. For CSL-Stream, the other parameters of CSL-Stream are set as follows: maximum tree's height $H = 5$, the noise threshold $\epsilon = 1$, the neighborhood range $\delta = 2$, and the statistical range factor $\theta = 3$. The scalability and sensitivity of these parameters are evaluated in the next section.

We simulate the data stream environment with an *Interleaved-Chunk* setting. Data is fed into many equal-size chunks. Each new data chunk is first used to test the existing model, then is used to update the model. We run the experiments 10 times for each dataset and summarize their running time and performance with their means and standard deviations in Tables 5.3 and 5.3.3.

### 5.3.2 Clustering Evaluation

Here, we use the BCubed measure to evaluate clustering results in our experiments [167]. It computes the average correctness over the entire dataset and has been proven to be more useful than the purity, inverse purity, entropy, and F-measures.

We conducted experiments with the 100%-labeled datasets to assess the clustering results of CSL-Stream, Alone-Clustering and D-Stream. Table 5.3 shows the running time as well as B-Cubed comparisons among the three clustering methods.

$$Bcubed_P = Avg_x[Avg_{x'.C(x)=C(x')}[Correctness(x, x')]]$$
$$Bcubed_R = Avg_x[Avg_{x'.L(x)=L(x')}[Correctness(x, x')]],$$

where $C$ denotes the set of clusters, $L$ denotes the set of classes, and $Correctness(x, x') = 1$ if $x$ and $x'$ have the same class label and the same cluster, otherwise it is 0. Bcubed is the F-combination of its precision and its recall:

$$Bcubed = \frac{1}{\alpha(\frac{1}{Bcubed_P}) + (1-\alpha)(\frac{1}{Bcubed_R})}$$

Table 5.3: Comparison among CSL-Stream, Alone-Clustering and D-Stream with B-Cubed measure. Time is measured in seconds. For each dataset, the lowest running time is underlined, and the highest B-Cubed value is **boldfaced**.

| | **CSL-Stream** | | **Alone-Clustering** | | **D-Stream** | |
|---|---|---|---|---|---|---|
| | Time | B-Cubed | Time | B-Cubed | Time | B-Cubed |
| RBF(10,0.001) | $\underline{10.64_{\pm 1.56}}$ | $\mathbf{37.67_{\pm 4.93}}$ | $17.22_{\pm 1.72}$ | $36.27_{\pm 2.32}$ | $17.37_{\pm 1.36}$ | $17.39_{\pm 2.41}$ |
| HYP(10,0.001) | $\underline{13.37_{\pm 1.23}}$ | $\mathbf{65.24_{\pm 10.66}}$ | $23.67_{\pm 1.66}$ | $57.14_{\pm 4.01}$ | $33.37_{\pm 1.46}$ | $55.54_{\pm 4.66}$ |
| LED | $\underline{53.9_{\pm 1.68}}$ | $\mathbf{68.38_{\pm 11.74}}$ | $205.61_{\pm 2.34}$ | $19.1_{\pm 0.58}$ | $203.8_{\pm 2.94}$ | $19.3_{\pm 0.63}$ |
| SHUTTLE | $\underline{1.37_{\pm 0.16}}$ | $\mathbf{93.46_{\pm 1.04}}$ | $1.37_{\pm 0.16}$ | $89.07_{\pm 1.83}$ | $1.45_{\pm 0.17}$ | $88.1_{\pm 0.93}$ |
| KDD'99 | $39.78_{\pm 0.62}$ | $\mathbf{76.89_{\pm 27.83}}$ | $\underline{38.68_{\pm 0.84}}$ | $76.88_{\pm 27.83}$ | $53.79_{\pm 1.06}$ | $73.5_{\pm 31.24}$ |
| COVERTYPE | $\underline{130.24_{\pm 1.87}}$ | $26.54_{\pm 9.68}$ | $212.07_{\pm 2.45}$ | $\mathbf{35.11_{\pm 10.79}}$ | $152.46_{\pm 2.67}$ | $12.55_{\pm 5.8}$ |

We observe that CSL-Stream achieves the lowest running time and the highest B-Cubed value for many datasets. Unlike Alone-Clustering and D-Stream, CSL-Stream takes into account the class labels during clustering and guarantees that no two clusters

with the same label are merged. Thus, it achieves better B-Cubed results than Alone-Clustering and D-Stream. Moreover, using the dynamic tree structure, CSL-Stream can adapt quickly to concept drifts. It is also the fastest method because the pruning process helps to remove unnecessary tree nodes. Although CSL-Stream has low B-Cubed values in some datasets, e.g., RBF, HYP, and COVERTYPE as each class has many clusters in these datasets, CSL-Stream still attains high purity in these cases ($> 90\%$) as shown in Table 5.4; purity is a measure of cluster homogeneity.

Table 5.4: Comparison among CSL-Stream, Alone-Clustering and D-Stream with purity measure. Time is measured in seconds. For each dataset, the lowest running time is underlined, and the highest purity value is boldfaced.

| | **CSL-Stream** | | **Alone-Clustering** | | **D-Stream** | |
|---|---|---|---|---|---|---|
| | Time | Purity | Time | Purity | Time | Purity |
| RBF(10,0.001) | $10.64_{\pm 1.56}$ | $\mathbf{100_{\pm 0}}$ | $17.22_{\pm 1.72}$ | $48.41_{\pm 10.66}$ | $17.37_{\pm 1.36}$ | $99_{\pm 2.05}$ |
| HYP(10,0.001) | $13.37_{\pm 1.23}$ | $\mathbf{99.99_{\pm 0.03}}$ | $23.67_{\pm 1.66}$ | $67.76_{\pm 8.09}$ | $33.37_{\pm 1.46}$ | $69.63_{\pm 8.54}$ |
| LED | $53.9_{\pm 1.68}$ | $\mathbf{100_{\pm 0.01}}$ | $205.61_{\pm 2.34}$ | $15.74_{\pm 2.41}$ | $203.8_{\pm 2.94}$ | $15.9_{\pm 2.45}$ |
| SHUTTLE | $1.37_{\pm 0.16}$ | $\mathbf{98.46_{\pm 0.44}}$ | $1.37_{\pm 0.16}$ | $97.78_{\pm 2.53}$ | $1.45_{\pm 0.17}$ | $98.22_{\pm 1.27}$ |
| KDD'99 | $39.78_{\pm 0.62}$ | $\mathbf{99.99_{\pm 0.05}}$ | $38.68_{\pm 0.84}$ | $99.99_{\pm 0.07}$ | $53.79_{\pm 1.06}$ | $99.99_{\pm 0.03}$ |
| COVERTYPE | $130.24_{\pm 1.87}$ | $\mathbf{99.28_{\pm 0.53}}$ | $212.07_{\pm 2.45}$ | $79.32_{\pm 7.63}$ | $152.46_{\pm 2.67}$ | $91.98_{\pm 3.98}$ |

### 5.3.3 Classification Evaluation

Table 5.5: Comparison with fully labeled datasets among CSL-Stream, Alone-Classification and SmSCluster. Time is measured in seconds. For each dataset, the lowest running time is underlined, and the highest accuracy is **boldfaced**.

| | **CSL-Stream** | | **Alone-Classification** | | **SmSCluster** | |
|---|---|---|---|---|---|---|
| | Time | Accuracy | Time | Accuracy | Time | Accuracy |
| RBF(10,0.001) | $49.29_{\pm 2.35}$ | $\mathbf{71.57_{\pm 6.37}}$ | $53.32_{\pm 3.24}$ | $44.57_{\pm 7.18}$ | $41.45_{\pm 3.35}$ | $30.1_{\pm 12.38}$ |
| HYP(10,0.001) | $15.78_{\pm 1.68}$ | $\mathbf{87.88_{\pm 1.88}}$ | $16.17_{\pm 2.03}$ | $70.66_{\pm 2.09}$ | $40.18_{\pm 2.65}$ | $76.05_{\pm 2.61}$ |
| LED | $34.17_{\pm 2.16}$ | $\mathbf{72.73_{\pm 1.82}}$ | $98.85_{\pm 3.12}$ | $10.24_{\pm 1}$ | $85.69_{\pm 3.87}$ | $54.70_{\pm 3.45}$ |
| SHUTTLE | $2.56_{\pm 0.35}$ | $\mathbf{98.3_{\pm 0.3}}$ | $2.64_{\pm 0.91}$ | $98.28_{\pm 0.31}$ | $20.35_{\pm 2.61}$ | $97.50_{\pm 0.49}$ |
| KDD'99 | $83.06_{\pm 2.47}$ | $\mathbf{98.06_{\pm 8.29}}$ | $87.57_{\pm 2.13}$ | $98.25_{\pm 8.24}$ | $565.02_{\pm 3.87}$ | $85.33_{\pm 33.39}$ |
| COVERTYPE | $183.75_{\pm 3.05}$ | $\mathbf{81.63_{\pm 10.43}}$ | $194.41_{\pm 3.46}$ | $78.96_{\pm 9.39}$ | $320.65_{\pm 3.02}$ | $49.23_{\pm 15.42}$ |

We compare the running time and accuracy among CSL-Stream, Alone-Classification, and SmSCluster [133]. Table 5.3.3 reports the running time and accuracy comparisons

among the above classification algorithms when the datasets are fully labeled. It is found that CSL-Stream is the fastest and the most accurate algorithm for many datasets. CSL-Stream is better than Alone-Classification in terms of speed and accuracy because it exploits clustering results for classification. CSL-Stream also takes less time as it only compares the testing instances to a small number of clusters and is resistant to noise. SmSCluster suffers from a high time complexity and low accuracy with non-spherical clusters since it is based on the $k$-Means algorithm.

Table 5.6: Comparison with partially labeled datasets among CSL-Stream, Alone-Classification and SmSCluster. Time is measured in seconds. For each dataset, the lowest running time is underlined, and the highest accuracy is boldfaced.

| | **CSL-Stream** | | **Alone-Classification** | | **SmSCluster** | |
|---|---|---|---|---|---|---|
| | Time | Accuracy | Time | Accuracy | Time | Accuracy |
| RBF(10,0.001) 50% | $52.86_{\pm 2.13}$ | $\mathbf{54_{\pm 10.13}}$ | $\underline{40.36_{\pm 2.48}}$ | $26.12_{\pm 7.51}$ | $48.33_{\pm 1.43}$ | $30.47_{\pm 12.29}$ |
| RBF(10,0.001) 25% | $49.53_{\pm 2.54}$ | $\mathbf{51.15_{\pm 10.81}}$ | $\underline{38.92_{\pm 2.67}}$ | $15.47_{\pm 5.86}$ | $44.36_{\pm 1.86}$ | $27.93_{\pm 12.3}$ |
| RBF(10,0.001) 10% | $49.09_{\pm 2.01}$ | $\mathbf{48.88_{\pm 10.06}}$ | $38.56_{\pm 2.65}$ | $8.16_{\pm 4.86}$ | $\underline{30.73_{\pm 1.08}}$ | $28.43_{\pm 12.16}$ |
| HYP(10,0.001) 50% | $\underline{20.08_{\pm 1.48}}$ | $\mathbf{81.6_{\pm 3.61}}$ | $21.11_{\pm 2.16}$ | $35.97_{\pm 3.02}$ | $37.89_{\pm 2.14}$ | $74.36_{\pm 2.68}$ |
| HYP(10,0.001) 25% | $\underline{17.34_{\pm 1.32}}$ | $\mathbf{73.88_{\pm 4.25}}$ | $18.65_{\pm 2.09}$ | $18.29_{\pm 2.1}$ | $46.26_{\pm 2.09}$ | $64.36_{\pm 3.11}$ |
| HYP(10,0.001) 10% | $\underline{15.97_{\pm 1.65}}$ | $61.18_{\pm 7.15}$ | $17.47_{\pm 2.05}$ | $7.57_{\pm 1.32}$ | $31.56_{\pm 1.86}$ | $\mathbf{61.41_{\pm 4.50}}$ |
| LED 50% | $\underline{50.52_{\pm 2.41}}$ | $\mathbf{62.08_{\pm 9.33}}$ | $108.78_{\pm 2.57}$ | $6.53_{\pm 5.04}$ | $133.3_{\pm 2.19}$ | $52.14_{\pm 4.59}$ |
| LED 25% | $\underline{54.3_{\pm 2.54}}$ | $\mathbf{51.85_{\pm 5.87}}$ | $99.65_{\pm 2.88}$ | $2.73_{\pm 4.53}$ | $127.64_{\pm 2.76}$ | $47.44_{\pm 4.11}$ |
| LED 10% | $\underline{36.36_{\pm 2.87}}$ | $\mathbf{45.43_{\pm 4.07}}$ | $97.77_{\pm 2.71}$ | $0.88_{\pm 2.86}$ | $74.75_{\pm 2.77}$ | $44.93_{\pm 3.61}$ |
| SHUTTLE 50% | $2.59_{\pm 0.34}$ | $\mathbf{97.7_{\pm 2.3}}$ | $2.47_{\pm 0.63}$ | $97.54_{\pm 2.61}$ | $18.88_{\pm 1.51}$ | $97.69_{\pm 0.44}$ |
| SHUTTLE 25% | $\underline{2.5_{\pm 0.42}}$ | $\mathbf{97.5_{\pm 1.09}}$ | $\underline{2.5_{\pm 0.88}}$ | $97.31_{\pm 2.4}$ | $18.34_{\pm 1.93}$ | $97.41_{\pm 0.44}$ |
| SHUTTLE 10% | $2.48_{\pm 0.26}$ | $96.12_{\pm 2.15}$ | $\underline{2.47_{\pm 0.47}}$ | $96.71_{\pm 2.36}$ | $19_{\pm 1.91}$ | $\mathbf{97.02_{\pm 1}}$ |
| KDD'99 50% | $\underline{72.3_{\pm 2.65}}$ | $\mathbf{97.08_{\pm 8.71}}$ | $77.26_{\pm 2.35}$ | $95_{\pm 12.11}$ | $691.25_{\pm 4.26}$ | $75.33_{\pm 14.27}$ |
| KDD'99 25% | $\underline{72.56_{\pm 2.69}}$ | $\mathbf{96.49_{\pm 9.25}}$ | $78.12_{\pm 2.48}$ | $92.82_{\pm 15.47}$ | $703.56_{\pm 4.35}$ | $75.03_{\pm 14.65}$ |
| KDD'99 10% | $\underline{79.46_{\pm 2.13}}$ | $\mathbf{95.07_{\pm 11.7}}$ | $85.44_{\pm 2.49}$ | $90.63_{\pm 18.89}$ | $821.56_{\pm 4.07}$ | $75.25_{\pm 14.25}$ |
| COVERTYPE 50% | $196.34_{\pm 3.4}$ | $\mathbf{74.71_{\pm 11.66}}$ | $\underline{178.95_{\pm 3.65}}$ | $53.56_{\pm 7.86}$ | $400.62_{\pm 4.27}$ | $35.95_{\pm 14.14}$ |
| COVERTYPE 25% | $190.55_{\pm 3.57}$ | $\mathbf{72.37_{\pm 11.33}}$ | $\underline{168.12_{\pm 3.08}}$ | $38.27_{\pm 7.02}$ | $394.8_{\pm 4.65}$ | $33.08_{\pm 17.93}$ |
| COVERTYPE 10% | $184.88_{\pm 3.79}$ | $\mathbf{67.36_{\pm 11.99}}$ | $\underline{165.11_{\pm 3.49}}$ | $25.09_{\pm 5.53}$ | $350.9_{\pm 4.92}$ | $28.05_{\pm 14.61}$ |

Furthermore, we measure performances of the above algorithms when unlabeled data is present. We conduct experiments with the Hyperplane and KDD'99 datasets with different proportions of labeled data instances: *50%, 25%* and *10%*. Table 5.3.3 shows the running time and the accuracy of the above classification algorithms. We observe

that CSL-Stream and SmSCluster can work well with partially labeled data. When the percentage of labeled data decreases, the accuracy of Alone-Classification drops quickly while the accuracy of CSL-Stream and SmSCluster remains high. It is clear that CSL-Stream outperforms SmSCluster in terms of speed and accuracy. These results also show that CSL-Stream has the best overall performance. We would like to next examine our algorithm in cases where there are very few labeled instances.

### 5.3.4 Enhancing Concurrent Mining with Active Learning

In order to evaluate the effectiveness of active learning, we set the proportion of labeled data to just 1%. Moreover, our active learning methods are configured with the following thresholds of the number of queries: 1%, 0.5%, 0.2%, and 0.1%. Table 5.3.4 shows the comparisons between CSL-Stream and its variants. We observe that CSL-Stream does not work well with datasets with very few labeled samples. Although it attains good results for SHUTTLE and KDD'99 datasets, its accuracy is low for remaining ones. With active learning, CSL-Stream increases its average accuracy by *5.81%, 4.11%, 3.42%, 2.91%* with query thresholds of 1%, 0.5%, 0.2%, and 0.1% respectively. We also find that our active learning method is more effective with real datasets than synthetic datasets, which are typically generated by random processes.

The impressive experimental results show that our active learning method is able to work well with very sparsely labeled datasets by exploiting the most informative queries.

### 5.3.5 Scalability Evaluation

Section 5.2.5 gives an upper bound for the maximum number of nodes and the running time of CSL-Stream. Here, we have examined the scalability of CSL-Stream in terms of the number of nodes and the running time w.r.t dimensionality and the tree's height.

Table 5.7: Comparison with sparsely labeled datasets *(1%)* among CSL-Stream and its variant enhanced with Active Learning with different thresholds of the number of queries.

| **Dataset** | **CSL-Stream** | **CSL-Stream + Active Learning** | | | |
|---|---|---|---|---|---|
| | | #query≤1% | #query≤0.5% | #query≤0.2% | #query≤0.1% |
| RBF(10,0.001) 1% | $44.81_{\pm 12.99}$ | $51.6_{\pm 11.55}$ | $52.26_{\pm 11.21}$ | $51.18_{\pm 11.37}$ | $50.5_{\pm 11.6}$ |
| HYP(10,0.001) 1% | $69.24_{\pm 5.94}$ | $72.72_{\pm 4.41}$ | $71.76_{\pm 4.7}$ | $71.87_{\pm 4.51}$ | $71.5_{\pm 4.25}$ |
| LED 1% | $11.16_{\pm 2.6}$ | $11.83_{\pm 2.7}$ | $11.57_{\pm 3.12}$ | $11.33_{\pm 3.1}$ | $11.64_{\pm 2.69}$ |
| SHUTTLE 1% | $74.38_{\pm 11.08}$ | $87.47_{\pm 2.81}$ | $79.53_{\pm 1.44}$ | $79.41_{\pm 5.88}$ | $79.33_{\pm 4.81}$ |
| KDD'99 1% | $92.31_{\pm 14.81}$ | $94.5_{\pm 12.56}$ | $94.47_{\pm 12.38}$ | $94.32_{\pm 12.22}$ | $93.99_{\pm 13.12}$ |
| COVERTYPE 1% | $59.26_{\pm 12.22}$ | $67.92_{\pm 12.34}$ | $66.24_{\pm 12.63}$ | $63.6_{\pm 12.08}$ | $61.67_{\pm 12.3}$ |
| *Average* | $58.53_{\pm 9.94}$ | $64.34_{\pm 7.73}$ | $62.64_{\pm 7.58}$ | $61.95_{\pm 8.19}$ | $61.44_{\pm 8.13}$ |
| **Average Improvement** | | **5.81** | **4.11** | **3.42** | **2.91** |

We conduct experiments with the KDD'99 dataset to assess the scalability w.r.t dimensionality $d$. We set the maximum tree height to 5, and increase $d$ from 5 to 34 with unit steps. With Figures 5.5 and 5.6, we can clearly see that the maximum number of nodes and the running time of CSL-Stream increase linearly as dimensionality increases.

To evaluate the scalability w.r.t the maximum tree height $H$, we select the first 30K data instances of KDD'99 dataset and set the dimensionality to 34. We increase $H$ from 3 to 10 with unit steps. Figures 5.7 and 5.8 show that the maximum number of nodes and the running time of CSL-Stream increase linearly as $H$ increases.

### 5.3.6 Sensitivity Analysis

In the experiments, we set the neighborhood range $\delta = 2$, the noise threshold $\epsilon = 1$, and the statistical range factor $\theta = 3$. We test the sensitivity of $\delta$, $\epsilon$, and $\theta$ with the KDD'99 dataset. We add 10% of random noise to test the sensitivity of $\theta$. Figure 5.9 shows the Bcubed values and accuracy of CSL-Stream with different parameters.

We observe that CSL-Stream is insensitive to the neighborhood range as the KDD'99 dataset is high-dimensional. When the noise threshold increases, CSL-Stream's accuracy decreases as some important clusters are ignored. Our algorithm is also sensitive to the statistical range factor, and it should be set between two to six to achieve good accuracy.

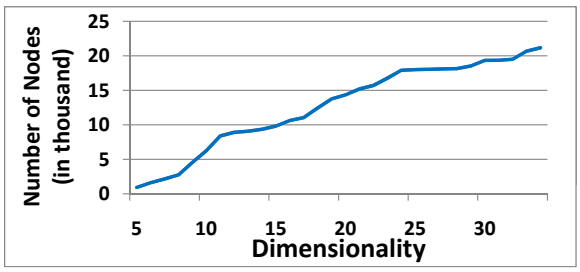



Figure 5.5: Number of nodes vs. dimensionality.

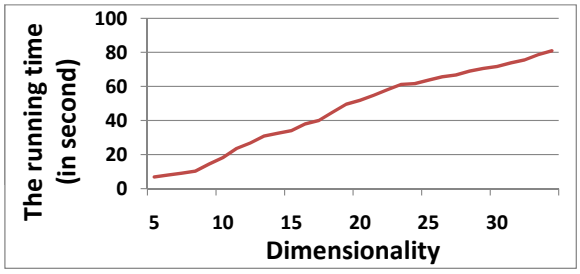



Figure 5.6: Running time vs. dimensionality.


Figure 5.7: Number of nodes vs. tree height.

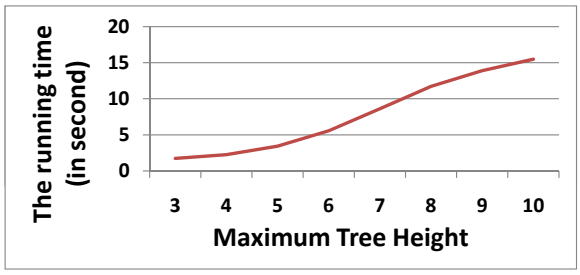



Figure 5.8: Running time vs. tree height.

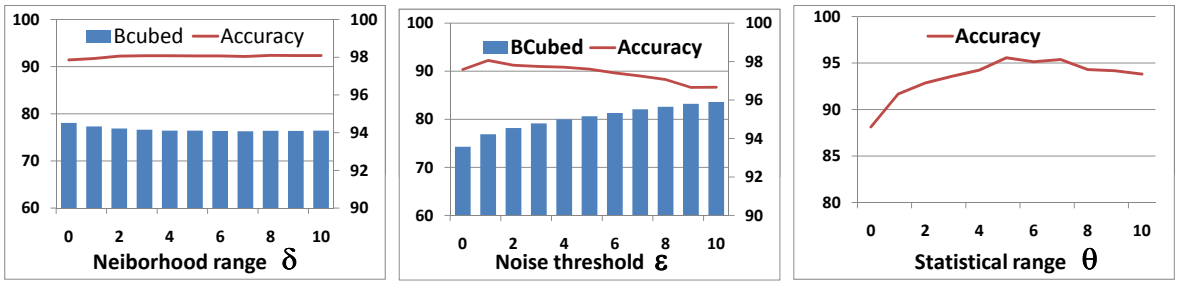



Figure 5.9: Sensitivity analysis.

## 5.4 Summary

We have studied the new problem of concurrent mining for data streams and proposed CSL-Stream to perform semi-supervised classification and clustering at the same time. We have conducted extensive experiments to show that CSL-Stream outperforms state-of-the-art data stream algorithms in terms of speed, accuracy and scalability. Experimental results also showed that it can produce mining results in a real-time manner, making

it suitable for online applications. Furthermore, we have enhanced CSL-Stream with a novel active learning method so that it can achieve excellent results even with sparsely labeled datasets. Therefore, our algorithm is very useful in many real applications, where a domain expert is typically only able to handle a very small number of queries.

*"I can accept failure, everyone fails at something.*
*But I can't accept not trying."*
MICHAEL JORDAN (1963-) Athlete

*"What keeps me going is goals."*
MUHAMMAD ALI (1942-) Boxer

# 6 Conclusions and Future Work

This final chapter of the dissertation restates the research problem and reviews the major methods used. The sections here summarize the experimental results, discuss their implications and provide directions for possible future research.

## 6.1 Conclusions

Nowadays, with fast-paced technological developments, many organizations generate large amounts of sequential data in many scientific and business applications. Time series data and data streams are the two most popular types of sequential data, sharing many common characteristics and mining constraints. The ultimate goal of this study is to develop classification algorithms and frameworks for them, which must be *fast*, *accurate* and *scalable* with respect to the size and dimensionality of datasets. The contributions of this dissertation are summarized as follows:

(i) **Closed Motifs for Streaming Time Series Classification:**

Motif discovery plays an important role in time series classification. However, state-of-the-art motif discovery algorithms suffer from the problem of finding only predefined length motifs, which greatly affects their performance and practicality. In addition, it is difficult to determine the optimal length of interesting motifs; a suboptimal choice results in missing the key motifs or having too many redundant motifs. To overcome this challenge, we introduce the notion of a *closed* motif; a motif is *closed* if there is no motif with a longer length having the same number of occurrences. We propose a novel algorithm $closedMotif$ to discover closed motifs in a single scan for streaming time series. We also use the nearest-neighbor classifier using the most distinctive closed motifs to validate their potential in time series classification. Extensive experiments with both synthetic and real time series datasets in various domains have shown that our $closedMotif$ algorithm not only discovers closed motifs with different lengths, but also outperforms prominent time series classifiers in terms of accuracy and speed.

(ii) **Heterogeneous Ensemble for Feature Drifts in Data Streams:**

Dimensionality reduction is essential for high dimensional data streams since it makes the learning process faster with more meaningful results. However, in a data stream environment, the set of important features is dynamic and changes over time. We define the concept of *feature drift* and explore relationships between feature drifts and concept drifts. We proposed a general framework that adapts to different types of concept drifts by integrating feature selection and heterogeneous ensemble learning. We detect feature drifts by modifying feature selection techniques with a sliding window approach. In a heterogeneous ensemble, the ratios of classifier types are dynamically adjusted to increase the ensemble's diversity, and to enable the ensemble to work well with many kinds of datasets. The ensemble adapts to the severity of concept drifts by updating online classifier members

for gradual drifts, and replacing an outdated member by a new one for sudden drifts. Furthermore, it is equipped with an optimal weighting strategy that has been proven theoretically and experimentally. We have conducted extensive experiments to show that our ensemble outperforms state-of-the-art ensemble learning algorithms for data streams.

(iii) **Concurrent Semi-Supervised Learning for Data Streams:**

Conventional stream mining algorithms focus on stand-alone mining tasks. Given the single-pass nature of data streams, it makes sense to maximize throughput by performing multiple complementary mining tasks concurrently; the knowledge gained by one mining task may be useful to other mining tasks. We propose an incremental algorithm called CSL-Stream that performs clustering and classification at the same time. We also introduce advanced techniques to keep the space and time complexities low with theoretical bounds and empirical evaluations. Moreover, enhanced with a novel active learning technique, CSL-Stream only requires a small number of queries to work well with very sparsely labeled datasets. The success of CSL-Stream paves the way for a new research direction in understanding latent commonalities among various data mining tasks in order to exploit the power of concurrent stream mining.

## 6.2 Future Work

Our proposed time series and data stream classifiers can be extended in various ways.

- **Extension for Dynamic Feature Space:**

  Our algorithm HEFT-Stream is able to remove both irrelevant and redundant features and works with various types of classifiers. However, HEFT-Stream currently

uses the filter model, which is independent of classifier members. Therefore, the problem of dynamic feature selection in data streams still requires further exploration. A hybrid feature selection technique that considers classifier members' performance while capturing features' evolution dynamically is much desired. We will continue to examine the relationship between concept and feature drifts and develop a metric to quantify concept drifts and use it to further adapt ensembles to achieve better accuracy. We will also investigate more intelligent methods to adjust the ratios of classifier types as well as the ensemble size. Furthermore, we wish to apply our algorithm for classifying text and other similar data streams where the number of features is very large ($> 10,000$) and evolves over time. For example, consider the problem of text mining in the *Twitter* social network. Twitter messages are short and generated constantly; new topics (classes) may emerge frequently. The vocabulary (number of features) is large and expected to increase rapidly over time. Consequently, feature drifts may occur more often with complex behaviors and a testing instance would have different features from the training instances.

- **Cloud Computing:**

  Cloud computing has recently become a highly disruptive technology as costs go down and adoption proliferates. Cloud computing aims to provide computation power, data access, software, and storage services that are available anywhere, anytime, on demand and does not require end-user knowledge of the underlying system. As data streams are growing larger and faster, it would be difficult to perform mining with a single computer with limited computing resources. Therefore, cloud computing is certainly going to be important for data stream mining in the near future. It is thus necessary to adapt and migrate our algorithms to cloud computing platforms, such as MapReduce [168] or Hadoop [169].

- **Privacy-Preserving Data Stream Mining:**

  We would like to study the problem of privacy in data stream mining, especially in a cloud computing context. Recently, there have been demands for mining data from many organizations without violating individual privacy. For example, many hospitals in Singapore may wish to cooperate by applying data mining on a large unified dataset without compromising the privacy of individual patients. In the context of data streams, the privacy-preservation problem has not been effectively addressed. With stream data arriving with high speed, it is difficult to perform expensive privacy transformations, such as encryption or randomization. Concept drifts may also further complicate the mining process as its detection will be doubly difficult.

The road ahead certainly looks exciting, laden with challenges and opportunities, and it is our hope that this thesis has made a difference in paving the way for the exploitation of data stream mining for the betterment of our world at large.

## 6.3 Publications

The author's main publications are listed as follows:

- Hai-Long Nguyen, Wee-Keong Ng, Yew-Kwong Woon. *"A Survey on Data Stream Clustering and Classification"*, Knowledge and Information Systems (KAIS), Springer (under review).
- Hai-Long Nguyen, Wee-Keong Ng, Yew-Kwong Woon. *"Closed Motifs for Streaming Time Series Classification"*, Knowledge and Information Systems (KAIS), 1-25, 2013, Springer.
- Hai-Long Nguyen, Wee-Keong Ng, Yew-Kwong Woon. *"Concurrent semi-supervised learning with active learning of data streams"*, Transactions on Large-Scale Data- and Knowledge-Centered Systems VIII, 113-136, 2013, Springer.
- Hai-Long Nguyen, Yew-Kwong Woon, Wee-Keong Ng, Li Wan. *"Heterogeneous ensemble for feature drifts in data streams"*, In Proceedings of 16th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II, 1-12, 2012, Springer.
- Hai-Long Nguyen, Wee-Keong Ng, Yew-Kwong Woon, Huy-Duc Tran. *"Concurrent semi-supervised learning of data streams"*, In Proceedings of the 13th international conference on Data Warehousing and Knowledge Discovery, 445-459, 2011, Springer.

Other publications:

- Huy-Duc Tran, Wee-Keong Ng, Hoon Lim, Hai-Long Nguyen. *"An efficient cacheable secure scalar product protocol for privacy-preserving data mining"*, In Proceedings of the 13th international conference on Data Warehousing and Knowledge Discovery, 354-366, 2011, Springer.

- Huy-Duc Tran, Hai-Long Nguyen, Wei Zha, Wee-Keong Ng. *"Towards security in sharing data on cloud-based social networks"*, In Proceedings of 8th International Conference on Information, Communications and Signal Processing (ICICS), 1-5, 2011, IEEE.

*"If I have seen further it is by standing on the shoulders of giants."*

Isaac Newton (1642 -1727) Scientist

# References


[1] Anand Narasimhamurthy and Ludmila I Kuncheva. A framework for generating data to simulate changing environments. In *Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications*, volume 549, page 389. ACTA Press, 2007.

[2] L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. pages 947–956. ACM, 2009.

[3] Jessical Lin, Eamonn Keogh, Stefano Lonardi, and Pranav Patel. Finding motifs in time series. In *the 2nd Workshop on Temporal Data Mining*, pages 53–68, 2002.

[4] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and C. A. Ratanamahatana. The ucr time series classification/clustering homepage: www.cs.ucr.edu/ eamonn/time_series_data ., 2011.

[5] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[6] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, 2008.

[7] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction, 2006.

[8] Krisztian Buza, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Insight: efficient and effective instance selection for time-series classification. In *Proceedings of the 15th Pacific-Asia conference on Advances in knowledge discovery and data mining - Volume Part II*, pages 149–160, 2022863, 2011. Springer-Verlag.

[9] Abdullah Mueen, Eamonn Keogh, and Neal Young. Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1154–1162, 2020587, 2011. ACM.

[10] H. Tang and S.S. Liao. Discovering original motifs with different lengths from time series. *Knowledge-Based Systems*, 21(7):666–671, 2008.

[11] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 347107, 2000. ACM.

[12] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, 502529, 2001. ACM.

[13] Thomas Seidl, Ira Assent, Philipp Kranen, Ralph Krieger, and Jennifer Herrmann. Indexing density models for incremental learning and anytime classification on data streams. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 311–322, 1516397, 2009. ACM.

[14] Philipp Kranen, Stephan Gnnemann, Sergej Fries, and Thomas Seidl. *MC-Tree: Improving Bayesian Anytime Classification*, volume 6187 of *Lecture Notes in Computer Science*, pages 252–269. Springer Berlin / Heidelberg, 2010.

[15] Ivor W. Tsang, Andras Kocsor, and James T. Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th international conference on Machine learning*, pages 911–918, 1273611, 2007. ACM.

[16] Piyush Rai, Hal Daum, and Suresh Venkatasubramanian. Streamed learning: one-pass svms. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1211–1216, 1661639, 2009. Morgan Kaufmann Publishers Inc.

[17] P. Zhang, B.J. Gao, X. Zhu, and L. Guo. Enabling fast lazy learning for data streams. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM-11)*, pages 932–941. IEEE, 2011.

[18] N. C. Oza. Online bagging and boosting. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345. IEEE, 2005.

[19] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235, 956778, 2003. ACM.

[20] Peng Zhang, Xingquan Zhu, Yong Shi, and Xindong Wu. *An Aggregate Ensemble for Mining Concept Drifting Data Streams with Noise*, volume 5476 of *Lecture Notes in Computer Science*, pages 1021–1029. Springer Berlin / Heidelberg, 2009.

[21] Peng Zhang, Xingquan Zhu, Yong Shi, Li Guo, and Xindong Wu. Robust ensemble learning for mining noisy data streams. *Decision Support Systems*, 50(2):469–479, 2011.

[22] Hashemi Sattar, Yang Ying, Mirzamomen Zahra, and Kangavari Mohammadreza. Adapted one-vs-all decision trees for data stream classification. In *IEEE Transactions on Knowledge and Data Engineering*, volume 21, pages 624 – 637, 2009.

[23] Peng Zhang, Jun Li, Peng Wang, Byron J. Gao, Xingquan Zhu, and Li Guo. Enabling fast prediction for ensemble models on data streams. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 177–185, 2020442, 2011. ACM.

[24] P.P. Rodrigues, J. Gama, and J.P. Pedroso. Odac: Hierarchical clustering of time series data streams. pages 499–503. Bethesda, Maryland, USA: SIAM, 2006.

[25] Y. Yang and K. Chen. Temporal data clustering via weighted clustering ensemble with different representations. *Knowledge and Data Engineering, IEEE Transactions on*, 23(2):307–320, 2011.

[26] Ling Chen, Ling-Jun Zou, and Li Tu. A clustering algorithm for multiple data streams based on spectral component similarity. *Information Sciences*, 183(1):35–47, 2012.

[27] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 685–694, 2002.

[28] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases*, volume 29, pages 81–92. VLDB Endowment, 2003.

[29] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, pages 852–863, 1316763, 2004. VLDB Endowment.

[30] Aoying Zhou, Feng Cao, Weining Qian, and Cheqing Jin. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems*, 15(2):181–214, 2008.

[31] Komkrit Udommanetanakit, Thanawin Rakthanmanon, and Kitsana Waiyamai. *E-Stream: Evolution-Based Technique for Stream Clustering*, volume 4632 of *Lecture Notes in Computer Science*, pages 605–615. Springer Berlin / Heidelberg, 2007.

[32] Sebastian Lhr and Mihai Lazarescu. Incremental clustering of dynamic data streams using connectivity based representative points. *Data and Knowledge Engineering*, 68(1):1–27, 2009.

[33] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 328–339, 2006.

[34] Dimitris K. Tasoulis, Gordon Ross, and Niall M. Adams. Visualising the cluster structure of data streams. In *Proceedings of the 7th international conference on Intelligent data analysis*, pages 81–92, 1771633, 2007. Springer-Verlag.

[35] Li Wan, Wee Keong Ng, Xuan Hong Dang, Philip S. Yu, and Kuan Zhang. Density-based clustering of data streams at multiple resolutions. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(3):1–28, 2009.

[36] Nam Hun Park and Won Suk Lee. Statistical grid-based clustering over data streams. *ACM SIGMOD Record*, 33(1):32–37, 2004.

[37] Sudipto Guha, Chulyun Kim, and Kyuseok Shim. Xwave: optimal and approximate extended wavelets. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, pages 288–299, 1316716, 2004. VLDB Endowment.

[38] Xuan Dang, Vincent Lee, Wee Ng, Arridhana Ciptadi, and Kok Ong. *An EM-Based Algorithm for Clustering Data Streams in Sliding Windows*, volume 5463 of *Lecture Notes in Computer Science*, pages 230–235. Springer Berlin / Heidelberg, 2009.

[39] Toby Smith and Damminda Alahakoon. *Growing Self-Organizing Map for Online Continuous Clustering*, volume 204 of *Studies in Computational Intelligence*, pages 49–83. Springer Berlin / Heidelberg, 2009.

[40] B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–498. ACM, 2003.

[41] T. Id. Why does subsequence time-series clustering produce sine waves? *10th European conference on Principle and Practice of Knowledge Discovery in Databases*, pages 211–222, 2006.

[42] J. Lin, E. Keogh, S. Lonardi, J.P. Lankford, and D.M. Nystrom. Visually mining and monitoring massive time series. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 460–469. ACM, 2004.

[43] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[44] J. Shieh and E. Keogh. isax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631. ACM, 2008.

[45] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. isax 2.0: Indexing and mining one billion time series. In *IEEE 10th International Conference on Data Mining (ICDM)*, pages 58–67. IEEE, 2010.

[46] Chao Li, Michael Hay, Vibhor Rastogi, Gerome Miklau, and Andrew McGregor. Optimizing linear counting queries under differential privacy. In *Proceedings of*

*the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 123–134. ACM, 2010.

[47] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357, 1287400, 2002. VLDB Endowment.

[48] Giannella Chris, Y. Jiaweihan, Z. Jianpei, Y. Xifeng Yan, and S. Yu Philip. Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212, 2003.

[49] Nan Jiang and Le Gruenwald. Cfi-stream: mining closed frequent itemsets in data streams. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 592–597, 1150473, 2006. ACM.

[50] James Cheng, Yiping Ke, and Wilfred Ng. A survey on algorithms for mining frequent itemsets over data streams. *Knowl. Inf. Syst.*, 16(1):1–27, 2008.

[51] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proc. VLDB Endow.*, 1(2):1530–1541, 2008.

[52] Hua-Fu Li, Man-Kwan Shan, and Suh-Yin Lee. Dsm-fi: an efficient algorithm for mining frequent itemsets in data streams. *Knowledge and Information Systems*, 17(1):79–97, 2008.

[53] Xuan Hong Dang, Wee-Keong Ng, and Kok-Leong Ong. Online mining of frequent sets in data streams with error guarantee. *Knowl. Inf. Syst.*, 16(2):245–258, 2008.

[54] Woo Ho Jin and Lee Won Suk. estmax: Tracing maximal frequent item sets instantly over online transactional data streams. *Knowledge and Data Engineering, IEEE Transactions on*, 21(10):1418–1431, 2009.

[55] Hoang Thanh Lam and Toon Calders. Mining top-k frequent items in a data stream with flexible sliding windows. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 283–292, 1835842, 2010. ACM.

[56] Leo Breiman, Jerome Friedman, Charles Stone, and R. A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[57] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[58] Ross Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, 1993.

[59] S. Russell, J. Binder, D. Koller, and K. Kanazawa. Local learning in probabilistic networks with hidden variables. volume 14, pages 1146–1152. Citeseer, 1995.

[60] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *The Journal of Machine Learning Research*, 29(2-3):103–130, 1997.

[61] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[62] W.W. Cohen and Y. Singer. A simple, fast, and effective rule learner. pages 335–342. JOHN WILEY & SONS LTD, 1999.

[63] D.E. Rumelhart, G.E. Hintont, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[64] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1999.

[65] Jiang Liangxiao, Cai Zhihua, Wang Dianhong, and Jiang Siwei. Survey of improving k-nearest-neighbor for classification. In *Fourth International Conference on Fuzzy*

*Systems and Knowledge Discovery, 2007. FSKD 2007.*, volume 1, pages 679–683, 2007.

[66] Yoav Freund and Robert Schapire. Experiments with a new boosting algorithm. In *the 13th International Conference on Machine Learning*, pages 148–156, 1996.

[67] W. Nick Street and YongSeog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, 502568, 2001. ACM.

[68] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavald. New ensemble methods for evolving data streams. In *15th ACM SIGKDD*, pages 139–148, 1557041, 2009. ACM.

[69] Yu Lei and Liu Huan. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *The 20th ICML*, pages 856–863, 2003.

[70] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005.

[71] Joo Gama and Pedro Rodrigues. *An Overview on Mining Data Streams*, volume 206 of *Studies in Computational Intelligence*, pages 29–45. Springer Berlin / Heidelberg, 2009.

[72] Charu C. Aggarwal. *Data Streams: An Overview and Scientific Applications*, pages 377–397. Springer Berlin Heidelberg, 2009.

[73] Joo Gama and Mohamed Medhat Gaber. *Learning from Data Streams – Processing Techniques in Sensor Networks*. Springer, 2007.

[74] Daby Sow, Alain Biem, Marion Blount, Maria Ebling, and Olivier Verscheure. Body sensor data processing using stream computing. In *Proceedings of the international conference on Multimedia information retrieval*, pages 449–458, 1743465, 2010. ACM.

[75] Hua-Jun Zeng, Qi-Cai He, Zheng Chen, Wei-Ying Ma, and Jinwen Ma. Learning to cluster web search results. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 210–217, 1009030, 2004. ACM.

[76] Steve Chien and Nicole Immorlica. Semantic similarity between search engine queries using temporal correlation. In *Proceedings of the 14th international conference on World Wide Web*, pages 2–11, 1060752, 2005. ACM.

[77] S. Muthu Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc, 2003.

[78] Mark G. Kelly, David J. Hand, and Niall M. Adams. The impact of changing populations on classifier performance. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 367–371, 312285, 1999. ACM.

[79] Albert Bifet. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. Proceeding of the 2010 conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams. IOS Press, 2010.

[80] Mark Last. Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6(2):129–147, 2002.

[81] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 1281210, 2007. ACM.

[82] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for on-demand classification of evolving data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(5):577–589, 2006.

[83] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 359–366, 2000.

[84] D. F. Leite, P. Costa, and F. Gomide. Evolving granular classification neural networks. In *International Joint Conference on Neural Networks, 2009. IJCNN 2009.*, pages 1736–1743, 2009.

[85] Nam Hun Park and Won Suk Lee. Cell trees: An adaptive synopsis structure for clustering multi-dimensional on-line data streams. *Data and Knowledge Engineering*, 63(2):528–549, 2007.

[86] Leo Breiman. Bagging predictors. *The Journal of Machine Learning Research*, 24(2):123–140, 1996.

[87] Ho Tin Kam. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[88] Leo Breiman. Random forests. *The Journal of Machine Learning Research*, 45(1):5–32, 2001.

[89] Gunther Eibl and Karl-Peter Pfeiffer. Multiclass boosting for weak classifiers. *The Journal of Machine Learning Research*, 6:189–210, 2005.

[90] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[91] Chunhua Shen and Hanxi Li. On the dual formulation of boosting algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2216–2231, 2010.

[92] R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[93] M. Kadous and C. Sammut. Constructive induction for classifying time series. *Machine Learning: ECML 2004*, pages 192–204, 2004.

[94] A. Kehagias and V. Petridis. Predictive modular neural networks for time series classification. *Neural networks*, 10(1):31–49, 1997.

[95] V. Pavlovic, B.J. Frey, and T.S. Huang. Time-series classification using mixed-state dynamic bayesian networks. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

[96] D. Eads, D. Hill, S. Davis, S. Perkins, J. Ma, R. Porter, and J. Theiler. Genetic algorithms and support vector machines for time series classification. In *Proc. of SPIE Vol*, volume 4787, page 75, 2002.

[97] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[98] C.A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *the SIAM International Data Mining Conference*, pages 506–510, 2005.

[99] Eamonn Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 406–417, 1287405, 2002. VLDB Endowment.

[100] A. Bagnall, L. Davis, J. Hills, and J. Lines. Transformation based ensembles for time series classification. *Proc. 12th SDM*, 2012.

[101] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 370–377. IEEE, 2002.

[102] E. Keogh and J. Lin. Clustering of time-series subsequences is meaningless: implications for previous and future research. *Knowledge and Information Systems*, 8(2):154–177, 2005.

[103] A. Mueen and E. Keogh. Online discovery and maintenance of time series motifs. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1089–1098. ACM, 2010.

[104] H.T. Lam, N.D. Pham, and T. Calders. Online discovery of top-k similar motifs in time series data. In *Proceedings of the Eleventh SIAM International Conference*, 2011.

[105] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *ACM SIGMOD international conference on Management of data*, volume 23. ACM, 1994.

[106] K.P. Chan and A.W.C. Fu. Efficient time series matching by wavelets. In *the 15th International Conference on Data Engineering*, pages 126–133. IEEE, 1999.

[107] K Chakrabarti, E Keogh, S Mehrotra, and M Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems (TODS)*, 27(2):188–228, 2002.

[108] Michalis Vlachos. A practical time series tutorial with tutorial with matlab, 2005.

[109] Ioannis P. Androulakis. New approaches for representing, analyzing and visualizing complex kinetic mechanisms. In *Proceedings of the 15th European symposium on computer aided process engineering*, 2005.

[110] Pedro G. Ferreira, Paulo J. Azevedo, Candida G. Silva, and Rui M. M. Brito. Mining approximate motifs in time series. In *Proceedings of the 9th international conference on Discovery Science*, pages 89–101, 2089941, 2006. Springer-Verlag.

[111] Amy McGovern, Adrianna Kruger, Derek Rosendahl, and Kelvin Droegemeier. Open problem: dynamic relational models for improved hazardous weather prediction. In *Proceedings of ICML workshop on open problems in statistical relational learning*, 2006.

[112] Jian-Sheng Chen, Yiu-Sang Moon, and Hoi-Wo Yeung. Palmprint authentication using time series. In *Proceedings of the 5th international conference on Audio- and Video-Based Biometric Person Authentication*, pages 376–385, 2134905, 2005. Springer-Verlag.

[113] K. Uehara, Y. Tanaka, and K. Makio. Motif discovery algorithm from motion data. In *Annual Conference on JSAI*, volume 18, pages 3D3–01, 2004.

[114] B. Celly and V.B. Zordan. Animated people textures. In *Proceedings of the 17th international conference on computer animation and social agents*, 2004.

[115] A Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[116] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: a linear-time algorithm. *J. Artif. Int. Res.*, 7(1):67–82, 1997.

[117] S. Deroski, V. Gjorgjioski, I. Slavkov, and J. Struyf. Analysis of time series data with predictive clustering trees. *Knowledge Discovery in Inductive Databases*, pages 63–80, 2007.

[118] P. Ferreira and P. Azevedo. Protein sequence classification through relevant sequence mining and bayes classifiers. *Progress in Artificial Intelligence*, pages 236–247, 2005.

[119] V. Kunik, Z. Solan, S. Edelman, E. Ruppin, and D. Horn. Motif extraction and protein classification. In *Computational Systems Bioinformatics Conference, 2005. Proceedings. 2005 IEEE*, pages 80–85. IEEE, 2005.

[120] K. Buza and L. Schmidt-Thieme. Motif-based classification of time series with bayesian networks and svms. *Advances in Data Analysis, Data Handling and Business Intelligence*, pages 105–114, 2010.

[121] Jan de Leeuw. *Applications of Convex Analysis to Multidimensional Scaling*, pages 133–146. North Holland Publishing Company, Amsterdam, 2005.

[122] D. Leite, P. Costa, and F. Gomide. Evolving granular neural network for semi-supervised data stream classification. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

[123] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 671005, 1997. Morgan Kaufmann Publishers Inc.

[124] Hans-Peter Kriegel, Peer Kroger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1–58, 2009.

[125] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[126] Hans-Peter Kriegel, Peer Krger, Irene Ntoutsi, and Arthur Zimek. *Density Based Subspace Clustering over Dynamic Data*, volume 6809 of *Lecture Notes in Computer Science*, pages 387–404. Springer Berlin / Heidelberg, 2011.

[127] Xiaojin Zhu, Andrew B. Goldberg, Ronald Brachman, and Thomas Dietterich. *Introduction to Semi-Supervised Learning*. Morgan and Claypool Publishers, 2009.

[128] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research*, 7:2399–2434, 2006.

[129] Avrim Blum and Shuchi Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 19–26, 757779, 2001. Morgan Kaufmann Publishers Inc.

[130] Thorsten Joachims. *Making large-scale support vector machine learning practical*, pages 169–184. MIT Press, 1999.

[131] Olivier Chapelle, Vikas Sindhwani, and Sathiya S. Keerthi. Optimization techniques for semi-supervised support vector machines. *The Journal of Machine Learning Research*, 9:203–233, 2008.

[132] Vikas Sindhwani and S. Sathiya Keerthi. Large scale semi-supervised linear svms. In *ACM SIGIR*, pages 477–484, 1148253, 2006. ACM.

[133] M. M. Masud, Gao Jing, L. Khan, Han Jiawei, and B. Thuraisingham. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *ICDM*, pages 929–934, 2008.

[134] Kiri Wagstaff, Claire Cardie, and Seth Rogers. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, 655669, 2001. Morgan Kaufmann Inc.

[135] Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 59–68, 1014062, 2004. ACM.

[136] Charu Aggarwal, Jiawei Han, Jianyong Wang, and Philip Yu. *On Clustering Massive Data Streams: A Summarization Paradigm*, volume 31 of *Advances in Database Systems*, pages 9–38. Springer US, 2007.

[137] Like Gao and X. Sean Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 370–381, 564734, 2002. ACM.

[138] G. Das, K.I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. *Knowledge Discovery and Data Mining*, pages 16–22, 1998.

[139] Frank Hoppner. Discovery of temporal patterns – learning rules about the qualitative behavior of time series. In Luc De Raedt and Arno Siebes, editors, *Principles and Practice of Knowledge Discovery in Databases*, volume 2168 of *Lecture Notes in Computer Science*, pages 192–203. Springer Berlin / Heidelberg, 2001.

[140] F. Mrchen. *Time series knowledge mining*. PhD thesis, 2006.

[141] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. pages 239–241, 1998.

[142] S. Rombo and G. Terracina. Discovering representative models in large time series databases. *Flexible Query Answering Systems*, pages 84–97, 2004.

[143] E. Keogh, J. Lin, and A. Fu. Hot sax: efficiently finding the most unusual time series subsequence. In *Data Mining, Fifth IEEE International Conference on*, page 8 pp., 2005.

[144] M. Ohsaki, Y. Sato, H. Yokoi, and T. Yamaguchi. A rule discovery support system for sequential medical data, in the case study of a chronic hepatitis dataset. In *Proc of the ECML/PKDD-2003 Discovery Challenge Workshop*, pages 154–165, 2003.

[145] D. Minnen, T. Starner, M. Essa, and C. Isbell. Discovering characteristic actions from on-body sensor data. In *Wearable Computers, 2006 10th IEEE International Symposium on*, pages 11–18, 2006.

[146] P. Nunthanid, V. Niennattrakul, and C.A. Ratanamahatana. Discovery of variable length time series motif. In *ECTI-CON*, pages 472–475. IEEE, 2011.

[147] Jian Pei, Jiawei Han, and Runying Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD*, pages 21–30, 2000.

[148] G.S. Linoff and M.J. Berry. *Data mining techniques: for marketing, sales, and customer relationship management*. Wiley Computer Publishing, 2011.

[149] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Trans. Internet Technol.*, 3(1):1–27, 2003.

[150] C. Liu, X. Yan, H. Yu, J. Han, and P.S. Yu. Mining behavior graphs for backtrace of noncrashing bugs. In *SIAM International Conference on Data Mining*, pages 286–297, 2005.

[151] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, Mark R. G., Mietus J. E., G. B. Moody, C. K. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101:215–220, 2000.

[152] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover. Exact discovery of time series motifs. In *Proceedings of SIAM International Conference on Data Mining*, pages 473–484, 2009.

[153] N. Castro and P. Azevedo. Multiresolution motif discovery in time series. In *Proceedings of the SIAM International Conference on Data Mining*, pages 665–676, 2010.

[154] Intel lab data. available online at: http://berkeley.intel-research.net/labdata/ .

[155] Kuo-Wei Hsu and Jaideep Srivastava. *Diversity in Combinations of Heterogeneous Classifiers*, volume 5476 of *Lecture Notes in Computer Science*, pages 923–932. Springer Berlin / Heidelberg, 2009.

[156] Kevin Woods, Jr. W. Philip Kegelmeyer, and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, 1997.

[157] Lu Zhenyu, Wu Xindong, and J. Bongard. Active learning with adaptive heterogeneous ensembles. In *the 9th IEEE ICDM*, pages 327–336, 2009.

[158] K. Tumer and J. Ghosh. Linear and order statistics combiners for pattern classification. *Springer-Verlag*, 1999.

[159] G. Fumera and F. Roli. A theoretical and experimental analysis of linear combiners for multiple classifier systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):942–956, 2005.

[160] Albert Bifet, Geoff Holmes, and Richard Kirkby. Moa: Massive online analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[161] R. Vilalta and I. Rish. A decomposition of classes via clustering to explain and improve naive bayes. In *In Proceedings of 14th European Conference on Machine Learning (ECML/PKDD)*, 2003.

[162] Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.

[163] Ester Martin, Kriegel Hans-Peter, Sander Jrg, and Xu Xiaowei. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, pages 226–231. AAAI Press, 1996.

[164] E.B. Baum and K. Lang. Query learning can work poorly when a human oracle is used. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, page 335340. IEEE Press, 1992.

[165] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *the ACM SIGIR Conference on Research and Development in Information Retrieval.*, pages 3–12. ACM/Springer,, 1994.

[166] A. Frank and A. Asuncion. UCI machine learning repository [http://archive.ics.uci.edu/ml], 2010.

[167] Enrique Amigo, Julio Gonzalo, and Javier Artiles. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, 12(4):461–486, 2009.

[168] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[169] D. Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11:21, 2007.