Change Detection in Streaming Data

Dissertation
Zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt der
Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von
M.Sc. Dang Hoan Tran

Vorgelegt am 30. Mai 2013
Tag der wissenschaftlichen Aussprache: am 08. Oktober 2013

Gutachter

1. Prof. Dr.-Ing. habil. Kai-Uwe Sattler
2. Dr. Mohamed Medhat Gaber
3. Prof. Dr. Peter Fischer

urn:nbn:de:gbv:ilm1-2013000513

CHANGE DETECTION IN STREAMING DATA

A DISSERTATION SUBMITTED
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOKTOR-INGENIEUR (DR.-ING.)

FACULTY OF COMPUTER SCIENCE AND AUTOMATION
ILMENAU UNIVERSITY OF TECHNOLOGY

FROM
M.SC. DANG HOAN TRAN

READING COMMITTEE

1. PROF. DR.-ING. HABIL. KAI-UWE SATTLER
2. DR. MOHAMED MEDHAT GABER
3. PROF. DR. PETER FISCHER

URN:NBN:DE:GBV:ILM1-2013000513

Abstract

Change detection is the process of identifying differences in the state of an object or phenomenon by observing it at different times or different locations in space. In the streaming context, it is the process of segmenting a data stream into different segments by identifying the points where the stream dynamics changes. Decentralized change detection can be used in many interesting, and important applications such environmental observing systems, medicare monitoring systems. Although there is great deal of work on distributed detection and data fusion, most of work focuses on the one-time change detection solutions. One-time change detection method requires to proceed data once in response to the change occurring. The trade-off of a continuous distributed detection of changes include detection accuracy, space-efficiency, detection delay, and communication-efficiency.

To achieve these goals, the wildfire warning system is used as a motivating scenario. From the challenges and requirements of the wildfire warning system, the change detection algorithms for streaming data are proposed a part of the solution to the wildfire warning system. By selecting various models of local change detection, different schemes for distributed change detections, and the data exchange protocols, different designs can be achieved.

Based on this approach, the contributions of this dissertation are as follows. A general two-window framework for detecting changes in a single data stream is presented. A general synopsis-based change detection framework is proposed. Theoretical and empirical analysis shows that the detection performance of synopsis-based detector is similar to that of non-synopsis change detector if a distance function quantifying the changes is preserved under the process of constructing synopsis. A clustering-based change detection and clustering maintenance method over sliding window is presented. Clustering-based detector can automatically detect the changes in the multivariate streaming data. A framework for decentralized change detection in wireless sensor networks is proposed. A distributed framework for clustering streaming data is proposed by extending the two-phased stream clustering approach which is widely used to cluster a single data stream.

Zusammenfassung

Unter Änderungserkennung wird der Prozess der Erkennung von Unterschieden im Zustand eines Objekts oder Phänomens verstanden, wenn dieses zu verschiedenen Zeitpunkten oder an verschiedenen Orten beobachtet wird. Im Kontext der Datenstromverarbeitung stellt dieser Prozess die Segmentierung eines Datenstroms anhand der identifizierten Punkte, an denen sich die Stromdynamiken ändern, dar. Die Fähigkeit, Änderungen in den Stromdaten zu erkennen, darauf zu reagieren und sich daran anzupassen, spielt in vielen Anwendungsbereichen, wie z.B. dem Aktivitätsüberwachung, dem Datenstrom-Mining und Maschinenlernen sowie dem Datenmanagement hinsichtlich Datenmenge und Datenqualität, eine wichtige Rolle. Dezentralisierte Änderungserkennung kann in vielen interessanten und wichtigen Anwendungsbereichen, wie z.B. in Umgebungsüberwachungssystemen oder medizinischen Überwachungssystemen, eingesetzt werden. Obgleich es eine Vielzahl von Arbeiten im Bereich der verteilten Änderungserkennung und Datenfusion gibt, liegt der Fokus dieser Arbeiten meist lediglich auf der Erkennung von einmaligen Änderungen. Die einmalige Änderungserkennungsmethode erfordert die einmalige Verarbeitung der Daten als Antwort auf die auftretende Änderung. Der Kompromiss einer kontinuierlichen, verteilten Erkennung von Änderungen umfasst die Erkennungsgenauigkeit, die Speichereffizienz sowie die Berechnungseffizienz. Um dieses Ziel zu erreichen, wird das Flächenbrandwarnsystem als motivierendes Szenario genutzt. Basierend auf den Herausforderungen und Anforderungen dieses Warnsystems wird ein Algorithmus zur Erkennung von Änderungen in Stromdaten als Teil einer Gesamtlösung für das Flächenbrandwarnsystem vorgestellt. Durch die Auswahl verschiedener Modelle zur lokalen und verteilten Änderungserkennung sowie verschiedener Datenaustauschprotokolle können verschiedene Systemdesigns entwickelt werden. Basierend auf diesem Ansatz leistet diese Dissertation nachfolgend aufgeführte Beiträge. Es wird ein allgemeines 2-Fenster Framework zur Erkennung von Änderungen in einem einzelnen Datenstrom vorgestellt. Weiterhin wird ein allgemeines synopsenbasiertes Framework zur Änderungserkennung beschrieben. Mittels theoretischer und empirischer Analysen wird gezeigt, dass die Erkennungs-Performance des synopsenbasierten Änderungsdetektors ähnlich der eines nicht-synopsenbasierten ist, solange eine Distanzfunktion, welche die Änderungen quantifiziert, während der Erstellung der Synopse eingehalten wird. Es wird Cluster-basierte Änderungserkennung und Cluster-Pflege über gleitenden Fenstern vorgestellt. Weiterhin wird ein Framework zur verteilten Änderungserkennung in drahtlosen Sensornetzwerken beschrieben. Basierend auf dem 2-Phasen Stromdaten-Cluster-Ansatz, welcher weitestgehend zur Clusterung eines einzelnen Datenstroms eingesetzt wird, wird ein verteiltes Framework zur Clusterung von Stromdaten vorgestellt.

For Minh Duc, Quoc Bao

Acknowledgments

I am fortunate to have opportunity in living and studying in Germany, a nation is the origin of the modern higher education with the famous Humboldt model which promotes and advocates the free spirit of research university. In particular, I am fortunate to have Professor. Dr.-Ing.habil. Kai-Uwe Sattler as my supervisor, or Doktorvater in German. I like this word because it has more meanings than supervisor, and advisor. I am profoundly grateful to him for offering me this difficult but interesting research topic and for his insightful advices, patience, and kindness over the years.

I thank Dr Mohamed Gaber at the Data Science Research Group of School of Computing University of Portsmouth for reading and giving insightful suggestions that have improved this dissertation.

I thank Professor. Dr. Peter Fischer at the Web Science group of Institut für Informatik, Albert-Ludwigs-Universität Freiburg for spending time on reading my thesis and giving me many detailed and insightful comments that make thesis more clearly.

I would like to thank all the members of Ph.D. committee: Professor. Dr. Martin Dietzfelbinger, Professor. Dr.-Ing.habil. Armin Zimmermann, and Professor. Dr. sc.techn. Beat Brüderlin.

I thank all my coauthors Professor Kai-Uwe Sattler, and Yang Jin. I thank Dr. rer. nat. Rita Schindler, Dr.-Ing. Christine Krause, Dr.-Ing. Daniel Klan, Stephan Baumann, Francis Gropengiesser, Felix Beier, and Liz Ribe Baumann, who support me very much during the time working in the group. I thank Mr. Peter Henkel, and Mr. Sauerbrey Martin for their technical support. I thank Ms. Hoang Hanh at School of Languages and Comparative Cultural Studies of the University of Queensland, for proofreading this dissertation. I thank all the officemates who support me when working in the same office: Anja Poelck, Felix Beier, Yang Jin, Heiko Betz, and Omran Saleh. I thank Ms.Katja Wolf, Ms.Marion Koch, Ms. Cordula Giewald, and Ms. Silvia Benz for their office supports. This thesis would have been impossible without you.

I gratefully acknowledge the financial supports of the Vietnam Ministry of Education and Training, TU Ilmenau, and the DAAD.

Finally, I am deeply indebted to my parent, my wife, and sisters for their love and support.

Contents

List of Figures

List of Tables

Part I

Introduction and Background

CHAPTER 1

Introduction

Change we need! Change we can believe in! (Barack Obama)

Contents

1.1 Thesis Statement

Today's world is changing very fast. The changes occur in every aspects of life. Therefore, the ability to detect, adapt, and react to the change play an important role in all aspects of life. The physical world is often represented in some model or some information system. The changes in the physical world are reflected in terms of the changes in data or model built from data. Therefore, the nature of data is changing. The advance of technology results in the data deluge. The data volume is increasing with an estimated rate of 50% per year [Manyika 2011]. Data flood makes traditional methods including traditional distributed framework and parallel models inappropriate for processing, analyzing, storing, and understanding these massive data sets. Data deluge needs a new generation of computing tools that Jim Gray calls the 4th paradigm in scientific computing [Hey 2009]. Recently, there have been some emerging computing paradigms that meet the requirements of Big Data as follows. Parallel batch processing model only deals with the stationary massive data [Dean 2008]. However, evolving data continuously arrives with

high speed. In fact, online data stream processing is the main approach to dealing with the problem of three characteristics of Big Data including big volume, big velocity, and big variety. Streaming data processing is a model of Big Data processing. Streaming data is temporal data in nature. In addition to the temporal nature, streaming data may include spatial characteristics. For example, geographic information systems can produce spatial-temporal data stream. Streaming data processing and mining have been deploying in real-world systems such as InforSphere Streams (IBM)[1], Rapidminer Streams Plugin [2], StreamBase [3], MOA [4], AnduIN [5]. In order to deal with the high-speed data streams, a hybrid model that combines the advantages of both parallel batch processing model and streaming data processing model is proposed. Some projects for such hybrid model include S4 [6], Storm [7], and Grok [8] .

One of these challenges facing data stream processing and mining is the changing nature of streaming data. Therefore, the ability to identify trends, patterns, and changes in the underlying processes generating data contributes to the success of processing and mining massive high-speed data streams.

A model of continuous distributed monitoring has been recently proposed to deal with streaming data coming from multiple sources. A model of continuous distributed monitoring consists of many observers and each observer monitoring a single data stream. The goal of continuous distributed monitoring is to perform some task that need to aggregate the incoming data from the observers. The continuous distributed monitoring is applied to monitor networks such as sensor networks, social networks, networks of ISP [Cormode 2013]. Along with this line of research, this dissertation deals with the problem of detection of changes in streaming data.

Change detection is the process of identifying differences in the state of an object or phenomenon by observing it at different times or different locations in space. In the streaming context, change detection is the process of segmenting a data stream into different segments by identifying the points where the stream dynamics changes [Ross 2009]. A change detection method consists of the following tasks: change detection and localization of change. Change detection detects whether a change occurs, and response to the presence of the change. Besides change detection, localization of changes determines the location of change. The problem of locating the change has been studied in the statistics in the problems of

[1] http://www-01.ibm.com/software/data/infosphere/streams/
[2] http://www-ai.cs.uni-dortmund.de/auto?self=$eit184kc
[3] http://www.streambase.com/
[4] http://moa.cs.waikato.ac.nz/
[5] http://www.tu-ilmenau.de/dbis/research/anduin/
[6] http://incubator.apache.org/s4/
[7] https://github.com/nathanmarz/storm/wiki/Tutorial
[8] https://www.numenta.com/grok_info.html

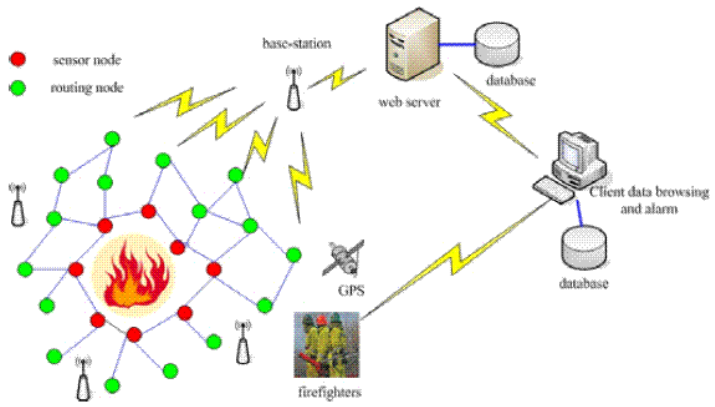


Figure 1.1: A wireless sensor network for wildfire warning system presented in [Li 2006]

change point detection. In this dissertation, we both determine whether a change occurs and content of a change, that means showing a change point, or a set of change points.

1.1.1 Motivating Scenario

The wildfire warning system is selected to study the problem of distributed detection of changes in streaming data as shown in Figure 1.1. The goal of this scenario is to describe the underlying integration model in which our change detection algorithms are components. The main tasks of a wildfire warning system include detecting a small fire, and determining its position in order to intervene and limit the spreading of the fire as fast as possible. Detecting the event of fire is based on the detecting of abnormal changes in sensor readings including temperature, humidity, and light intensity. Determining location of event is based on the GPS technology or the localization algorithm in wireless sensor networks [Li 2006].

The changes of environment such as the wildfire in Figure 1.1 can occur in some region where only a portion of the sensors are capable of detecting changes. As such, these changes of observations occur locally. The rest of the sensor network should know the information of occurred changes to make their decisions timely and accurately. Sensors detect changes in their ambient environments, such as the abnormal raise of the temperature in some area of a forest, then disseminate information of the detected changes to the rest of sensor network serving as alarms. The system architecture for wildfire warning system depicted in Figure 1.1 includes the following components. First, sensor nodes for sampling environmental measures such temperature, humidity, barometric pressure, and light intensity. Sensor node is a sensing and computing devices with restricted resources such as small mem-

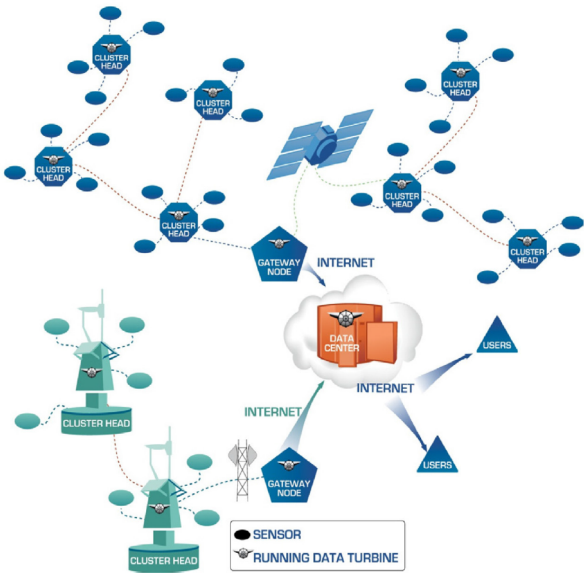


Figure 1.2: A system architecture for environmental observing systems presented in [Fountain 2012]

ory, limited sensing ranges, low computing power, and limited energy. Each node in the network is able to detect the changes of the environment within its vicinity. Sensors in wireless sensor networks are intelligent ones. An intelligent sensor can execute a program itself. There are two kinds of sensor nodes: sensor node and routing node. In fact, depending the distance with the location of event, a node can be a sensor node or routing node. Second, routing nodes transmits sensor readings to the base stations or the coordinators. Third, web server collects sensor readings from the base stations. Forth, database stores the sensor readings as history for later analysis. Finally, client reports the fire event and the location of fire based on GPS information to the fire-fighters to response to the fire if it occurs.

This architecture has the following drawbacks. First, the base station periodically collects the sensor readings reflecting the information about the environment. Such pull-based queries for information incurs high cost in terms of resource and accuracy of data and model. Second, the sensor readings are stored in a database, in particular MySQL. This database may not meet the requirement for data storage because of the continuous arrival of sensor readings.

Figure 1.2 shows a system architecture for environmental observing systems [Fountain 2012]. This architecture is more general than the architecture of Figure 1.1. In particular, the architecture of Figure 1.2 includes a robust real-time streaming data engine called DataTurbine. This streaming data engine is useful for streaming data from sensors to the base stations to the data centers. Instead of using traditional databases such as MySQL, SQL Server, data centers can be the stream

processing technologies that process data in real-time such as InfoSphere Streams, TIBICO, or streaming data warehouse DataDepot [Golab 2009].

We consider a network of M sensors randomly deployed in a region of interest. Let $S = \{S_1, ..., S_M\}$ be time series data streams of observations arriving continuously from M sensors in the above sensor network. If a change occurs in the environment close to some node in the sensor network, the tasks of the sensor network are to detect and to report the changes of the environment as quickly as possible by using a small amount of memory, yet to assure some certain accuracy. We note that in the wildfire warning system, only a portion of nodes sense, and detect the events. To detect a fire event, only a portion of nodes detects the abnormal changes of the measures such as temperature, or humidity.

1.1.2 Challenges and Solutions

Designing and developing a change detector in a wireless sensor network face the following main challenges. First, the data streams produced by wireless sensor networks must be processed in real-time, which is a special case of change detection in distributed data streams. Second, processing data streams on sensor nodes has to consider resource limitations of sensor nodes, such as low power batteries, small memory, and limited communication resources. Third, effective schemes should be adopted to overcome local detection failures. Last by not least, different applications poses different requirements. For example, a wireless sensor network for fire warning system requires to propagate the detected event to multiple locations in the network timely. This requirement results in a multi-source multi-sink problem with time constraints. The challenges facing change detection in streaming data include the challenges of the change detection problem and those of data stream processing model.

Change detection has been studied and applied in many fields for a long time. Each research field has solved the problem of change detection by developing its own conceptual framework based on each specific context of application and how the concept "change" is understood [Dasu 2009]. The change detection algorithms are therefore diverse. For example, in the statistical signal processing, it is often stated as the problem of detecting signal in noise. In statistics, it has been considered in terms of the problem of hypothesis testing and the problem of change point detection. In general, there is no unified and universal framework for change detection that can be used for many disciplines.

Methods for detecting changes must be scalable with very large data sets, be able to adapt to streaming context, to apply to small data sets as well, to deal with multidimensional data, to capture spatial relationships and correlations. Furthermore, due to the restricted resources, when designing the change detection schemes for streaming data stream, we must consider the systems resources such as memory,

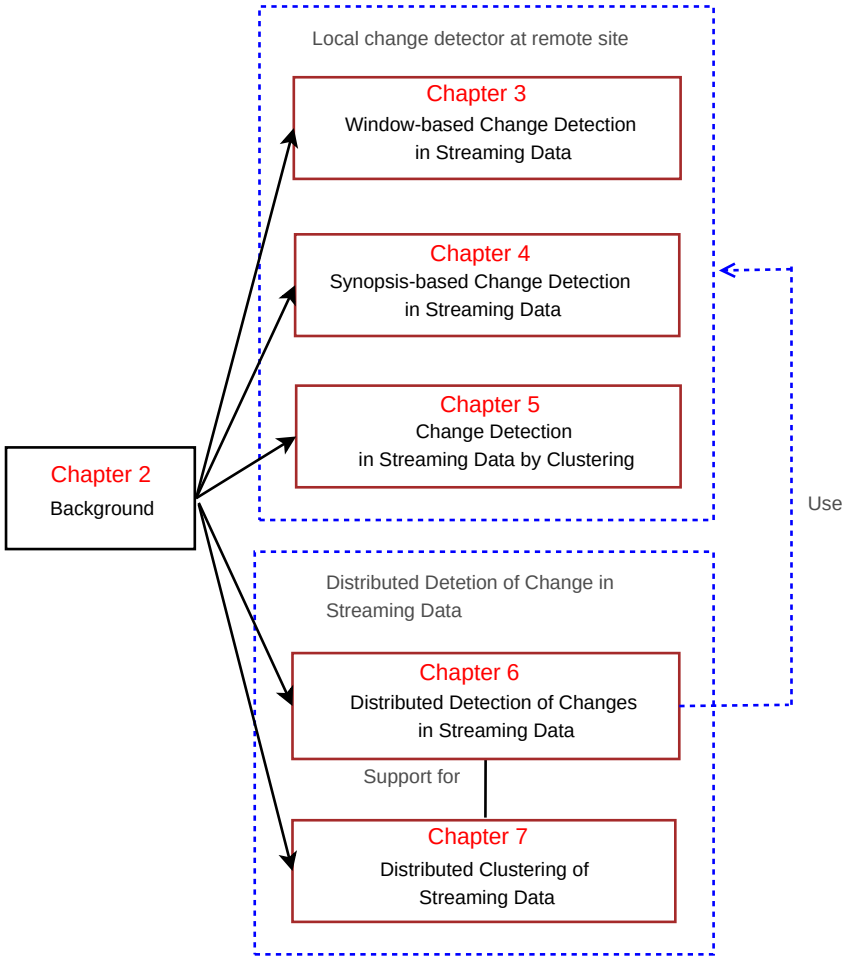


Figure 1.3: Relationships among the chapters

computing power, or energy consumption [Gama 2007].

Wireless sensor networks for monitoring outdoor environment requires reactivity, reliability, robustness, and network lifetime. In fact, sensor data arrives as a continuous and fast data stream. Thus, change detection in streaming sensor data works under resource-limited constraints. The challenges facing sensor data stream processing originate from the restrictions of sensor resources [Gama 2007].

Figure 1.3 shows the relationships among the chapters in this dissertation. This dissertation contains 3 parts. Part I consists of Chapter 1 and Chapter 2. Chapter 2 introduces the problem of change detection and the the concepts of change and change detection in streaming data.

Part II presents the change detection methods in a single data stream. This part consists of the following chapters

- Chapter 3 presents a window-based change detection in streaming data. Change detection in streaming data is challenging due to the time-evolving

and theoretically unlimited nature of the data stream. As data stream is unlimited in theory and the memory of sensor is limited, we use two-window model to quantify, and to detect changes in streaming data. Furthermore, as the value of data decreases overtime, we are only interested in the recent streaming data. A change detection method should maximize the probability of detection while minimizing the false alarm rate. A local change detector can be used as an onboard change detector at each sensor. Local change detection can reduce communication cost and prolong battery life. While traditional change detection methods can scan data sets many times, the change detection methods in streaming data must be limited in the number of scans in data sets due to the theoretically infinite nature of data streams and the restriction of systems resources. One-pass change detection algorithms are preferred. An advantage of local computation is energy-efficiency. Although windows are used to extract data, in many cases, the window width is too large to store and process in memory. We then propose to summarize data in windows as compact synopses, quantify and detect the changes in these compact summaries.

- Chapter 4 introduces a general framework for change detection in a single data stream by using synopses that summarize data. By this way, the resources such as time, memory needed for change detection in streaming data may reduce. While change detection methods in Chapter 3 and 4 have been focused on the univariate streaming data, many change detection methods in real world applications should be considered in the correlation of multiple attributes. Therefore, change detection in multivariate streaming data becomes important.

- Chapter 5 proposes an automated change detection algorithm in multivariate streaming data by clustering. Sensor networks need the automated change detection methods in which detection threshold must adapt to these changes of the environment. Automated systems should detect the changes without the given detection threshold. A stream clustering algorithm should be able to adapt to the changes in data distribution. Besides, we present a method for building and maintaining clustering over sliding window. In the reactive approach, the clustering is rebuilt when a change in the data distribution is detected. Change detection in streaming data is closely related to the problem of clustering streaming data. Recently, some change detection methods in streaming data by clustering have been proposed [Aggarwal 2012, Gaber 2006, CHEN 2009]. Gaber and Yu have presented STREAM-DETECT to capture the changes in data distribution and domain based on the clustering method. Their approach includes two phases: online clustering and off-line classification [Gaber 2006]. The goal of the first phase

is to detect changes based on determining the changes of clustering features such as cluster mean, cluster standard deviation, mean size. Aggarwal and Yu study change detection methods using micro-clustering [Aggarwal 2006]. Closely related to our work is the segment-based change detection method in multivariate data stream proposed by Chen et al. [CHEN 2009].

Change detection methods presented in Part II can be used as a component of a distributed framework for detecting changes in Part III. Part III presents the distributed frameworks for detecting and monitoring the changes in streaming data that comes from multiple sources. In particular, chapter 6 and 7 is concerned with the problem of continuous monitoring multiple data streams.

- Chapter 6 proposes a distributed framework for detecting changes in streaming data that comes from multiple sources. Change detection from sensor data is really decentralized change detection from multiple distributed data streams due to the distributed nature of a sensor network. We present a distributed framework for detecting changes in streaming data by using local change detection at the remote sites, and decision fusion at the coordinator. The goal of our distributed framework for detecting changes in streaming data is to reduce the communication overhead , yet to assure the detection accuracy. Due to the limited bandwidth of the communication channel, and limited power battery of sensor, each sensor sends its decision represented by a compact data structure to the fusion center. All the local change detectors use an identical local threshold.

  There are two approaches to distributed detection of changes in streaming data. In the first approach called data fusion, the sink node is responsible for fusing the data sent by all the nodes in the network, change detection is then performed on the fused data. Compared to data fusion, decision fusion avoids the congestion of network traffic better by transmitting the local decisions resulted from every change detector to the sink node at which local decisions are fused to reach the global decision. We assume that each node in the sensor network has reached its local decision u_i which is the local decision of the local change detector at node i. The local decision at node i is propagated to the other nodes of the sensor network, based on the received information from node i, these nodes also make local decision themselves. All the decisions are transmitted to the sink node in order to get the global decision. The goals and performance metrics for a distributed detection of changes in streaming data include communication overhead, promptness, and detection accuracy. Since sensor networks operate over wireless networks with limited bandwidth while transmission rates are very high and all the nodes share a common bandwidth, communication efficiency is therefore a challenge. In fact, the communication efficiency is closely related to the energy

efficiency because communication over a sensor network consumes a considerable amount of energy [Klan 2011]. In chapter 5, we describes how to detect changes in streaming multivariate data, hence in Chapter 7, we present a distributed clustering method for streaming data, and then extend it for the distributed detection of changes in streaming multivariate data. In particular, remote sites will summarize streaming multivariate data by using online phase, and send it to the coordinator, coordinator will detect the changes by using the clustering-based change detection presented in Chapter 5.

- Chapter 7 introduces a novel distributed framework for clustering streaming data by using local micro-clustering approach. The energy consumption is one of the main design desiderata when devising query processing strategies for sensor networks. Clustering of distributed streaming aims to support for the following tasks: monitoring of changes using clustering; clustering sensor nodes based on similarity of sensor readings; computing the sleeping time for a sensor node when it goes to the sleeping state. A sensor can be in sleeping state if it does not transmit data. As sensors near-by may have high correlations in their observation measures, therefore, in Chapter 7 we propose a distributed framework for clustering streaming sensor data. One of the main goals is to prolong network lifetime. An approach to maximizing the network lifetime is switching sensor nodes between active and sleeping states. A sensor node goes to the sleeping node when the radio and the sensing are turned off. As sensor nodes in wireless sensor networks are moving , wireless sensor network application should manage a huge amount of GPS streaming data. One of the solutions to this challenge is to reduce the communication overhead by using the algorithms for clustering sensor nodes in senor network such as [Handy 2002, Younis 2004]. Clustering sensor nodes in sensor networks supports for power management and network routing. Another approach to clustering sensors in sensor networks based sensor data have been recently presented [Sagen 2010]. Sagen et al. have applied distributed stream clustering algorithm for computing sleeping time.

- Chapter 8 summarizes our main contributions and introduces the future work.

Figure 1.4 shows three dimensions of a solution to the problem of distributed detection of changes in streaming data: the communication solution, local change detection algorithm, and distributed detection and decision fusion. The salient feature of this framework is its flexibility. As we can select various solutions for each dimension,, there are many solutions to the problem of distributed change detection. For local change detection, we can select or develop various local change detection algorithms. In this thesis, we present the local change detectors such as the Euclidean-distance-based detector, the Manhattan distance-based detector,

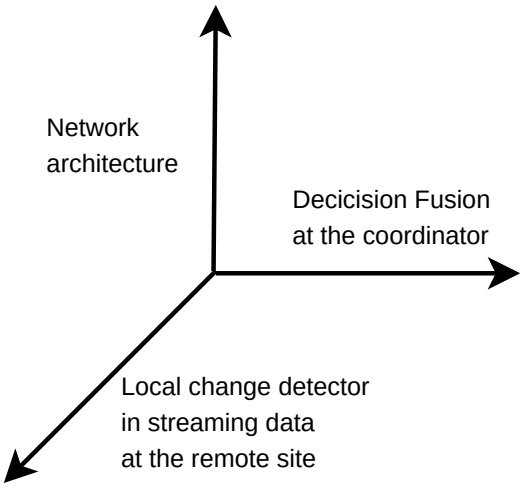


Figure 1.4: Three dimensions of the distributed detection of changes in streaming data

DFT-based change detector, and the change detector using clustering. For distributed change detection and decision fusion, we can also select or develop various algorithms for transmitting data, decisions, and making the decision on the event. Such distributed change detection algorithms include data flooding protocol [Juang 2002], gossiping-based fusion protocol [Yang 2011, Tran 2011b]. In this thesis, we propose three novel distributed change detection algorithms. The first algorithm is a distributed detection algorithm using gossiping scheme. The second algorithm is a distributed detection algorithm using the false alarm discovery rate. The advantage of this approach is that it does not require the probability of hit, and probability of false alarm. Therefore, it fits well to the real world applications in which these information are unknown. The third algorithm is a distributed change detection algorithm using reactive approach. In particular, it uses the clustering approach to identify whether a new data point or a block of new data points fit to the existing global clustering. The advantages of this approach are that it is an automated change detection method and it works for multivariate streaming data.

1.2 Contributions of the Dissertation

The contributions of this dissertation are built upon the wealth of research on both change detection and data stream processing. In particular, this dissertation makes the following contributions.

- A general framework for detecting changes in streaming data from a single source by using two fixed-size windows is proposed.

- A general framework for detecting changes in streaming data from a single source by using synopses constructed from two windows is proposed. It is proved that if the distance is preserved under some synopsis transformation then the detection performance of the change detectors using that synopsis is preserved. A concrete example illustrating this theorem is the change detector using Discrete Fourier Transform coefficients as synopses. The significance of incremental change detection is demonstrated by experiment with DFT-based change detection.

- An automated change detection algorithm using clustering is presented. Besides a reactive method for building maintaining clustering by using the above change detection method is proposed. The reactive approach to building and maintaining clustering means that a new clustering is only rebuilt when a change in incoming data is detected.

- A framework for detecting changes in streaming data coming from multiple sources is introduced. An instance of this framework using gossip-based scheme is simulated to illustrate the proposed framework. We presents a continuous distributed monitoring of changes and clustering from multivariate streaming data. The salient feature of continuous distributed monitoring of changes using the clustering approach is that it does not require the local detection threshold and the global detection threshold. We present a distributed framework using decision fusion for detecting changes without requiring the detection accuracy of local decisions.

- A distributed framework for clustering data streams is proposed. A widely used approach to clustering a single data stream is the two-phased approach in which the online phase creates and maintains micro-clusters while the offline phase generates the macro-clustering from the micro-clusters. We use this approach to propose a distributed framework for clustering streaming data. Every remote-site process generates and maintains micro-clusters that represent cluster information summary from its local data stream. Remote sites send the local micro-clusterings to the coordinator, or the coordinator invokes the remote methods in order to get the local micro-clusterings from the remote sites. Having received all the local micro-clusterings from the remote sites, the coordinator generates the global clustering by the macro-clustering method. Our theoretical and empirical results show that the global clustering generated by our distributed framework is similar to the clustering generated by the underlying centralized algorithm on the same data set. By using the local micro-clustering approach, our framework achieves high scalability, and communication-efficiency.

1.3 Related Work

Related work to this dissertation include change detection in a single data stream, distributed detection of changes, and building and maintaining clustering.

1.3.1 Change Detection in A Single Data Stream

Our change detectors are closely related to the meta algorithm proposed by Kifer et al. [Kifer 2004]. To detect change in a data stream, they compare the distributional distance between two sliding windows with a given threshold. Their approach requires no prior assumptions on the nature of the data distribution. They can compare two sliding windows of different sizes because the distance used by their algorithm is the Kolmogorov- Smirnov statistical distance. However, this algorithm is not well suited for sensor network applications because computing the Kolmogorov-Smirnov has high computation complexity and some other important limitations [Massey Jr 1951]. For example, the K-S distance applies only to continuous distributions. Perhaps, the most serious limitation of the K-S distribution is that it typically must be determined by simulation. Therefore, we have used in our work simple geometric and algebraic functions such as the Euclidean and Manhattan distances. As data streams evolve overtime in nature, there is growing emphasis on detecting changes not only in the underlying data distribution, but also in the models generated by data stream process and data stream mining.

In order to find a suitable solution to the change detection problem in resource-restricted settings of sensor networks, we have adopted the principle of work of Kifer et al. In our change detectors, the two-window paradigm is exploited, but we have improved this model by using the sliding window model first introduced by Zhu et al. [Zhu 2002].

Specially, Zhu et al. divided a large sliding window into smaller windows called basic windows. This division eases the evaluation of the change detectors. The distinction between their work and our work is that instead of computing the statistics and finding the correlations among data streams, we apply this sliding window framework to the problem of change detection. An important distinction among our change detection methods presented in this dissertation and others is that we can continuously evaluate the detection performance of change detection methods on the entire data stream while other work only evaluate on a snapshot of data stream. We use metrics from detection theory and information retrieval theory to evaluate and compare the detection performance of change detectors.

We should distinguish among three following problems: change detection in the underlying data distribution, change detection in model, and data stream mining with concept drift. First, change detection in the underlying data distribution is to find the difference among data sets over time based on some criteria such as the

dissimilarity measure between two data sets, or from the difference between two models constructed from two data sets to infer the difference between two data sets. Second, change detection in model is finding the difference between two models constructed from data sets overtime. Third, data stream mining with the presence of concept drift refers to the mining process whose result called concept changes overtime. As such, concept change refers to the change of the underlying concept over time. Concept drift can be classified into abrupt concept drift and gradual concept drift.

1.3.2 Reactive Monitoring

Building models from the continuous data streams aims to capture patterns and time-evolving trends in these streams. There are three approaches to building and monitoring models extracted from data streams [Bifet 2010a]. The first approach is called the periodic approach in which the model is rebuilt time to time. The second approach is the incremental approach in which the model is updated whenever the data changes. The benefits of incremental approaches are accurate and optimal. The third approach is called the reactive approach which monitors the change, and rebuild the model only when it no longer suits the data. Likewise, building and maintaining of model from distributed streaming data includes three approaches: the periodic approach, incremental approach, and the reactive approach [Bhaduri 2008]. Bhaduri et al. have shown the advantages and disadvantages of each approach as follows. The periodic approach may incur high cost in terms of resources and model accuracy. In particular, this approach may not well be suited for streaming evolving data. The best approach to building and maintaining model is the incremental approach. However, this incremental approach depends every specific problem. The reactive approach is widely used for many computation tasks because of its simplicity and efficiency. Chapter 5 presents a reactive method for monitoring of clustering in streaming data over sliding window. We then extends this method for distributed environment in Section 6.3.3 of Chapter 6.

1.3.3 Distributed Detection of Changes in Streaming Data

Over the last decades, the problem of decentralized detection has received much attention. There are two directions of research on decentralized detection. The first approach focuses on aggregating measurements from multiple sensors to test a single hypothesis. The second focuses on dealing with multiple dependent testing/estimation tasks from multiple sensors [Rajagopal 2008]. Distributed change detection usually involves a set of sensors that receive observations from the environment and then transmit those observations back to fusion center in order to reach the final consensus of detection. Decentralized detection and data fusion are

therefore two closely related tasks that arise in the context of sensor networks [Niu 2006b, Niu 2006a]. Two traditional approaches to the decentralized change detection are data fusion, and decision fusion. In data fusion, each node detects change and sends quantized version of its observation to a fusion center responsible for making decision on the detected changes, and further relaying information. In contrast, in decision fusion, each node performs local change detection by using some local change algorithm and updates its decision based on the received information and broadcasts again its new decision. This process repeats until consensus among the nodes are reached. Compared to data fusion, decision fusion can reduce the communication cost because sensors need only to transmit the local decisions represented by small data structures. Although there is great deal of work on distributed detection and data fusion, most of work focuses on the one-time change detection solutions. One-time query is defined as a query that needs to proceed data once in order to provide the answer [Cormode 2005]. Likewise, one-time change detection method is a change detection that requires to proceed data once in response to the change occurred. In real-world applications, we need the approaches capable of continuously monitoring the changes of the events occurring in the environment. Recently, work on continuous detection and monitoring of changes has been started receiving attention such as [Palpanas 2003, Das 2009, Pham 2012]. Das et al. [Das 2009] have presented a scalable distributed framework for detecting changes in astronomy data streams using local, asynchronous eigen monitoring algorithms. Closely related to our work is that of Palpanas et al. [Palpanas 2003], which proposed a distributed framework for outlier detection in real-time data streams. In their framework, each sensor estimates and maintains a model for its underlying distribution by using kernel density estimators. However, they did not show how to reach the global detection decision. The important distinction between our work and theirs is that instead of comparing the model extracted from the window with the norm, we quantify and detect changes in a single data stream by using two-window method. Depending on the position of the reference and current windows, and the selection of various distances, we can develop many change detection methods.

1.4 Summary

Change detection in streaming data is an exciting and challenging area of research, and a promising source of research problems.Successful applications of sensor networks will mostly depend on the ability to detect changes of change detection algorithms. This dissertation provides initial steps towards that target.

CHAPTER 2

Background

All things flow, everything runs, as the waters of a river, which seem to be the same but in reality are never the same, as they are in a state of continuous flows (The doctrine of Heraclitus)

Contents

2.1 Introduction

This chapter presents the background issues and notation relevant to the problem of change detection in data streams. As in the real world applications, it is difficult to determine the data distributions before and after change, this dissertation emphasizes the nonparametric change detection methods. The problems relevant to the evolving characteristics of data streams include building a model from the evolving data stream, representation of change, detecting changes in the data generating process, or the model generated from the data stream. We briefly review these problems in the sections below.

2.2 Change Detection in Streaming Data

Streaming computational model is considered one of the widely-used models for processing and analyzing massive data. Streaming data processing helps the decision-making process in realtime. A data stream is defined as follows.

Definition 2.2.1. *A data stream is an infinite sequence of elements*

$$S = \{(X_1, T_1), .., (X_j, T_j), ...\} \tag{2.2.1}$$

Each element is a pair (X_j, T_j) *where* X_j *is a* d*-dimensional vector* $X_j = (x_1, x_2, ..., x_d)$ *arriving at the time stamp* T_j*. Time-stamp is defined over discrete domain with a total order. There are two types of time-stamps: explicit time-stamp is generated when data arrive; implicit time-stamp is assigned by some data stream processing system.*

Streaming data includes the fundamental characteristics as follows. First, data arrives continuously. Second, streaming data evolves overtime. Third, streaming data is noisy, corrupted. Forth, timely interfering is important. From the characteristics of streaming data and data stream model, data stream processing and mining pose the following challenges. First, as streaming data arrives rapidly, the techniques of streaming data process and analysis must keep up with the data rate to prevent from the loss of important information as well as avoid data redundancy. Second, as the speed of streaming data is very high, the data volume overcomes the processing capacity of the existing systems. Third, the value of data decreases over time, the recent streaming data is sufficient for many applications. Therefore, one can only capture and process the data as soon as it is generated.

2.2.1 Change Detection: Definitions and Notation

This section presents concepts and classification of changes and change detection methods. To develop a change detection method, we should understand what a change is.

Definition 2.2.2. *Change is defined as the difference in the state of an object or phenomenon over time and/or space [Roddick 2000].*

In the view of system, change is the process of transition from a state of a system to another. In other words, a change can be defined as the difference between an earlier state and a later state. An important distinction between change and difference is that a change refers to a transition in the state of an object or a phenomenon overtime while the difference means the dissimilarity in the characteristics of two objects. A change can reflect the short-term trend or long-term

trend. For example, a stock analyst may be interested in the short-term change of the stock price. Throughout this dissertation, we mainly focus on the short-term change by using the window-based change detection methods. Change can be classified based on the magnitude of change, velocity of change, frequency of change. A change can be categorized into gradual change or abrupt change. An example of abrupt change is the change of temperature that is caused by a fire. Small change is defined as a change whose change magnitude per time unit is small. Small changes can be easily detected by sensors such as the light automatically turns on when the door opens. For example, detection of small changes plays an vital role in many applications such as flutter detection for analyzing the instability phenomenon for aircraft, or forest cover change detection [Zouari 2008]. As can be seen in Chapter 3, 4, 5, gradual change can be detected by using overlapping windows model with the small step of sliding. A change can be also classified into rare change or frequent change. A rare change can be an outlier, or anomaly. Periodic changes such as the seasonal changes in the climate model, or the changes in ECG signal. For example Tao and Özu present method for detecting periodic changes in streaming data [Tao 2009]. Chapter 4 will present a method for detecting changes by using Discrete Fourier Transform coefficients constructed from raw streaming data. DFT-based change detection method is well-suited for these periodic changes such the changes in ECG signal, and sensor data stream.

Change detection is defined as the process of identifying differences in the state of an object or phenomenon by observing it at different times [Singh 1989]. In the above definition, a change is detected on the basis of differences of an object at different times without considering the differences of an object in locations in space. In many real world applications, changes can occur both in terms of both time and space. For example, multiple spatial-temporal data streams representing triple (latitude, longitude, time) are created in traffic information systems using GPS [Geisler 2010]. Hence, a general definition of the change detection can be as follows.

Definition 2.2.3. *Change detection is the process of identifying differences in the state of an object or phenomenon by observing it at different times and/or different locations in space.*

A distinction between concept drift detection and change detection is that concept drift detection focuses on the labeled data while change detection can deal with both labeled and unlabeled data. Change analysis both detects and explains the change. Hido et al. [Hido 2008] proposed a method for change analysis by using supervised learning.

Definition 2.2.4. *Change point detection is identifying time points at which properties of time series data change[Kawahara 2009]*

Depending on specific application, change detection can be called in different terms such as burst detection, outlier detection, or anomaly detection. Burst detection a special kind of change detection. Burst is a period on stream with aggregated sum exceeding a threshold [Karnstedt 2009]. Outlier detection is a special kind of change detection. Anomaly detection can be seen as a special type of change detection in streaming data.

To find a solution to the problem of change detection, we should consider the aspects of change of the system in which we want to detect. As shown in [Roddick 2000], the following aspects of change, which must be considered, include subject of change, type of change, cause of change, effect of change, response of change, temporal issues, and spatial issues. In particular, to design an algorithm for detecting changes in sensor streaming data, the major questions we need to answer include: What is the system in which the changes need to be detected? What are the principles used to model the problem? What is data type? What are the constraints of the problem? What is the physical subject of change? What is the meaning of change to the user? how to respond and react to this change?

A change detection method can fall into one of two types: batch change detection and sequential change detection. Given a sequence of N observations $x_1, .., x_N$, where N is invariant. The task of a batch change detection method is to decide whether a change occurs at some point in the sequence by using all N available observations. When the arriving speed of data is too high, batch change detection is suitable. In other words, change detection method using two adjacent windows model will be used. However, the drawback of batch change detection method is that its running time is very large when detecting changes in a large amount of data. In contrast, the sequential change detection problem is based on the observations so far. If no change is detected, the next observation is processed. Whenever a change is detected, the change detector is reset.

Change detection methods can be classified into the following approaches: threshold-based change detection method; state-based change detection method; trend-based change detection method. In Chapter 3, and 4, we present threshold-based change detection methods. A change detection algorithm should meet three main requirements [Liu 2010]: accuracy, promptness, and online. The algorithm should detect as many as possible actual change points and generate as few as possible false alarms. The algorithm should detect change point as early as possible. The algorithm should be efficient sufficient for a real time environment. The change detection methods using two overlapping windows proposed in Chapter 3, 4, 5 can meet these requirements.

Change detection in data stream allows us to identify the time-evolving trends, and time-evolving patterns. Research issues on mining changes in data streams include modeling and representation of changes, change-adaptive mining method, and interactive exploration of changes [Dong].

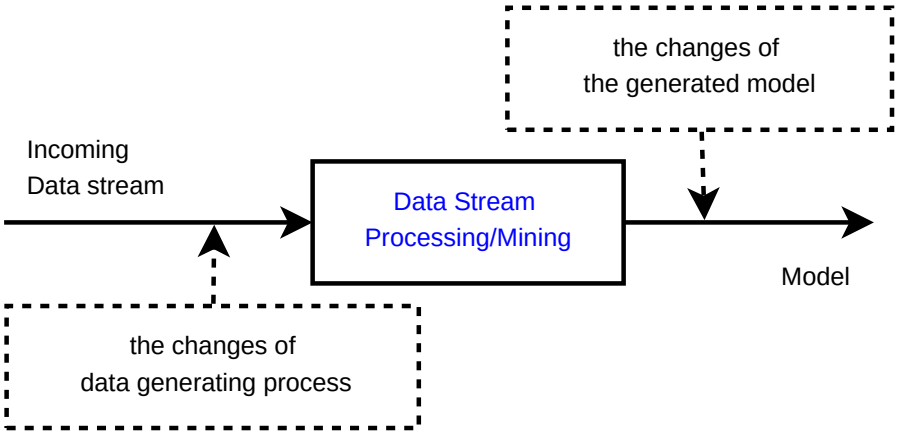


Figure 2.1: A general diagram for detecting changes in data stream

Change detection plays an important role in the field of data stream analysis. Since change in model may convey interesting time-dependent information and knowledge, the change of the data stream can be used for understanding the nature of several applications. Basically, interesting research problems on mining changes in data streams can be classified into three categories: modeling and representation of changes, mining methods, and interactive exploration of changes. Change detection algorithm can be used as a sub-procedure in many other data stream mining algorithms in order to deal with the changing data in data streams [Huang 2003, Aggarwal 2007]. A definition of change detection for streaming data is given as follows

Definition 2.2.5. *Change detection is the process of segmenting a data stream into different segments by identifying the points where the stream dynamics changes [Ross 2009].*

As data streams evolve overtime in nature, there is growing emphasis on detecting changes not only in the underlying data distribution, but also in the models generated by data stream process and data stream mining. As can be seen in Figure 2.1, a change can occur in the data stream, or the streaming model. Therefore, there are two types of the problems of change detection: change detection in the data generating process and change detection in the model generated by a data stream processing, or mining. The fundamental issues of detecting changes in data streams includes characterizing and quantifying of changes and detecting changes. A change detection method in streaming data needs a trade-off among space-efficiency, detection performance, and time-efficiency.

2.2.2 Change Detection Methods in Streaming Data

Over the last 50 years, change detection has been widely studied and applied in both academic research and industry. For example, it has been studied for a long time in the following fields: statistics, signal processing, and control theory. In recent years many change detection methods have been proposed for streaming data. The approaches to detecting changes in data stream can be classified as follows.

- ***Data stream model:*** A data stream can fall into one of the following models: time series model, cash register model, and turnstile model [Muthukrishnan 2005]. On the basis of the data stream model, there are change detection algorithms developed for the corresponding data stream model. Krishnamurthy et al presented a sketch-based change detection method for the most general streaming model Turnstile model [Krishnamurthy 2003]. As we select a wildfire warning system as a motivated scenario for thesis, we will focus on the problem of change detection in streaming data where data stream model is time series data stream. Because time series data stream is the most suitable for representing sensor data stream.

- ***Data characteristics:*** Change detection methods can be classified on the basis of the data characteristics of streaming data such as data dimensionality, data label, and data type. A data item coming from the data stream can be univariate or multi-dimensional. It would be great if we could develop a general algorithm able to detect changes in both univariate and multidimensional data streams. Change detection algorithms in streaming multivariate data have been presented [Dasu 2009, Kim 2009, Kuncheva 2011]. Chapter 3 and Chapter 4 are devoted to develop the methods for detecting changes in univariate streaming data. Chapter 5 presents a clustering-based method for detecting changes in multivariate streaming data. Data streams can be classified into categorial data stream and numerical data stream. We can develop the change detection algorithm for categorial data stream or numerical data stream. In real world applications, each data item in data stream may include multiple attributes of both numerical and categorial data. In such situations, these data streams can be projected by each attribute or group of attributes. Change detection methods can be applied to the corresponding projected data streams afterwards. Data streams are classified into labeled data stream and unlabeled data streams. A labeled data stream is one whose individual example is associated with a given class label, otherwise, it is unlabeled data stream. A change detection algorithm that identifies changes in the labeled data stream is supervised change detection [Kim 2009, Bondu 2011], while one detecting changes in the unlabeled data stream is called unsupervised

change detection algorithm [Cabanes 2012]. The advantage of the supervised approach is that the detection accuracy is high. However, the ground truth data must be generated. Thus a unsupervised change detection approach is preferred to the supervised one in case the ground truth data is unavailable. All the change detection methods presented in this dissertation are unsupervised change methods.

- ***Completeness of statistical information:*** On the basis of the completeness of statistical information, a change detection algorithm can fall into one of three following categories. Parametric change detection schemes are based on knowing the full prior information before and after change. For example, in the distributional change detection methods, the data distributions before and after change are known [Muthukrishnan 2005, Muthukrishnan 2007]. A recently introduced method to detecting changes in order stock streams is a parametric method in which the distribution of stream of stock orders confide to the Poisson distribution [Liu 2010]. The advantage of parametric change detection approaches is that they can produce a higher accurate result than semi-parametric and nonparametric methods. However, in many real-time applications, data may not confine to any standard distribution, thus parametric approaches are inapplicable. Semi-parametric methods are based on the assumption that the distribution of observations belongs to some class of distribution function, and parameters of the distribution function change in disorder moments. Recently, Kuncheva [Kuncheva 2011] has proposed a semi-parametric method using a semi-parametric log-likelihood for testing a change. Nonparametric methods make no distribution assumptions on the data. Nonparametric methods for detecting changes in the underlying data distribution includes Wilcoxon, kernel method, Kullback-Leiber distance, and Kolmogorov-Smirnov test. Nonparametric methods can be classified into two categories: nonparametric methods using window [Kifer 2004]; nonparametric methods without using window [Ho 2007]. In this dissertation, we have paid particular attention to the nonparametric change detection methods using window because in many real-world applications, the distributions of both null hypothesis and alternative hypothesis are unknown in advance. Furthermore, we are only interested in recent data. A common approach to identifying the change is to comparing two samples in order to find out the difference between them, which is called two-sample change detection, or window-based change detection. As data stream is infinite, a sliding window is often used to detect changes. Window based change detection incurs the high delay [Liu 2010]. Window-based change detection scheme is based on the dissimilarity measure between two distributions or synopses extracted from the reference window and the current window.

- ***Speed of response:*** If a change detection method needs to react to the detected changes as fast as possible, the quickest detection of change should be proposed. Quickest change detection can help a system make a timely alarm. Timely alarm warning is benefit for economical. In some cases, it may save the human life such as in fire-fighting system. As can seen in Chapter 3, change detection methods using two overlapping windows can quickly react to the changes in streaming data while methods using adjacent windows model may incur the high delay. As change can be abrupt change or gradual change, there exists the abrupt change detection algorithm and gradual change detection algorithm [Nikovski 2010, Maslov 2012].

- ***Decision making methodology:*** Based on the decision making methodology, a change detection method can fall into one of the following categories: rank-based method [Kifer 2004], density-based method [Song 2007], information-theoretic method [Dasu 2005]. A change detection problem can be also classified into batch change detection and sequential change detection. Based on detection delay that a change detector suffers from, a change detection methods can fall into one of two following types: real-time change detection, and retrospective change detection. Based on the spatial or temporal characteristics of data, change detection algorithm can fall into one of three kinds: spatial change detection; temporal change detection; or spatio-temporal change detection [Boriah 2008].

- ***Application:*** On the basis of applications that generate data streams, data streams can be classified as into transactional data stream, sensor data stream, network data stream, stock order data stream, astronomy data stream, video data stream, etc. Based on the specific applications, there are the change detection methods for the corresponding applications such as change detection methods for sensor streaming data [Tran 2011a], change detection methods for transactional streaming data [Ng 2008b, van Leeuwen 2008, Chang 2003]. For example, van Leeuwen and Siebes [van Leeuwen 2008] have presented a change detection method for transactional streaming data based on the principle of Minimum Description Length.

- ***Stream processing methodology:*** Based on methodology for processing data stream, a data stream can be classified into online data stream and off-line data stream [Manku 2002]. In some work, an online data stream is called a live stream while an off-line data stream is called archived data stream [Dindar 2011]. Online data stream needs to be processed online because of its high speed. Such online data streams include streams of stock ticker, streams of network measurements, and streams of sensor data,etc. Off-line stream is a sequence of updates to warehouses or backup devices. The queries over the

off-line streams can be processed off-line. However, as it is insufficient time to process off-line streams, techniques for summarizing data are necessary. In off-line change detection method, the entire data set is available for the analysis process to detect the change. The online method detects the change incrementally based on the recently incoming data item. An important distinction between off-line method and online one is that the online method is constrained by the detection and reaction time due to the requirement of real-time applications while the off-line is free from the detection time, and reaction time. Methods for detecting changes can be useful for streaming data warehouses where both live streams of data and archived data streams are available [Golab 2009, Huang 2009]. In this work, we focus on developing the methods for detecting changes in online data streams, in particular, sensor data streams.

A change can occur in the data stream, or in the model generated by some data stream processing and mining algorithm. Besides a streaming algorithm can also evolve in order to adapt to the changes. The issues of the evolution in the data streams can be classified into three following categories [Huang 2003, Aggarwal 2007]: The first work on model-based change detection proposed by [Ganti 1999, Ganti 2002] is FOCUS. The central idea behind FOCUS is that the models can be divided into structural and measurement components. To detect deviation between two models, they compare specific parts of these corresponding models. The models obtained by data mining algorithms includes frequent item sets, decision trees, and clusters. The change in model may convey interesting information or knowledge of an event or phenomenon. Model change is defined in terms of the difference between two set of parameters of two models and the quantitative characteristics of two models. As such, model change detection is finding the difference between two set of parameters of two models and the quantitative characteristics of these two models. We should distinguish between detection of changes in data distribution by using models and detection of changes in model built from streaming data. While model change detection aims to identify the difference between two models, change detection in the underlying data distribution by using models is inferring the changes in two data sets from the difference between two models constructed from two data sets. The changes in the underlying data distribution can induce the corresponding changes in the model produced from the data generating process.

As models can be generated by statistics method or data mining methods, change detection in models can be classified into data mining model and statistical model. Two kinds of models we are interested in detecting changes are predictive model and explanatory model. Predictive model is used to predict the changes in the future. Detecting changes in the pattern can be beneficial for many applica-

tions. In explanatory model, a change that occurred is both detected and explained. There are some approaches to change detection: one-model approach, two-model approach, or multiple-model approach. This dissertation deals with the problem of change detection using the two-model approach. We introduce a general framework for detecting changes in model.

A model-based change detection algorithm consists of two phases as follows: model construction and change detection. First, a model is built by using some stream mining method such as decision tree, clustering, frequent pattern. Second, a difference measure between two models is computed based the characteristics of the model, this step is also called the quantification of model difference. Therefore, one fundamental issue here is to quantify the changes between two models and to determine criteria for making decision whether and when a change in the model occurs. Based on the data stream mining model, we may have the corresponding problems of detecting changes in model as follows. Ikonomovska et al. [Ikonomovska 2009] have presented an algorithm for learning regression trees from streaming data in the presence of concept drifts. Their change detection method is based on sequential statistical tests that monitoring the changes of the local error, at each node of tree, and inform the learning process of the local changes.

Detecting changes of stream cluster model has been received increasing attention. Zhou et al. [Zhou 2008] have presented a method for tracking the evolution of clusters over sliding windows by using temporal cluster features and the exponential histogram, which called exponential histogram of cluster features. Chen and Liu [Chen] have presented a framework for detecting the changes in clustering structures constructed from categorial data streams by using hierarchial entropy trees to capture the entropy characteristics of clusters, and then detecting changes in clustering structures based on these entropy characteristics.

Based on the data stream mining model, we may have the corresponding problems of detecting changes in model as follows [Dasu 2009]. Recently Ng and Dash [Ng 2008a] have introduced an algorithm for mining frequent patterns from evolving data streams. Their algorithm is capable of updating the frequent patterns based on the algorithms for detecting changes in the underlying data distributions. Two windows are used for change detection: the reference window and the current window. At the initial stage, the reference is initialized with the first batch of transactions from data stream. The current window moves on the data stream and captures the next batch of transactions. Two frequent item sets are constructed from two corresponding windows by using the Apriori algorithm. A statistical test is performed on two absolute support values that are computed by the Apriori from the reference window and current window. Based on the statistical test, the deviation can be significant or insignificant. If the deviation is significant then a change in the data stream is reported. Chang and Lee [Chang 2003] have presented a method

for monitoring the recent change of frequent item sets from data stream by using sliding window.

2.2.3 Design Methodology

There are two design methodologies for developing the change detection algorithms in streaming data. The first methodology is to adapt the existing change detection methods for streaming data. However, many traditional change detection methods cannot be extended for streaming data because of the high computational complexity such as some kernel-based change detection methods, and density-based change detection methods. The second methodology is to develop new change detection methods for streaming data.

There are two common approaches to the problem of change detection in streaming data distributions: distance-based change detectors and predictive model-based change detectors. In the former, two windows are used to extract two data segments from the data stream. The change is quantified by using some dissimilarity measure. If the dissimilarity measure is greater than a given threshold then a change is detected. Similar to distance-based change detectors, two windows are used for detecting changes. Instead of comparing the dissimilarity measure between two windows with a given threshold, a change is detected by using the prediction error of the model built from the current window and the predictive model constructed from the reference window.

2.3 Distributed Change Detection in Streaming Data

Knowledge discovery from massive amount of streaming data can be achieved only when we could develop the change detection frameworks that monitor streaming data created by multiple sources such as sensor networks, WWW [Das 2009]. The objectives of designing a distributed change detection scheme are maximizing the lifetime of the network, maximizing the detection capability, and minimizing the communication cost [Veeravalli 2012].

There are two approaches to the problem of change detection in streaming data that is created from multiple sources. In the centralized approach: all remote sites send raw data to the coordinator. The coordinator aggregates all the raw streaming data that is received from the remote sites. Detection of changes is performed on the aggregated streaming data. In most cases, communication consumes the largest amount of energy. The lifetime of sensors therefore drastically reduces when they communicate raw measurements to a centralized server for analysis. Centralized approaches suffer from the following problems: communication constraint, power consumption, robustness, and privacy. Distributed detection of changes in stream-

ing data addresses the challenges that come from the problem of change detection, data stream processing, and the problem of distributed computing. The challenges coming from the distributed computing environment are as follows

- Distributed change detection in streaming data is a problem of distributed computing in nature. Therefore, a distributed framework for detecting changes should meet the properties of distributed computing such scalability, and fault tolerance. The scalability refers to the ability to extend the size of the network without significantly reducing the performance of the framework. As faults may occur due to the transmission error and the effects of noisy channels between local sensors and fusion center, a distributed change detection method should be able to tolerate these faults in order to assure the function of the system.

- Distributed change detection using the local approach is directly relevant to the problem of multiple hypotheses testing and data fusion because each local change detector needs to perform a hypothesis test to determine whether a change occurs. Therefore, besides considering the detection performance of local change detection algorithms including probability of detection and probability of false alarm at the node level, the detection performance of a distributed change detection method at the fusion center must be taken into account.

Distributed detection and data fusion have been widely studied for many decades. However, only recently, distributed detection in streaming data has received attention.

2.3.1 Distributed Detection: One-time versus. Continuous

Distributed detection of changes can be classified into two types of models as follows.

- One-time distributed detection of changes: Figure 2.2 shows two models of one-time distributed change detection. One-time change detection method is a change detection that requires to proceed data once in response to the change occurring. One-time distributed change detection have received a great deal of attention for a long time [Varshney 1997, Veeravalli 2012]. One-time distributed change detection includes two models: distributed detection without decision fusion as illustrated in Figure 2.2(a); distributed detection with decision fusion as shown in Figure 2.2(b).

- Continuous distributed detection of changes [Palpanas 2003, Das 2009]: Figure 2.3 depicts two models of continuous distributed detection of changes in

streaming data: distributed change detection without fusion and distributed change detection with fusion. In this context, data fusion from streams refers to the identification of the interesting relationships among multiple data sets. An important distinction between continuous distributed detection of changes and one-time distributed detection of changes is that the inputs to the one-time distributed change detection are batches of data while the inputs to the continuous distributed detection of changes are the data streams in which data items continuously arrive.

Although there is great deal of work on distributed detection and data fusion, most of work focuses on the one-time change detection solutions. In real-world applications, we need the approaches capable of continuously monitoring the changes of the events occurring in the environment. Recently, work on continuous detection and monitoring of changes has been received attention such as [Palpanas 2003, Das 2009, Tran 2011b, Tran 2011a]. Distributed detection model without fusion is a truly distributed detection model in which the decision-making process occurs at each sensor.

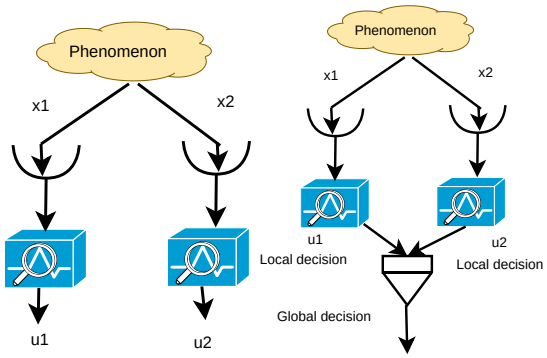
As a continuous query can be represented as a sequence of one-time queries which are invoked when the changes in the data sources are detected [Botan 2012], the result of a continuous change detection method can be also seen as an infinite sequence of the results of one-time change detection procedures. In summary, the result of a continuous distributed detection framework of changes can be considered the theoretically infinite sequence of the results of the one-time distributed change detection methods.

2.3.2 Locality in Distributed Computing

As one of the properties of distributed computational systems is locality [Naor 1993], a distributed algorithm for detecting changes in streaming data should meet the locality. A local algorithm is defined as one whose resource consumption is independent of the system size. The scalability of distributed stream mining algorithms can be achieved by using the local change detection algorithms

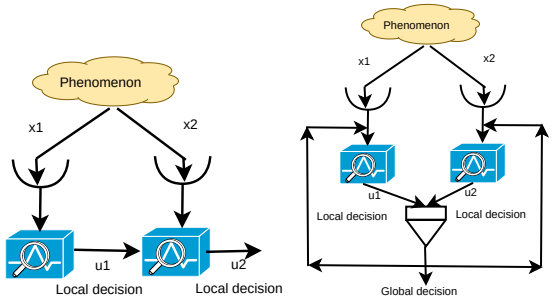
Local algorithms can fall into one of two categories [Datta 2006]:Exact local algorithms are defined as ones that produce the same results as a centralized algorithm; Approximate local algorithms are algorithms that produce approximations of the results that centralized algorithms would produce. Two attractive properties of local algorithms are scalability and fault tolerance. A distributed framework for mining streaming data should be robust to network partitions, and node failures.

The advantage of local approaches is the ability to preserve privacy [Ganguly 2008]. A drawback of the local approach to the problem of distributed change detection is the synchronization problem. For example, the local change



(a) Distributed detection without fusion

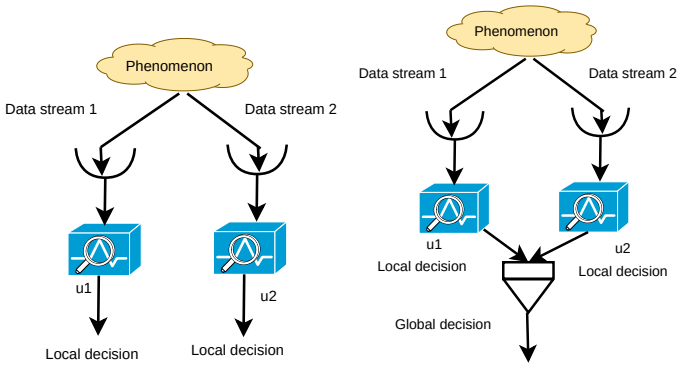
(b) Distributed detection with decision fusion



(c) Serial detection network

(d) Parallel fusion network with feedback

Figure 2.2: One-time distributed change detection models



(a) Distributed continuous detection without decision fusion

(b) Distributed continuous detection with decision fusion

Figure 2.3: Continuous distributed change detection models

approach can meet the principle of localized algorithms in wireless sensor networks in which data processing is performed at node-level as much as possible in order to reduce the amount of information to be sent in the network. In Chapter 6 and 7, we will use the local approach to develop the distributed frameworks for detecting changes of streaming data and clustering streaming data.

2.4 Efficiency Metrics

Detection performance is the key requirement of a change detection method. This section presents some metrics used to evaluate the performance of a change detection algorithm. A change detection method is often evaluated in terms of detection accuracy, temporal aspects such as detection time, reaction time, and memory consumption. Criteria for evaluating the detection performance of a change detection algorithm usually originate from detection theory and information retrieval.

2.4.1 Efficiency Metrics from Detection Theory

Change detection closely relates to the detection theory which formulates the criteria and methods for evaluating the performance of a detector [Macmillan 2005]. Therefore, some metrics of detection theory can be used to assess the detection performance of a change detector. Each observed event is represented by an instance consisting of multiple attributes. An observed event can be in one of two states: actual change, actual non-change. A change detector identifies the change of an observed event by mapping the corresponding instance into one of two states: detected change, or detected non-change.

A decision made by a change detector on the change of an event can fall into the following states:

- *Hit:* If the state of the event is actual change and the detector identifies the event as change. Hit is also called true positive.

- *Miss:* If the state of the event is actual change and the detector identifies the event as non-change. Miss is also called true negative.

- *False alarm:* If the state of the event is actual unchange and the detector identifies the event as change. False alarm is also called false positive.

- *Correct rejection:* If the state of the event is actual non-change and the detector identifies the event as non-change. Correct rejection is also called false negative.

	Detected change	Detected unchange	Total
Actual change	Hits	Misses	TAC
Actual unchange	False alarms	Correct rejections	TAU
	TDC	TDU	N

Table 2.1: The confusion matrix of the change detector

It is assumed that there are N observations of some event or phenomenon. Every event includes two states "actual change" and "actual non-change".

Table 2.1 shows the result of a change detector detecting the changes of N observations. Hits is the number of hit states. Misses is the number of miss states. False alarms is the number of false alarm states. Correct rejections is the number of correct rejection states.

Let TDC and TDU be the total detected change points and the total detected unchange points respectively. Let TAC and TAU be the total actual change points and the total actual unchange points respectively.

The probability of detection (hit rate, true positive rate, recall) is defined as the number of actual change points correctly detected as the changed points to the total actual change points. The hit rate of a change detector is estimated by

$$HR = Pr(\text{"}Yes\text{"}|Changed) = \frac{Pr(\text{"}Yes\text{"} \cap Changed)}{Pr(Changed)} = \frac{Hits}{Hits + Misses} \tag{2.4.1}$$

The hit rate is also called the power of a change detector.

The probability of false alarm (false positive rate) is defined as the number of actual unchanged points that are incorrectly detected as the change points to the number of unchange points. The false alarm rate of a change detector is computed by

$$FR = Pr(\text{"}Yes\text{"}|Unchanged) = \frac{Pr(\text{"}Yes\text{"} \cap Unchanged)}{Pr(Unchanged)} \tag{2.4.2}$$

Naturally, the rate of misses and the rate of correct rejection are computed by $MH = 1 - H, CR = 1 - FA$ respectively.

One of the widely used metrics of a detector is sensitivity. Let H denote the probability of a hit, and F the probability of a false alarm, then the change detector's sensitivity measure is a function of H and F. The sensitivity of a detector is given by

$$sensitivity = z(H) - z(F) \tag{2.4.3}$$

where the z transformation converts a hit or false-alarm rate to a z score. A perfectly sensitive change detector would have a hit rate of 1 and a false-alarm rate of 0.

	Detected changes	Detected unchange	Total
Actual change	132	18	150
Actual unchange	194	51629	51823
	326	51647	51973

Table 2.2: An example of confusion matrix of the Euclidean distance-based change detector

Specificity measures the proportion of unchange points that are correctly identified as such. Specificity is defined as follows

$$specificity = \frac{CR}{FA + CR} = 1 - FARate \tag{2.4.4}$$

where CR is the number of unchange points that are correctly detected by the change detector and FA is the number of unchange points that are reported as change points.

Detection delay is the duration of time, which is computed from the time at which a change occurs to the time at which a change is detected. A change detector is considered as an optimal detector if the detection delay is minimized, and with the given rate of false alarms, the rate of hits is maximized. The Neyman-Pearson lemma [Neyman 1933] says that the decision rule should be constructed in order to have the maximum probability of detection while not allowing the probability of false alarm to exceed a certain value α. In other words, the Neyman-Pearson criterion is the solution to the following constrained optimization problem: $max\{Pr(Hits)\}$, such that $Pr(FA) \leq \alpha$ where α is an user-specified upper bound of the probability of false alarm. As such, there are two kinds of errors that may occur in the detection result: false alarm, and missed detections.

From Table 2.2, we compute the hit rate and false alarm as follows. The hit rate is $\frac{132}{132+18} \simeq 0.88$. The false alarm rate is $\frac{194}{194+51629} \simeq 0.0037435$.

2.4.2 Efficiency Metrics from Information Retrieval

Performance metrics in the field of information retrieval used to evaluate a change detection algorithm includes Precision, Recall, and F-measure. *Precision* is defined as the probability that a point detected as 'changed' (or range) is truly a changed point. Precision is given by

$$Precision = \frac{Hits}{DCP} = \frac{Hits}{Hits + FA} \tag{2.4.5}$$

Recall is the probability that a change detector identifies a truly changed point. Recall is given by

$$Recall = \frac{Hits}{TCP} = \frac{Hits}{Hits + Misses} \tag{2.4.6}$$

A measure that expresses the tradeoff between precision and recall is $F-measure$ given by

$$F - measure = \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{2.4.7}$$

Let $Hits$ be the number of truly change points that is detected as change points by a change detection algorithm. Let CoR be the number of non-change points that is detected as non-change points by a change detection algorithm. The accuracy of a change detection algorithm is given by

$$Accuracy = \frac{Hits + CoR}{TCP + TNP} \tag{2.4.8}$$

As we will shows in the experiments in Chapter 3, accuracy is insufficient for evaluating the performance of a change detection algorithm.

From Table 2.2, we compute Precision, Recall, F-measure, and Accuracy as follows. Precision is $\frac{132}{132+194} \simeq 0.4049$. Recall is $\frac{132}{132+18} \simeq 0.88$. F-score is $\frac{1}{\frac{1}{0.4049}+\frac{1}{0.88}} \simeq 0.2773$

2.4.3 ROC and PR Curves

To compare the performance of different change detection algorithms, we must use or propose criteria. In this thesis, we use ROC curve and PR curve for evaluating and comparing the detection accuracy of the change detectors. In this section, we present the concepts of ROC curve and PR curve. ROC and PR curves are commonly used for evaluating the machine learning and data mining algorithms . In fact, the problem of change detection is the binary classification problem in which the task of the binary classifier is to identify two classes: "changed" and "unchanged". Therefore, we can apply the results of the binary classification problem to analyze the problem of change detection. [Davis 2006].

Definition 2.4.1. *ROC graphs are two-dimensional graphs in which hit rate (TP rate) is plotted on the Y axis and FP rate is plotted on the X axis. An ROC graph depicts relative trade-offs between benefits (hits or true positives) and costs (false alarms or false positives) [Davis 2006].*

The area under the Receiver Operating Characteristic curve (AUC stands for Area Under receiver operating Curve) is often used as criteria for comparing the performance of the change detection algorithm with other approaches. The overall performance of a change detector can be assessed by the receiver operating characteristics (ROC) curve.

A perfect change detector would generate a line from (0,0) to (1,1), which is called the "chance diagonal". A change detector can be considered a good change detector if its ROC curve is above the chance diagonal line. The larger the AUC of

a change detector is, the better the better its detection accuracy is. The computation and drawing of ROC curve is presented in detail in [Fawcett 2004]. However, if there is unbalanced in the class distribution, ROC curve may provides an overly optimistic view of an algorithm's performance [Davis 2006]. An alternative approach is using PR curve (Precision-Recall curve). Precision-Recall curve is usually used in Information Retrieval.

Definition 2.4.2. *PR graphs are two-dimensional graphs in which Precision is plotted on the Y axis and Recall is plotted on the X axis [Davis 2006].*

The goal we want to achieve in ROC space is the left-upper corner while the goal in PR space is the right-upper corner. Davis and Goadrich shows that, an algorithm that optimizes the area under the ROC curve is not guaranteed to optimize the area under the PR curve. Thus, it should be better to evaluate and compare the detection accuracy of the change detectors in both ROC space and PR space. In Section 3.6.3, we will use both ROC curve and PR curve for evaluating and comparing the detection accuracy of the detectors using the Euclidean and Manhattan distances.

2.5 Summary

This chapter has introduced the concepts that will be used throughout this thesis. We have briefly reviewed the data stream processing and mining model, streaming data and its characteristics as well as the challenges facing the techniques of data stream processing and mining. The concepts of change, and change detection have been then presented. A short survey on change detection methods for streaming data has been given. The metrics for evaluating and comparing the detection performance of the change detectors have been presented.

Part II

Change Detection in A Single Data Stream

CHAPTER 3

Window-based Change Detection in Streaming Data

All things must change to something new, to something strange
(Henry Wadsworth Longfellow "Keramos")

Contents

3.1 Introduction

As change detection methods should adapt to the changes in the environment, change detection methods presented in this thesis detect the changes by quantifying

the difference between two windows. If a change detection method that uses sensor reading for detecting changes cannot adapt to the changes in the environment because it needs the absolute threshold of detection. This chapter addresses the problem of change detection in streaming data that is created from a single source by using two-window approach in conjunction with many different measures of dissimilarity. In particular, the contributions of this chapter are as follows.

- We propose a general framework for detecting changes in streaming data by using sliding windows model. Based on the location correlation of two windows, we can develop the change point algorithms based on the two overlapping windows model, or the interval-based change detection algorithms based on the two adjacent windows model.
- We study the change detectors with the different dissimilarity measures including geometric and statistical dissimilarity measures in terms of detection accuracy, performance measures including time and memory needed to run a change detector.
- To our best of knowledge, an important distinction between our work in this chapter and other work is that our work is the first work evaluate the detection performance of a change detection on the entire data stream with the assumption that data stream is finite. Change detection methods presented this chapter already exists in many work in the literature such as work of Kifer et al. However, in Chapter 3, the first time, we evaluate the detection performance of change detectors on the entire data streams while other work only evaluate the detection accuracy a snap-shot of a data stream.

The tradeoff of a change detection method in a single data stream is computational complexity and detection performance. The rest of this chapter is organized as follows. Section 3.2 briefly introduces the data stream processing by using window technique, and sliding window. We then describe the change detection methods based on the location correlation of two windows: change detectors using adjacent windows model, change detectors using overlapping windows model, and change detectors using fixed-sliding windows model. In Section 3.3, we present a general criteria for determining whether a change occurs. Section 3.5 discusses selection of detection threshold. Section 3.6 evaluates the effect of window, dissimilarity measures, and threshold selection on the detection accuracy of a change detector. We summarize the chapter with major contributions in Section 3.7.

3.2 Window-based Change Detection

Since data stream is infinite in nature, a common approach is observing and processing it in a window. Based on the window width, a window can fall into two

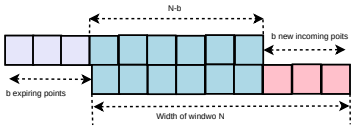


Figure 3.1: A window before and after sliding b steps

kinds: fixed-size window and variable-size window. Sliding window model is useful when we need to make decisions based on the recent observations such as stock data stream, sensor data streams. Since in most of streaming applications, we are only interested in the recent data or recent model,the sliding window is the most suitable choice. For example, in the sensor network applications, we are only interested in the recently observed data. Sliding windows can be classified as follows [Golab 2003]:

- *Direction of movement of the endpoints:* A fixed window has two fixed endpoints. A sliding window has two moving endpoints (either backward or forward). A landmark window has one fixed endpoint and one moving endpoint.

- *Physical vs. logical window:* Physical, or time-based windows are defined in terms of a time interval, while logical windows or count-based window are defined in terms of the number of tuples. Tuple-based window can react to primitive events as soon as possible. Stream engine StreamBase uses tuple-based window while Oracle CEP uses time-based window.

- *Update interval:* A window is updated upon arrival of each new tuple while batch processing leads to a 'jumping window'.

3.2.1 Sliding Window Model

Let window $w[t_c - N + 1, t_c]$ be a sliding window, where t_c is the current time. The window starts at the time $t_c - N + 1$ and ends at the current time t_c. An item is expired if its time stamp t is less than the time $t_c - N + 1$. An active item is defined as an item whose time stamp t lies in the range $[t_c - N + 1, t_c]$. Data items are inserted into the sliding window as they arrive, and the oldest (outdated, or stale) items are removed from it. A sliding window consists of two properties: window size and the slide step of the window. One of the goals of change detection is to detect the change with minimum delay. If the window size is large, the detection delay increases.

A data stream processing model can fall into one of two models: time-driven model or tuple-driven model. For example, the Oracle CEP model follows a time-driven model while processing model in StreamBase is tuple-driven model [Jain 2008, Dindar 2012]. Therefore, there also exists two corresponding kinds of

windows: tuple-based window, and time-based window. Jain et al. [Jain 2008] propose a new model that can exploit the best of two kinds of model. In this dissertation, we mainly focus developing the change detection methods for tuple-driven processing model.

3.2.2 Change Detection using Sliding Windows Model

Two-window model is widely used in many techniques for processing streaming data. Liarou et al. have recently proposed a method for incremental query processing using two overlapping windows [Liarou 2012, Liarou 2013]. Draisbach et al. have used two-window model to detect the duplication of a database [Draisbach 2012]. Bifet and Gavalda present a change detection method using adaptive window [Bifet 2007]. The window size automatically increases when no change occurs, the window size decreases when a change occurs.

The key idea underlying the window-based change detection methods is to compare two samples extracted from two windows: the reference window and the current window. Let w_1 and w_2 denote two basic windows of size b. The problem of change detection in data streams is to decide the null hypothesis $\mathcal{H}_0$ against alternative hypothesis $\mathcal{H}_1$ as below

$$\begin{cases} \mathcal{H}_0 & d(w_1, w_2) \leq \omega \\ \mathcal{H}_1 & d(w_1, w_2) > \omega \end{cases} \tag{3.2.1}$$

where $d(w_1, w_2)$ is a distance function which measures the dissimilarity of two sliding windows and ω is a distance-based threshold used to decide whether a change occurs. A change occurs if the dissimilarity measure between two windows exceeds a given threshold. Window-based methods for detecting changes may reduce the memory and time required to execute a change detector. Depending on the choice of window width and sliding step, the position correlation of two sliding windows can be fixed-sliding windows model, adjacent windows model, and disjoint windows model. Based on the selection of slide size, we can develop the different change detection methods. If the step size is greater than one, the change detector belongs to the batch incremental change detection. If the step size is one, the change detector belongs to the instance incremental change detection. Depending on how to choose the reference window, the position correlation between the reference window w_1 and the current window w_2, as well as how the windows w_1 and w_2 move, we can develop different algorithms for detecting changes.

The input to a two-window change detection algorithm consists of a data stream $S = \{x_1, x_2, ...\}$ and a given threshold ω. The output is a message reporting whether a change occurs, and the change point. The algorithm first assigns the first b points (data items) in the sliding window to the reference window w_1. The

next b points are set to the basic window w_2. A change has been considered to occur if the distance between two windows is greater than some threshold. When a change is detected, the change detector makes an alarm. The basic windows w_1 and w_2 continue capturing the new data items in the sliding windows. If a change does not occur, the basic window w_2 slides step by step until the change detected. An alarm is promptly issued at the end of each slide. Selection of window size plays a critical role in designing a window-based change detection scheme. The larger the window size is, the more accurate the change detection method is. However, the larger the window size is, the larger the detection delay is.

3.2.2.1 Adjacent Windows Model

If the sliding step is equal to the window width, two windows are adjacent. Two adjacent windows model is useful for detecting changes between two consecutive periods of time such as detection of changes of the temperature in Ilmenau between two consecutive months. Methods using adjacent windows model can detect cumulative changes in streaming data. Change detection methods using adjacent windows model can detect the changes between two consecutive time periods such as two consecutive days, two consecutive months, and two consecutive years. Kifer et al. [Kifer 2004] have presented a meta-algorithm for detecting changes in streaming data using fixed-sliding windows model. Algorithm 1 describes how

Algorithm 1: Change detection algorithm using adjacent windows model

input : A data stream S, a distance-based threshold d^*
output: The messages reporting the changes occurred

Step1:
begin
    $t \longleftarrow 0$;
    $w1 \longleftarrow$ first b points from time t;
    $w2 \longleftarrow$ next b points in the data stream;
Step2:
while *not at the end of the stream* **do**
    if $d(w1, w2) > d^*$ **then**
        $t \longleftarrow$ current time;
        Report change occurred at time t;
        Clear all windows and GOTO *Step 1*;
    else
        slide $w2$ by 1 point;

the change detection approach using adjacent windows model works in detail. We

call this algorithm adjacent windows model-based change detection because the reference window $w1$ and the current window $w2$ are adjacent after a change is detected. The detection delay depends on the computational complexity of the distance function and the window width. Change detection methods using adjacent windows model may suffer from the larger detection delay than the methods using overlapping windows model.

3.2.2.2 Overlapping Windows Model

If the sliding step is smaller than the window width, two windows are overlapping. Change detection methods using overlapping windows model can detect changes in streaming data incrementally. Sometimes, a full analysis of streaming data is likely impossible in real-time. We need to compute partial results so that a small amount of incremental computation with new data can be used to arrive at a quick decision. That is relevant to incremental computation. Algorithm 2 describes how

Algorithm 2: Change detection algorithm using overlapping windows model

input : A data stream S, a distance-based threshold d^*
output: The messages reporting the changes occurred

Step1:
begin
    $t \longleftarrow 0$;
    $w1 \longleftarrow$ first b points from time t;
    $w2 \longleftarrow$ slide $w1$ by 1 point in the data stream;
Step2:
while *not at the end of the stream* **do**
    if $d(w1, w2) > d^*$ **then**
        $t \longleftarrow$ current time;
        Report change occurred at time t;
        Clear all windows and GOTO *Step 1*;
    else
        Slide $w2$ by 1 point;

a change detection method using overlapping windows model works in detail. This algorithm is called overlapping windows model-based change detection because the reference window $w1$ and the current window $w2$ are overlapping after a change is detected. We note that if the window size is b then the overlapping part between the reference window and the current window is $b - 1$. By arranging such position correlation between two windows, Algorithm 2 can detect the change point. In

contrast, Algorithm 1 can only show the interval of change, that means the change point is unknown exactly.

We note that the distinction between Algorithm 1 and Algorithm 2 is only the init step. In Algorithm 1, two windows w1 and w2 are adjacent while in algorithm 2, two windows are overlapping by (b-1). In this case b is the window width. The init step is invoked once a change is detected.

3.2.2.3 Fixed-Sliding Windows Model

In this section, we present the change detection scheme in which the reference window is set to the given values while the current window moves on the data stream.

Algorithm 3: Change detection algorithm using fixed-sliding windows model

input : A sensor data stream S, a distance-based threshold d^*
output: The messages reporting the changes occurred

Step1:
begin
    $t \longleftarrow 0$;
    $w1 \longleftarrow b$ constant points;
    $w2 \longleftarrow$ first b points from time t;
Step2:
while *not at the end of the stream* **do**
    if $d(w1, w2) > d^*$ **then**
        $t \longleftarrow$ current time;
        Report change occurred at time t;
        Clear all windows and GOTO *Step 1*;
    else
        Slide $w2$ by 1 point;

Algorithm 3 describes a change detection method in which the reference window is assigned to the given values, for instance, a given temperature threshold (called raw threshold).

3.3 Change Detection Criteria

In this section, we study the change detection criteria or change detection rules. One of the commonly-used approach to designing a change detection method for streaming data is to use the statistical hypothesis testing in conjunction with the

data stream processing. Change detection can be considered the problem of hypothesis testing, where the hypothesis is a logical expression $change$ which indicates whether a change occurs.

$$\begin{cases} \mathcal{H}_0 & \neg change \\ \mathcal{H}_1 & change \end{cases} \tag{3.3.1}$$

In statistics test, the change condition is given by $change = (F_S \neq F_{S'})$, where F_S and $F_{S'}$ are two distributional functions that are corresponding to two data sets S and S'. Criteria for a good change detection method include: few false alarms, short delay for detection. Statistical hypothesis testing is of importance in data mining. In particular, many change detection methods are based on the statistical hypothesis testing. In this section, we briefly introduce the background of statistical hypothesis testing. Statistical hypothesis tests can be classified into the following categories: test of randomness,tests of goodness of fit, one-sample and paired-samples test, two-sample test, location test, scale test. We use two-sample to develop the change detection methods.

An one-sided hypothesis test can fall into one of the two following tests.

$$\begin{cases} \mathcal{H}_0 & m \leq m_0 \\ \mathcal{H}_1 & m > m_0 \end{cases} \tag{3.3.2}$$

or

$$\begin{cases} \mathcal{H}_0 & m \geq m_0 \\ \mathcal{H}_1 & m < m_0 \end{cases} \tag{3.3.3}$$

where m is a test statistic and m_0 is a test threshold. As such, one-sided hypothesis tests can be used for detecting the increase or decrease in the distribution compared with the given threshold. In some cases, we are only interested in the abruptly increasing change or the abruptly decreasing change. For example, sensors can be used to detect abrupt increase in pollutants in the air. The two-sided hypothesis is defined as follows

$$\begin{cases} \mathcal{H}_0 & m = m_0 \\ \mathcal{H}_1 & m \neq m_0 \end{cases} \tag{3.3.4}$$

The two-sided hypothesis tests can be used for detecting both the increase and decrease in the underlying data distribution.

Given two data sets X of size m and Y of size n with the corresponding distributions p and q. The statistical test $\mathcal{T}(X, Y) : \mathcal{X} \times \mathcal{X} \mapsto \{0, 1\}$ distinguish between the null hypothesis $\mathcal{H}_0$ and the alternative hypothesis $\mathcal{H}_1$. The null hypothesis is equivalent to the non-change state while the alternative hypothesis is equivalent to the change state.

The change can occur in the underlying data distribution or in the model constructed from the data. In order to detect changes of an object or a phenomenon,

we need to quantify the change. The problem of quantifying the change consists of quantifying the change of data distribution and quantifying the change of the model. In this section, we briefly review the dissimilarity measures used for quantifying and detecting changes.

One common approach to quantifying a change is to use dissimilarity metrics. A dissimilarity measure shows how dissimilar is a new data item from the data items seen sofar. To quantify the change in the statistical change detection approach,we need to select a distance between two distributions of two samples or a two-sample statistical test. Depending on the dimension of data, we choose the appropriate distance. In other words, the factors that affect the selection process of a change detection method include application context and the nature of data.

Definition 3.3.1. *Given the set (X), a metric on (X) is a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for $x, y, z \in \mathcal{X}$ it satisfies that*

- $d(x, y) > 0$
- $d(x, y) = d(y, x)$
- $d(x, y) = 0 \Leftrightarrow x = y$
- $d(x, z) \leq d(x, y) + d(y, z)$

Definition 3.3.2. *A distance function between two windows of size b is a mapping function $d : W \times W \to R$ such that for distinct i, j*

1. $d(w_i, w_j) = 0$ *if only if* $w_i = w_j$
2. $d(w_i, w_j) \geqslant 0$
3. $d(w_i, w_j) = d(w_j, w_i)$

A distance measure must truly quantify an intuitive notion of change so that change can be explained to non-technical users. The dissimilarity measure can be geometric distance measures, statistical measures, or information theoretic measures. We note that a change detection method cannot detect all types of changes. Therefore, depending on each type of change, one can select a suitable method.

Two aspects of the dissimilarity function should be considered: the detection accuracy and computation complexity. It would be ideal if a dissimilarity function could both quantify the changes accurately and the low computation complexity.

3.4 Dissimilarity Metrics

The success of many change detection algorithms depends on the selection of distance metric. Similarity and distance metrics are the central concepts in data mining. There are two kinds of dissimilarity metrics: geometric distance and statistical distance.

3.4.1 Geometric Dissimilarity Metrics

The most widely used distance is the Euclidean distance. The Euclidean distance is preferred to other distances, because it is preserved under orthogonal transforms such as Fourier transform. Additionally, the Manhattan distance is commonly used in some applications. Let $w_i = (x_0, x_1, ..., x_{b-1})$ and $w_j = (y_0, y_1, ..., y_{b-1})$ be two corresponding sets of coefficients on two basic windows moving in a sliding window, the Euclidean distance between two windows is defined as

$$d(w_i, w_j) = \sqrt{\sum_{k=0}^{b-1} (x_k - y_k)^2} \tag{3.4.1}$$

A commonly used distance measure is the Manhattan distance defined by

$$d(w_i, w_j) = \sum_{k=0}^{b-1} \mid x_k - y_k \mid \tag{3.4.2}$$

Chebychev distance between two windows w_i and w_j is given by

$$dist\,(w_i, w_j) = \max_k (\mid x_k - y_k \mid) \tag{3.4.3}$$

Figures 3.11 and 3.12 shows that the histogram of changes for the adjacent windows model using Euclidean distance is almost similar to the histogram of change for the adjacent windows model using Manhattan distance. In other words, the detection ability of the Euclidean distance is similar to that of the Manhattan distance. However, the computation complexity of the Manhattan distance is smaller than that of the Euclidean distance. The geometric measures of difference between two samples is more convenient than the statistical measures of change because of its simplicity and comprehensibility. In particular, recently Cohen and Kaplan [Cohen 2012] have proposed the novel approaches to estimating the changes from the samples for the Manhattan and Euclidean distances instead of computing these distances on the original data.

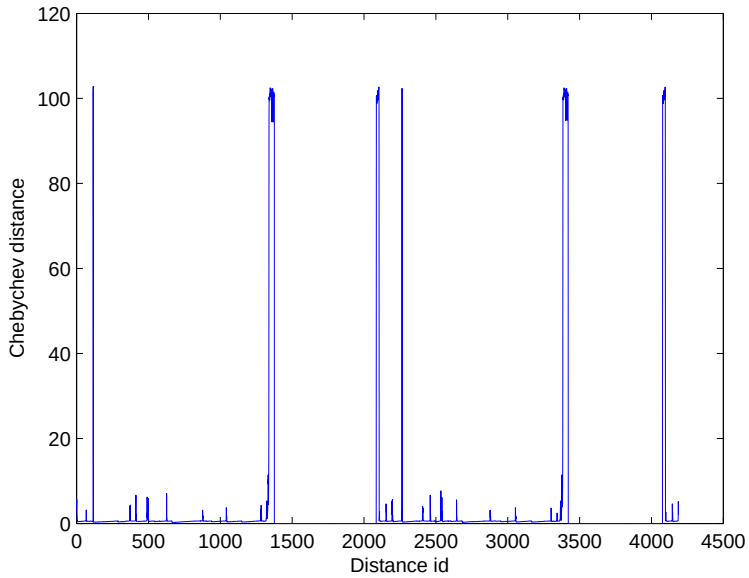


Figure 3.2: Chebychev dissimilarity measure between two adjacent windows

3.4.2 Statistical Dissimilarity Metrics

Due to the inherent uncertainty of data streams, the probabilistic and statistical approach is a natural option for much work on change detection in data stream. The major advantage of change detection methods using statistical hypothesis testing is that these methods can detect the statistically significant changes. Two problems we should considers include computational complexity and detection delay. In fact, the detection delay is consequence of computational complexity. Basically, there are two categories of distributional distance corresponding to single dimensional data and multidimensional data. Test statistic can fall into one of two kinds: distance-based test, and kernel-based test. Kernel is used as a measure of similarity between x and y Each hypothesis is characterized by a test statistic and p-value. The null hypothesis is rejected if the p-value is less than α. A two-sample hypothesis testing can be used to test whether a change occurs.

- *Single dimensional distribution measure:* Wilcoxin distance, Kolmogorov-Smirnov distance
- *Multidimensional distribution measure:* Kullback-Leiber distance. Song et al. [Song 2007] propose a method called density test for testing change in the multidimensional data.

A two-sample hypothesis testing can fall into two following types: parametric hypothesis testing and nonparametric hypothesis testing. In a parametric hypothesis

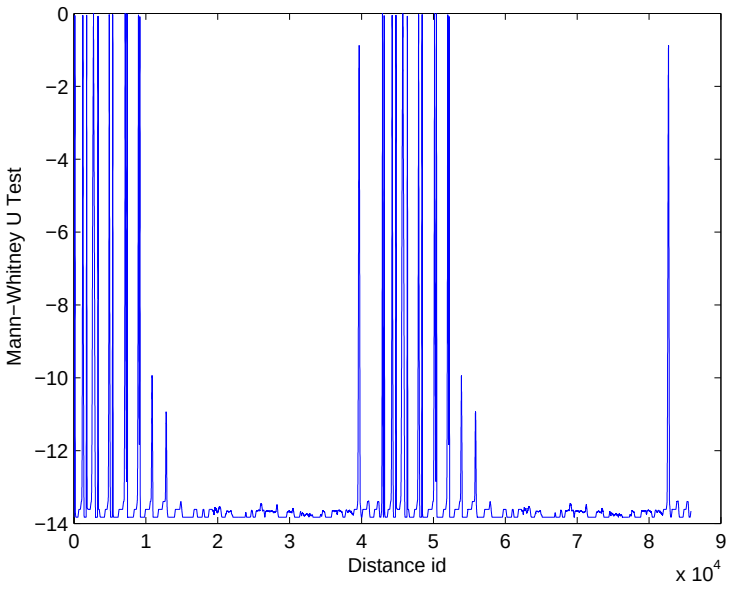


Figure 3.3: Histogram of change for the adjacent windows model using Mann-Whitney U Test

testing, data distributions before and after change are previously given. For example,in some parametric hypothesis testings such as t-test, paired t-test, and analysis of variance (ANOVA), data distribution is normal distribution. As we mainly focus on developing nonparametric change detection methods, we briefly introduce some nonparametric hypothesis tests. Based on the type of change, dissimilarity measures can be classified into the following types:

- Dissimilarity measures for detecting arbitrary change such Kolmogorov-Smirnov test, Cramer-von-Mises test.
- Dissimilarity measures for detecting scale shift such as Mood test, Kotz test.
- Dissimilarity measures for detecting location shift such Mann-Whitney U test, Normal Scores Test (median)

Wilcoxin Mann-Whitney test is a powerful nonparametric test. It is used to test whether populations have identical distributions. The Wilcoxin Mann-Whitney test does not assume that the difference between the samples is normally distributed. Wilcoxon Mann-Whitney test is a two-sample test that detect the location shift (median). It is used in the place of two-sample t-test when the data does not confines to the normality. Kolmogorov-Smirnove Goodness-of-Fit test is a two-sided test statistic that measures the goodness-of-fit between the empirical distribution

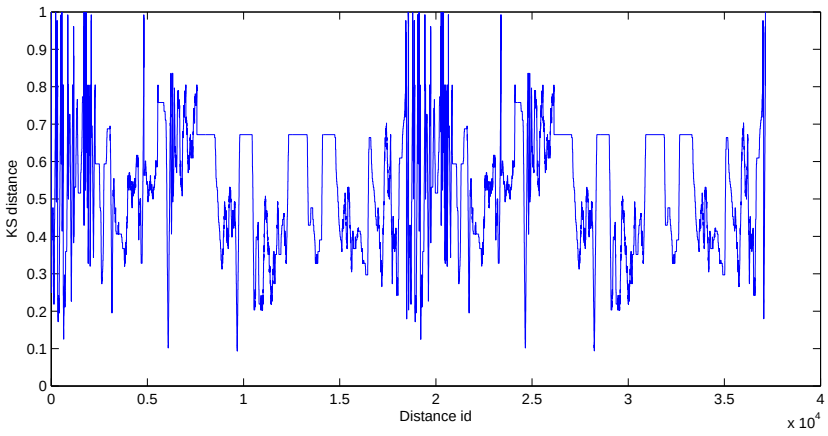


Figure 3.4: Histogram of change for the two adjacent windows model using Kolmogorov-Smirnov dissimilarity measure

and a given continuous probability distribution.Kolmogorov-Smirnov is a nonparametric test. The two-sample Kolmogorov-Smirnov test is sensitive in both location and shape of the data distributions of two samples. Therefore, the two-sample Kolmogorov-Smirnov test is one of the most powerful nonparametric methods for comparing two samples. Let $G(x)$ be a theoretical cumulative distribution function. Let $\hat{G}(x)$ denote the empirical cumulative distribution function of n observations. Two-sided statistic is defined by

$$D_n = \sup_x \mid G(x) - \hat{G}(x) \mid \tag{3.4.4}$$

or

$$D_n = \max_{1 \leq i \leq n} \left(G(x_i) - \frac{i-1}{n}, \frac{i}{n} - G(x_i) \right) \tag{3.4.5}$$

Figure 3.4shows the distribution of Kolmogorov-Smirnov dissimilarity measure between two adjacent windows.

Kullback-Leiber distance is used to measure the distance between two empirical distributions. The Kullback-Leiber distance between two probability mass function $p(x)$ and $q(x)$ is defined as

$$D(p,q) = \sum_{x \in \mathcal{X}} p(x) \log \left(\frac{p(x)}{q(x)} \right) \tag{3.4.6}$$

where $\mathcal{X}$ is the space of events.

3.4.3 Comparison of Geometric and Statistical Distances

As each distance function is capable of quantifying and detecting a type of change. We can only compare two detectors of detecting the same type of change. Detect-

Dissimilarity measure	Running time (ms)	Memory (MB)
Kolmogorov-Smirnov divergence	2953	4
Mann-Whitney U test	22077	4
Kullback-Leiber divergence	16270	4
Chebychev distance	685	2
Euclidean distance	654	3
Manhattan distance	567	3

Table 3.1: Time and memory required to compute dissimilarity measures

ing for means, detecting gradual change, or detecting abrupt change. The Euclidean distance can detect the large and abrupt changes while the Kolmogorov-Smirnov distance can detect small changes. Therefore they should not be compared with each other. The Manhattan and Euclidean distance are used for detecting the abrupt changes. In Section 3.6.3, we will evaluate and compare the Euclidean and the Manhattan distance-based detectors in terms of running time and memory consumption. Table 3.1 shows the running time and memory needed to compute the dissimilarity measures. In general, it takes more time to compute the statistical dissimilarity measures than the geometric measures. For example, the most time-consuming dissimilarity measure is Mann-Whitney U test (22077 ms). The least consuming-time dissimilarity measure is the Manhattan distance (567 ms). Computation of geometric dissimilarity measures consume less memory then that of statistical dissimilarity measures.

It would be ideal if there exists the incremental approaches to computing these distances, in particular for the geometric distances. To our best of knowledge, there is no incremental approach to computing the Euclidean distance. However, in chapter 4, we further reduces the time needed for computing the Euclidean distance based on the incremental computation of DFT in the window (Discrete Fourier Transform). As we can incrementally compute the DFT coefficients in the window , the Euclidean distance is computed on smaller data rather than the entire window. The details of DFT-based detector using the incremental strategy can be seen in Section 4.3. Specially, recently Cohen and Kaplan [Cohen 2012] have proposed the novel approaches to estimating the changes from the samples for the Manhattan and Euclidean distances instead of computing these distances on the original data.

3.5 Detection Threshold

A detection threshold is a value specified by a user or an automatic procedure in order to distinguish the state 'Changed' or 'Unchanged' of an event. As the balance between sensitivity and robustness of the change detection algorithm is partly

determined by detection threshold, selection of detection threshold is a critical step in order to develop powerful change detection algorithms. The goal of threshold selection is to choose the threshold in such a way that both the probability of false alarm and the probability of mis-detection are minimized.

There are two interrelated concepts of detection threshold in our work. The first kind of threshold called raw threshold is used to determine the number of truly changed points and the number of unchanged points. The second kind of threshold called characteristic threshold is used in our proposed algorithms for detecting changes in data streams. As synopsis-based change detection method usually uses some transformation to summarize raw streaming data into the streaming characteristics such as mean, variance, etc, there are synopsis-based detection threshold.

Assume that the value of raw threshold is ω, $\omega \in [\alpha, \beta]$, it is easy to see that the Euclidean distance-based threshold $\Omega_E \in \left[\sqrt{\mid \alpha - \omega \mid w}, \sqrt{\mid \beta - \omega \mid w}\right]$, the Manhattan distance-based threshold $\Omega_M \in [w \mid \alpha - \omega \mid, w \mid \beta - \omega \mid]$.

The choice of the threshold depends on the specific context of each application. Therefore, prior knowledge about the detection problem is needed for detection threshold to becomes meaningful. For example, change detection of battery-power level of sensors in sensor network, threshold is fixed and previously given. However, change detection in sensor network is used to detect the changes from the outside environment, threshold value must adapt to these changes of the environment. Chapter 5 will present an automated change detection method without requiring the given threshold of detection.

The choice of threshold is also another challenge of the change detection algorithm. It is one of the most important factors that affects the accuracy of the algorithm. Hence, the choice of threshold is sensitive. Change detection algorithms should be capable of automatically tuning detection thresholds when the false alarm rate is high. As change detection is a special case of the classification, selection of detection threshold for detecting changes is closely related to the problem of threshold selection in the classification problem. Recently Hernandez et al. have presented the approaches to choosing detection threshold based on performance metrics [Hernández-Orallo 2012]. In summary, the choice of detection threshold for the change detection algorithms is an interesting and open issue.

3.6 Evaluation of Change Detection

The change detectors were implemented in Java. All the experiments were run on a PC with a 2.60GHz 2x Pentium (R) Dual-Core CPU and 4GB memory, Windows platform.

All the proposed change detectors were thoroughly studied with the sensor data obtained from 54 sensors deployed in the Intel Berkeley Research lab between

Width	TP	FA	Misses	CR	N
8	2111	1090	2708	62304	68213
16	1093	1126	262	55828	58309
32	544	656	30	54167	55397
64	268	354	26	53309	53957
128	132	194	18	51629	51973

Table 3.2: Table of confusion matrixes of the Euclidean distance-based change detector

February $28th$ and April 5th 2004. There are a total of 2.3 million readings in this data. Each record contains the following information: date and time at which data were recorded, and the measures sensed by sensors, which includes temperature, humidity,light, and the battery power expressed in volts, ranging from 2 to 3. For convenience, without losing the generality we only experimented our change detector with the temperature from this data set by projecting the data set on temperature attribute. Our change detectors were evaluated in many aspects. The first group of experiments tested the effect of basic window sizes on the accuracy of change detection. The second group of experiments analyzed the effect of threshold selection on the accuracy. Finally, the third group of experiments compared the performance of change detectors using different distance functions.

For ease of evaluation, in particular, for obtaining the ground truth change, all the experiments in chapter were performed with overlapping windows model with sliding step of 1. We then assessed the running time of change detectors with different window sizes, and different data sets.

3.6.1 Effectiveness of Window Width

The goal of this group of experiments is to evaluate the effect of the window width on the performance measures of a change detector including running time, memory consumption, and detection accuracy. The Euclidean distance-based change detector was examined. The raw threshold was set to $30°C$. The distance-based threshold was fixed to 150, which corresponds to the raw threshold $30°C$.

As the change detection methods are developed for tuple-based data stream model, the window width is the number of tuples. The window width can be arbitrary value of integer. However, for ease of observation of the window width effect on the change detector, the window width was selected in the set $8, 16, 32, 64, 128$.

An increase in the window width results in the reduction in the number of observations and the number of actual change points (Table 3.2). Because each time a change is detected the current window is reset and the observations in the current window cannot be observed. However, if only based on these confusion matrixes,

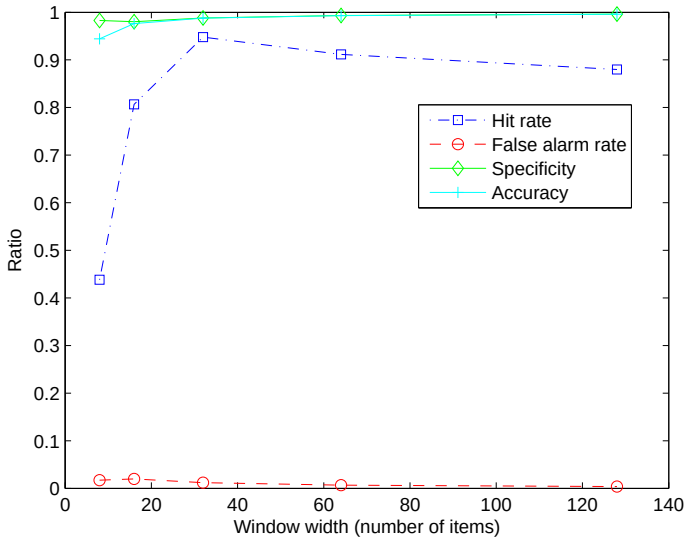


Figure 3.5: Effect of window width on detection performance of change detector in terms of detection theory metrics

it is impossible to conclude which window is better. The increase in the window width led to the decrease in the false alarm. We note that the hit rate in detection theory is Recall in information retrieval theory (also called the power of the change detector). The hit rate reached the maximum value with the window width 32. The detection accuracy and specificity (greater than 95%) were independent with the window width. The false alarm rate almost reduced with the increasing the width of the window. The precision of the change detector decreases with the increasing width of the window (Precision). For example, if the window width was 8, the precision of the corresponding detector was 65.95% while the precision of the corresponding detector was 40.49% with the window width was 128. The increase in the width of window resulted in the increase in the running time of the change detector. In other words, the detection delay increases with the increasing window width. As such, there is a tradeoff between the false alarm rate and the detection delay in the selection of the window width.

Figures 3.5 and 3.6 show the effect of window width on the detection performance of change detectors in terms of the performance metrics of detection theory, and information retrieval theory. Figure 3.7 shows the effect of window width on performance of change detector in terms of running time and memory consumption.

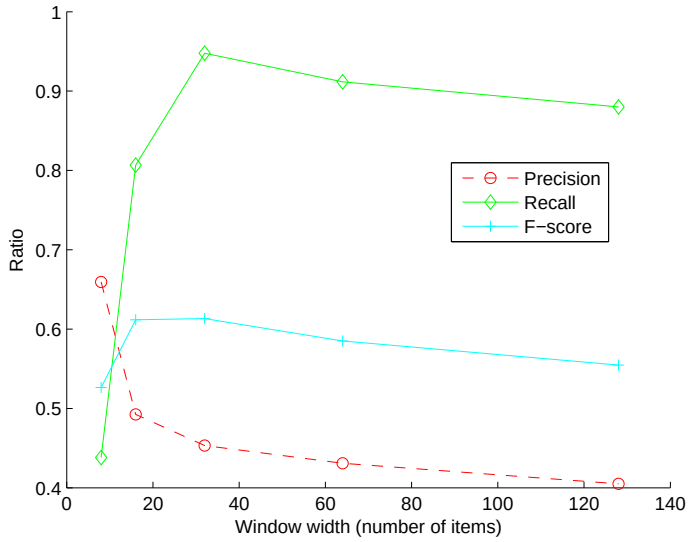


Figure 3.6: Effect of window width on performance of change detector in terms of information retrieval theory metrics

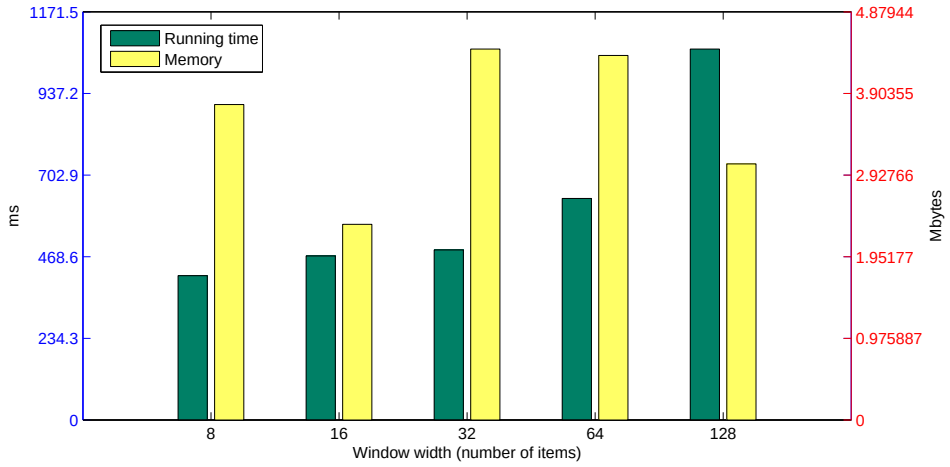


Figure 3.7: Effect of window width on detection performance of change detector in terms of running time and memory consumption

3.6.2 Effectiveness of Detection Threshold

To assess the effect of threshold selection on the performance of the change detector, the observed relationships (as shown in Figures 3.8, 3.9) between threshold and the performance metrics of change detector including performance metrics from the theory of information theory and the detection theory.

The Euclidean distance-based change detector was selected in this group of experiments. Without losing the generality, we selected the sensor data from sensor 1 in the Intel Berkeley Lab sensor data. The goal here was to detect the abnormal changes of the temperature in the sensor data.

To compute the performance metrics of a change detector empirically, we examined two kinds of change detectors. The first kind of change detector which we call absolute change detector using the absolute difference of two consecutive data items. We assume that the change points and unchange points detected by the absolute change detector are the actual change points and the actual unchange points. Therefore, there are two concepts of threshold: raw threshold and distance-based threshold.

The differences of sequence of a pair of consecutive data points were computed. Figure 3.10 shows the histogram of the absolute differences of the sequences of two consecutive points. The threshold determined by the absolute difference of two consecutive data points is called raw threshold.

To compute the performance metrics of the change detector using the Euclidean distance, the parameters were set as follows. The window size was set to 128. The raw threshold that is the difference between two consecutive points was fixed to $30°C$. The selection of the Euclidean distance-based threshold is based on the histogram of changes represented by the Euclidean distances as shown in Figure 3.11. The Euclidean distance-based threshold was varied in the range $50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900$. For each distance-based threshold, a confusion matrix was generated. From these confusion matrixes, the performance metrics of the detectors corresponding to the distance-based thresholds were computed. Figure 3.8 shows the relationships of distance-based threshold and the detection theory-based metrics including the hit rate, the false alarm rate, the accuracy rate, and the specificity rate. The results of this group of experiments show that, the smaller the distance-based threshold was, the higher the hit rate is. If the threshold is too small, the false alarm rate is high. In other words, the false alarms rate is inversely proportional to the threshold. However, if threshold is too big, the number the probability of hits is small, or in other words, the probability of missed changes is large. In summary, there is a tradeoff between the false alarm rate and the rate of missed changes. In particular, the false alarm rate increases when the detection threshold decreases. The missed change rate increases when the detection threshold increases. For example, with

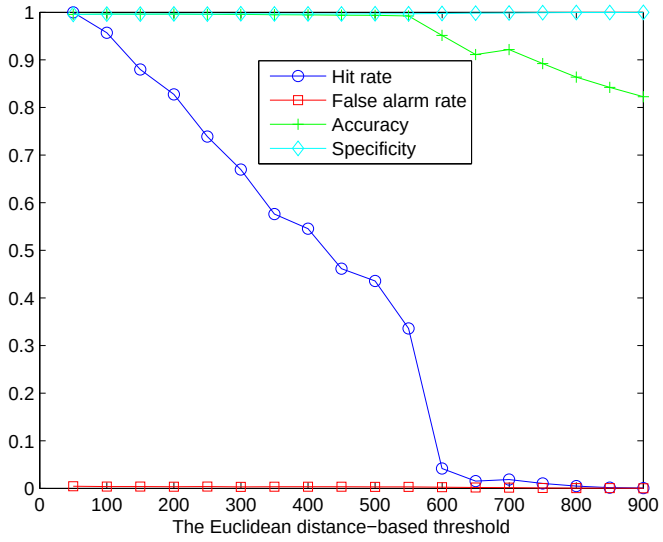


Figure 3.8: Effect of threshold on performance of change detector

threshold 100 the number of missed changes was 6 while the number of missed changes was 18 with the threshold 150. As shown in Section 2.4, the specificity reflects the ability to correctly reject the actual unchanged points, the detection accuracy represents the ability of the change detector to correctly detect the actual change points and rejecting the actual unchange points. The specificity of the Euclidean distance-based detector was high with specificity greater than 90%. The Euclidean distance-based detectors corresponding above thresholds were reliable with the accuracy rate greater than 85%. It is difficult to choose an optimal threshold if we are only based on individual performance metrics. Therefore, to select an appropriate threshold, we must consider multiple performance metrics at the same time instead of considering individual performance metrics.

Figure 3.9 depicts the relationships of distance-based threshold and the information retrieval theory-based metrics including Precision, Recall, and F-score. Recall (also called power of a change detector) represents the ability to correctly detect the actual change points. The higher Recall is, the more powerful the detector is. Contrary to the intuitive belief, larger precision and recall do not really mean better results for change detection algorithms [Liu 2010].

Selection of detection threshold depends on each specific application. For example, in a wildfire warning system, the probability of misses should be limited as small as possible, but in stock market related applications, both probability of false alarms and probability of misses should be limited.

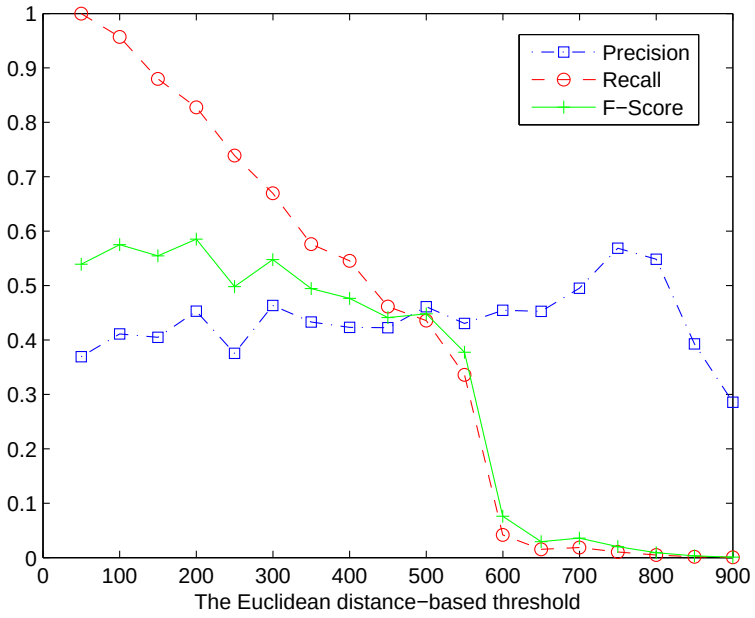


Figure 3.9: Effect of Threshold on Performance of Change Detector

3.6.3 Effectiveness of Dissimilarity Metrics

The goal of this group of experiments is to assess the effect of the dissimilarity metrics on the performance of a change detector. Given more than two change detectors, we want to select the best change detector. As such, we need to compare the change detectors in terms of detection accuracy, running time, or memory consumption.

We compared the detection performance of the Manhattan detector and the Euclidean detector. In this section, we compare two change detectors using overlapping windows model: the Euclidean distance-based change detector and the Manhattan distance-based change detector.

ROC curve is a method for selecting classifiers based on their performances [Fawcett 2004]. As a change detector can be a special binary classifier that classified the state of an observed event into one of two classes: change or unchange. Therefore, ROC graph can be used to compare the change detectors in terms of their performances.

We set the parameters for this group of experiments as follows. The window width was fixed to 128, and the absolute threshold was set to 30. To determine the ranges of the thresholds, we visualize the absolute threshold in Figure 3.10, the Euclidean distance in Figure 3.11, and the Manhattan distance in 3.12.

To compare the Euclidean distance-based detector and the Manhattan distance-

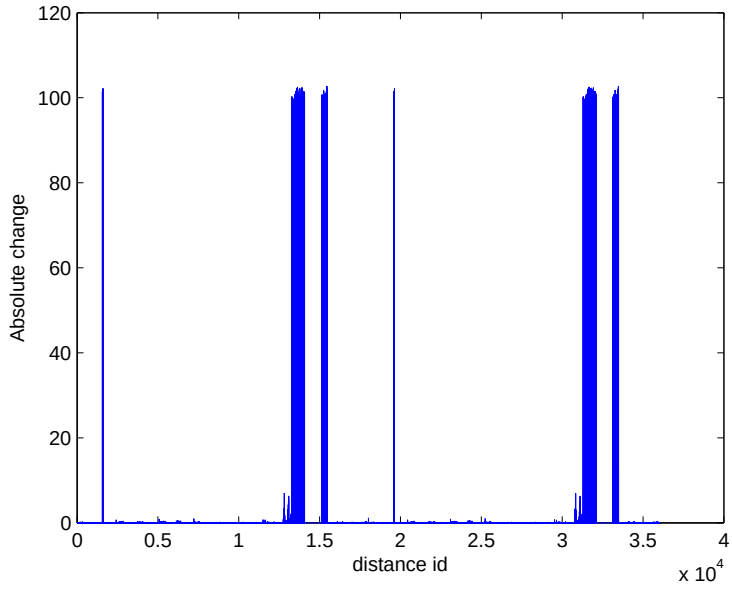


Figure 3.10: Histogram of the absolute temperature differences of two consecutive data points

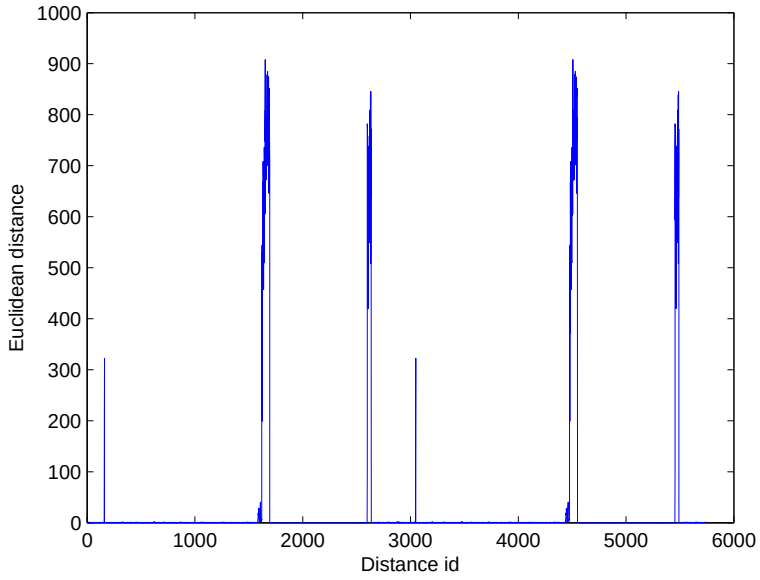


Figure 3.11: Histogram of the Euclidean distances between the reference window and the current window

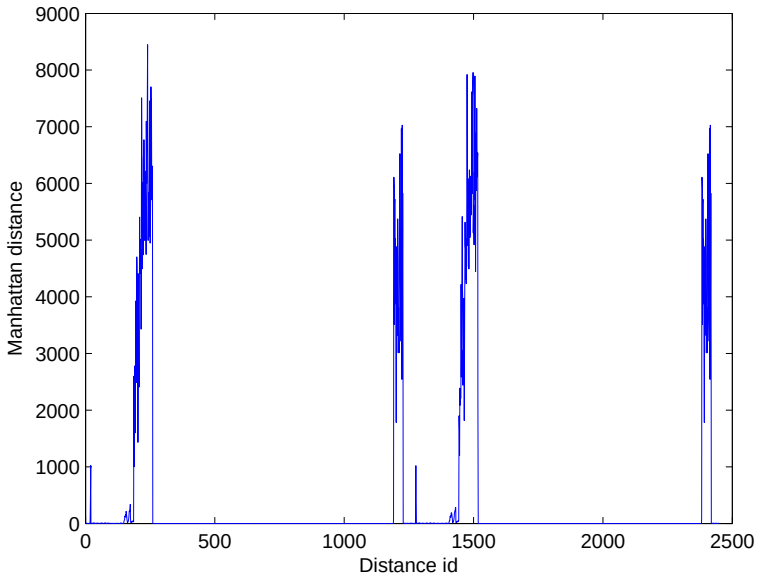


Figure 3.12: Histogram of the Manhattan distances between the reference window and the current window

based detector, two ROC curves of these two detectors were depicted in the same ROC space with the same window of 128 and the absolute threshold of 30. For the Euclidean distance-based detector, the distance-based threshold was varied in the range 50,100,150,200,250,300,350,400,450,500,550,600,650,700, 750,800,850,900. Each change detector with a distance-based threshold generated a pair of hit rate and false alarm rate (hit,fa) corresponding to a single point in ROC space. For the Manhattan distance-based detector, the distance-based threshold was changed in the range 500,1000,1500,2000,2500,3000,3500,4000,4500,5000,5500,6000,6500,7000, 7500,8000,8500,9000. The computation process of ROC curve of the Manhattan detector was performed the same as that of the Euclidean detector.

The better the performance of a change detector is (higher hit rate, lower false alarm rate), the more the ROC point corresponding to the change detector towards to the northwest. Figure 3.13 shows that, the change detector using the Euclidean distance is better than the change detector using Manhattan distance in ROC space. The ROC curves demonstrate that the Euclidean detector outperforms the Manhattan detector.

These results show that, it should be better to compare the detection accuracy of change detection methods in many metric spaces as each space of performance metric will provide us a different view of these change detection methods.

Figure 3.13 depicts the ROC curves of two change detectors: the Euclidean

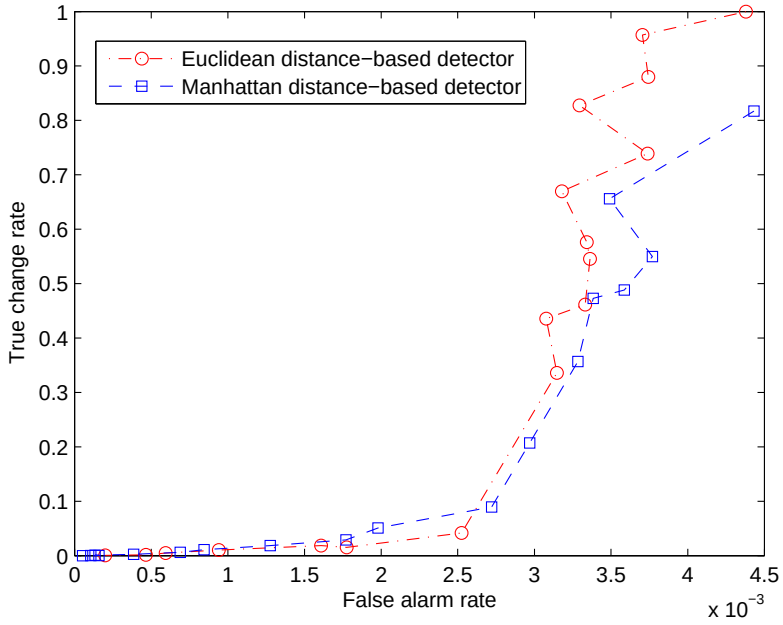


Figure 3.13: Comparison in ROC space of Change Detector

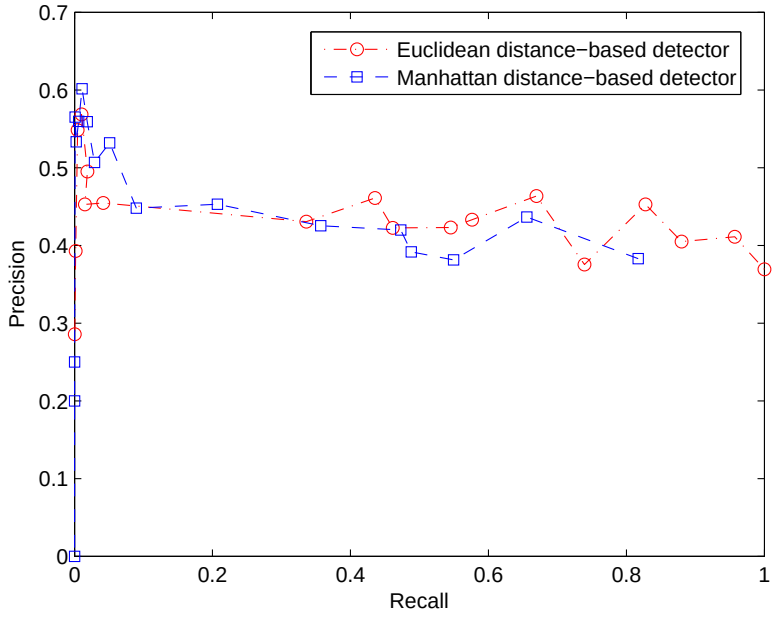


Figure 3.14: Comparison in PR space of Change Detector

distance-based detector and the Manhattan distance-based detector. Figure 3.14 shows the performances of the Euclidean distance-based change detector were the same as the those of the Manhattan distance-based change detector. In summary, in ROC space, the performance of the Euclidean distance based change detector was better than the Manhattan distance-based change detector while in PR space, two change detectors were almost the same. This difference is due to the unbalance of the number of actual change points and the number of the actual unchange points. For example, Table 3.2 showed the unbalance of the number of change points and the number of the unchange points. Therefore, a large change in the number of false alarms can only result in a small change in the false alarm rate. In PR space, Precision let us know the comparison of the number of false alarms and the number of actual change points. In other words, Precision and Recall are independent of the number of missed change points.

The results of these experiments show that the evaluation of a change detector should based on many metrics. Furthermore, comparison of two change detectors should be performed in both spaces ROC space and PR space, and two detectors should detect the same type of change.

3.7 Summary

In this chapter, we have introduced a collection of algorithms for detecting changes in a single data stream. The sliding window model is used to detect local changes in data stream. We have introduced a collection of algorithms for detecting changes in streaming data by using different sliding windows model.

By changing the relative positions of the reference and sliding windows, we can develop change detection algorithms for different purposes such as the problem of change point detection using overlapping windows model, or interval-based change detection using adjacent windows model.

In our experimental evaluation we have showed the effects of the factors such as the window width, the distance measure, and the detection threshold on detection the performance of change detectors.

We empirically analyzed the dissimilarity measure including geometric distances and statistical tests in terms of performance (memory and time required) and the detection accuracy. We compared the change detectors using different dissimilarity measures by using ROC and Precision-Recall curves. To our best knowledge, we have first evaluated the detection accuracy of a change detector at any time of a data stream or on the entire data stream.

CHAPTER 4

Synopsis-based Detection of Changes in A Single Data Stream

It is not the strongest of the species that survive, nor the most intelligent, but the one most responsive to change.
(Author unknown, commonly misattributed to Charles Darwin)

Contents

4.1 Introduction

As data streams arrive with high speed, detection of changes in raw streaming data in real-time is challenging [Gaber 2006]. Besides, if the window is so large that it may be impossible to store the window in memory. We need to summarize this window by constructing a compact synopsis. In this chapter, we propose a general framework for detecting changes in streaming data by using synopses that are constructed from two windows: the reference window and the current window.

In chapter 3, we introduced a general framework for change detection in a single data stream using the two-window approach. In fact, this framework focuses

on change detection in raw streaming data. In many applications, we are more interested in the features of streaming data than raw streaming data. For example, in medicare monitoring systems, physiology data is often periodic. Therefore, it is useful to analyse the physiology data in frequency domain. This chapter presents a general framework for detecting changes in synopses extracted from streaming data. It would be ideal if a synopsis-based change detector can simultaneously achieve the following goals: space-efficiency, computation efficiency, and detection accuracy. However these goals require the tradeoff

The contributions of this chapter are as follows

- We propose a general framework for detecting changes in data stream using synopsis.
- We analyze, design, and evaluate a specific algorithm for detecting changes in data stream by using Discrete Fourier Transform coefficients as synopsis structures. DFT-based change detection can be found in cognitive radio network [Kim 2008].
- We propose an incremental DFT-based change detector. In comparison with the direct DFT-based detector, this incremental DFT-based detector improves the detection performance in terms of running time and memory consumption.

The rest of this chapter is organized as follows. Section 4.2 introduces the general framework for synopsis-based change detection, and proves that the detection accuracy of synopsis-based detection method is preserved if the distance between two synopses is preserved under synopsis construction. Section 4.3 presents DFT-based change detector, a specific instance of synopsis-based change detector, by using Discrete Fourier Transform. We empirically evaluate the DFT-based detectors in Section 4.4 in terms of detection accuracy, running time, and memory consumption. We summarize the results of this chapter in Section 4.5.

4.2 Synopsis based Change Detection

Data stream summarization is one of the fundamental problems in data stream processing. Since data stream is infinite in nature, a common approach to summarizing a data stream is to observe and process it over a fixed size window. However, if the window is so large that it may be impossible to store the window in memory. The window need to be summarized by using compact synopsis construction. As the goal of change detection is to detect the abnormal or interesting event or behavior, the approximate answers to the queries relevant to the problem of change detection

is acceptable. Synopsis construction from streaming data has been long paid attention by research community of data stream processing due to the unique space and time constraints on the computation process of data streams. Algorithms for mining data streams can be based on constructing an accurate or approximate synopses of the real-time data streams.

Synopsis or sketch is a small data structure maintaining an approximate data structure rather than an exact representation [Babcock 2002]. Sketch is a compressed representation of a data set. If x is a data set then a sketch of this data set is given by a function $f(x)$ [Nelson 2012]. There are some types of synopses such as random samples, histograms, wavelets, and sketches. The issues related to synopses include time efficiency, space, practicality, and incremental maintenance. As streaming data changes overtime, synopses that summarize streaming data need to be updated in order to reflect the changes of streaming data. Synopses can be used in conjunction with a variety of techniques for processing and mining data streams. Synopsis can be used to estimate a global aggregate over a sliding window. Continuous queries using synopses reduce considerable amount of memory yet may provide almost the same result as query processing on the raw data [Klan 2011]. In particular, the technique of synopsis construction can be incorporated with change detection efficiently because it only needs some synopsis structure which informs the temporal behavior of the stream without considering the individual data points [Aggarwal 2007].

Many algorithms for detecting changes have been recently proposed for the corresponding synopses. Sampling can be used as a utility to construct a synopsis structure from a data stream [Li 2007]. Sebastiao et al. [Sebastião 2007]. propose a change detection method that can identify sudden changes in the data stream by using cluster histogram. Let H_1 and H_2 denote two cluster histograms that are constructed from two corresponding windows: the reference window w_1 and the current window w_2. In order to determine whether a change occurs, the following hypothesis test $Dist(H_1, H_2)$ is greater than the given threshold t. Chen et al. have used histograms to summarize data in window for change detection [CHEN 2009]. Krishnamurthy et al. [Krishnamurthy 2003] propose an approach to building compact summaries of the data using sketches. After that, detecting significant changes is performed on time many series forecast models built on sketches by looking for flows with large forecast errors. The advantage of this approach is that it is capable of detecting significant changes in massive data streams with a large number of time series. Casolari et al. have recently presented a method for detecting changes in streaming data that is created from Internet-based systems [Casolari 2012]. Their change detector uses wavelet transform to eliminate perturbations. Another approach to synopsis-based change detection is to use the data mining results as synopses. Instead of detecting changes in raw streaming data, the change detection schemes identify the changes in the most quintessential informa-

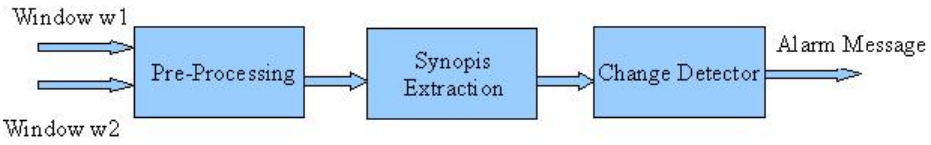


Figure 4.1: Block diagram for detecting changes in data stream using synopsis structures

tion structures constructed from massive data streams by streaming data mining methods such clustering, frequent pattern mining, and so on. The first work on model-based change detection proposed in [Ganti 1999, Ganti 2002] is FOCUS. The framework FOCUS consists of two components: structural and measurement component. To detect deviation between two models, they compare specific parts of these corresponding models. The models can be obtained by data mining algorithms such as frequent item sets, decision trees, and clusters. Valizadegan and Tan present a clustering based change detection method that summarizes streaming data by clustering [Valizadegan 2007]. Gaber and Yu [Gaber 2006] present a clustering-based change detection method that clusters data stream online, and determine the clustering features. Change is detected by using these clustering features. For example, Chen et al. [Chen] describes another clustering based approach. This approach stores the cluster entropy characteristics of data stream in a summary structure, called HE tree (Hierarchical Entropy), and detects the change of Best K. This method is memory efficient. Micro-clustering is used as a method for constructing synopses from streaming data.

There are many model-based approaches to detecting changes such as one-model approach, two-model approach, or multi-model approach [Gustafsson 2001]. Severo et al [Severo 2006] propose a change detection method using one-model approach in which the change is detected by using Kalman filtering and CUMSUM. An example of change detection using multi-model approach is proposed by Curry et al. [Curry 2007]. This dissertation focuses on the two-model approach. In particular, this chapter presents a general two-window change detection method. Chapter 4 and 5 present the synopsis-based change detectors using Discrete Fourier Transform and clustering. As Figure 4.1 shows, a synopsis-based method for detecting a single data stream consists of the following modules

- *Preprocessing:* Preprocessing is an important step in adaptive systems. As data comes from real world can be noisy and redundant, the tasks of preprocessing can be removing noise and redundancy, and data normalization. A commonly used approach to preprocessing is to use filters to remove noise. A recent work on adaptive preprocessing for streaming data can be seen in [Zliobaite and 2012].

- *Synopsis construction.* Given two basic windows of size b: w_1, w_2, two synopsis structures are constructed by techniques of synopsis constructions such as sampling, wavelets, sketches, histograms, and clustering etc.

- *Change detection:* The next step is to quantify the difference between two synopses that are constructed from two windows. The dissimilarity metrics used to quantify the diffeence between two synopses can be any dissimilarity metric that is suitable for each application. Let W_1 and W_2 be two corresponding synopses constructed from two windows w_1, w_2 respectively, the change is detected by testing the following hypotheses

$$\begin{cases} \widetilde{\mathcal{H}}_0 & d(W_1, W_2) \leq \Omega \\ \widetilde{\mathcal{H}}_1 & d(W_1, W_2) > \Omega \end{cases} \tag{4.2.1}$$

  where Ω is the distance-based threshold.

Therefore, instead of computing the dissimilarity measure between two windows of size N, we need only to compute the dissimilarity measure between two synopses of size K, where $K \ll N$.

More importantly, we have proposed a general change detection framework in which synopses can be incrementally computed. Algorithm 4 describes the synopsis-based change detector using two adjacent windows model. Method $synopsisConstruction()$ constructs a synopsis from the corresponding window. A distinction between a non-synopsis change detection and synopsis-based change detection is that the dissimilarity measure in the former is directly computed on the samples of two windows w_1 and w_2 while the dissimilarity measure in the later is computed on the synopses. Therefore, computation of the dissimilarity measure between two synopses can be considered as the heart of the synopsis-based change detection approach.

The probability of *false alarm* and the probability of *detection* are computed by $P_{FA} = Pr(\widetilde{\mathcal{H}}_1|\widetilde{H}_0)$ and $P_D = Pr(\widetilde{\mathcal{H}}_1|\widetilde{\mathcal{H}}_1)$ respectively. By using synopsis structures for change detection, the dimension of the problem can significantly reduces.

Lemma 4.2.1. *Let W_1 and W_2 be two synopses constructed from two corresponding windows w_1 and w_2. If the distance between two windows w_1 and w_2 $d(w_1, w_2)$ is equal to the distance between two synopses $d(W_1, W_2)$, then the detection performance of the synopsis-based change detector is similar to that of the change non-synopsis change detector. In other words, the detection performance of the synopsis-based detector is preserved if the synopsis-based distance is equal to the non-synopsis distance.*

Proof. Let two windows w_1 and w_2 be the reference window and the current window running on the data stream S. Let $W_1 = f(w_1)$ and $W_2 = f(w_2)$ be two

Algorithm 4: Change detection algorithm using synopsis structure (Two Adjacent Windows)

input : A data stream S, a distance based threshold D^*
output: The messages reporting the changes occurred, and time at t which changes occurred

Step1: **begin**
- $t \longleftarrow 0$;
- $w1 \longleftarrow$ first b points from time t;
- $W1 \longleftarrow synopsisConstruction(w1)$;
- $w2 \longleftarrow$ next b points in the data stream;
- $W2 \longleftarrow synopsisConstruction(w2)$;

Step2: **while** *not the end of the stream* **do**
- **if** $d(W1, W2) > D^*$ **then**
  - $t \longleftarrow$ current time;
  - Report change occurred at time t;
  - Clear all windows and GOTO *Step 1*;
- **else**
  - $w2 \longleftarrow sliding(w2, 1)$;
  - $W2 \longleftarrow synopsisConstruction(w2)$;

synopses constructed from two corresponding windows by a sketching function f The detection performance of non-synopsis change detector is determined by

$$H_{raw} = Pr(\mid d(w_1, w_2)\mid > \omega \mid actualchange = true) \tag{4.2.2}$$

and

$$FA_{raw} = Pr(\mid d(w_1, w_2) \mid > \omega \mid actualchange = false) \tag{4.2.3}$$

The detection performance of synopsis-based change detector is determined by

$$H_{syn} = Pr(\mid d(W_1, W_2)\mid > \omega \mid actualchange = true) \tag{4.2.4}$$

and

$$FA_{syn} = Pr(\mid d(W_1, W_2) \mid > \omega \mid actualchange = false) \tag{4.2.5}$$

As the distance is preserved under synopsis construction process $d(w_1, w_2) = d(W_1, W_2)$, the detection performance of synopsis-based change detector is similar to that of the non-synopsis change detector: $H_{raw} = H_{syn}$ and $FA_{raw} = FA_{syn}$. In summary, the performance of synopsis-based change detector is similar to the non-synopsis change detector if there exists a sketching function f that preserves the distance measure between two synopses.

□

Some transforms in discrete signal processing can preserve the Euclidean distance in the transformed domain such Discrete Fourier Transform, Haar Wavelet Transform [Chan 1999]. Section 4.3 presents the synopsis-based change detector with DFT coefficients (Discrete Fourier Transform) as synopses using the Euclidean distance. As the Euclidean distance is preserved under the Discrete Fourier Transform, the DFT-based change detector using the Euclidean distance meets Lemma 4.2.1.

4.3 DFT-based Change Detection

Discrete Fourier Transform is a well-known signal transformation transforming signal from a given domain to another. It is used in many domain applications such as digital signal processing (DSP), image processing, and so on.

In the fields of database and data mining, DFT is used as a method for extracting the features from the data sets [Mörchen 2003]. DFT transform is used to compress the decision trees for distributed data mining [Kargupta 2001]. It is also used for the problem of similarity search in sequence databases. Besides, as DFT transform

is specially efficient for periodic signals, it is useful for processing time series data as well as time series streaming data such as ECG data, acoustic data . Beringer and Hullermeier present an online method for clustering of parallel data streams using DFT transform. Zhu and Shasha use DFT transform to find the correlations among thousands of pairs of data streams [Zhu 2002].

As the Euclidean distance is preserved under the DFT transform according to Parseval's theorem [Oppenheim 1989], many data mining techniques using DFT and the Euclidean distance can preserve privacy in the distributed environment. As such, DFT-based change detector using the Euclidean distance can be seen as a privacy-preserving mining technique. The DFT is an orthogonal transformation that is used in many practical applications due to the existence of the fast computation algorithm FFT.

Definition 4.3.1. *The time series data stream $x(t)$ can be represented as a linear combination of basis functions.*

$$x(t) = \frac{1}{\sqrt{N}} \sum_{\omega=0}^{N-1} X(\omega) e^{\frac{2\pi i \omega t}{N}} \tag{4.3.1}$$

where

$$X(\omega) = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} x(t) e^{-\frac{2\pi i \omega t}{N}} \tag{4.3.2}$$

Discrete Fourier Transform has the following properties:

- *Preservation of the Euclidean distance:* The Euclidean distance is preserved under the Discrete Fourier Transform [Oppenheim 1989]. As shown in Lemma 4.2.1, the detection accuracy of DFT-based detector using the Euclidean distance is similar to the Euclidean distance-based detector. Therefore, instead of computing the Euclidean distance on two windows of N samples, the Euclidean distance between two windows w_1 and w_2 can be computed from the DFT coefficients extracted from two corresponding windows.

- *Symmetry:* Most time series data streams are real sequence, then $X(i) = X^*(N-i)$, for $i = 1, ..., N-1$ where X^* is the conjugate of the complex number X. As such, instead of computing N DFT coefficients, DFT algorithm only requires to compute $N/2+1$ DFT coefficients. For example, if the width of sliding window size is N = 1024, then the number of DFT coefficients is only 513.

Direct computation of DFT coefficients needs $O(N^2)$ units of time, but with Fast Fourier Transform, the time required reduces to $O(NlogN)$ based on the above

properties. Furthermore, since the energy of the time series data stream only concentrates on the first few DFT coefficients, it is sufficient to capture K most important coefficients where $K << N$. Recently, some groups of researchers have proposed faster Fourier transform algorithms for sparse data streams. For instance, [Zou 2006] explains that, it is possible to use a few DFT coefficients (K = 40), extracted from a very large window (N = 4million), in the compressive sensing process. In particular, Hassanieh et al. [Hassanieh 2012b, Hassanieh 2012a] have presented a fast Fourier transform algorithm for sparse data stream, which is one of the 10 breakthrough technologies in 2012. As most real world data is sparse, the sparse FFT is useful. It is shown that, data stream can be processed 10 to 100 times faster than it was processed with the traditional FFT. Integration of high-speed data stream and signal-processing operations into a single system is a mandatory for many applications such as preventive maintenance of industrial equipment; detection of water pipeline leakage; anomaly detection in electrocardiogram signals in medical applications [Girod 2007, Girod 2008].

Girod et al have proposed a signal-oriented data stream management systems combining both signal processing and data stream operators together. In this spirit, we propose a change detection method for streaming data using Discrete Fourier Transform. DFT-based change detection can be used as a component for the cognitive radio system [Lai 2008]. DFT-based change detection is useful for periodic change detection

4.3.1 Incremental Computation of DFT coefficients

Incremental computation is a fundamental problem in data stream processing and mining. Due to the unlimited nature of streaming data, the incremental estimation of synopsis for change detection is important. There has been recently some work focusing on this direction [Tschumitschew 2010]. To further improve the speed of change detector, Lemma 4.3.1 is exploited to reduce the time required for computing the DFT coefficients. Specifically, instead of computing the DFT coefficients of the basic window w_2, we use an incremental strategy to compute the DFT coefficients of the basic window w_2 because fast incremental processing of new incoming data items arises from the nature of on-line processing of data streams.

Lemma 4.3.1. *[Zhu 2002]Let X_k^{old} be the k-th DFT coefficient of the series in the sliding window $x_0, x_1, .., x_{N-1}$ and X_k^{new} be that coefficient of the series $x_1, x_2, .., x_N$,*

$$X_k^{new} = e^{\frac{j2\pi k}{N}}(X_k^{old} + \frac{x_N - x_0}{\sqrt{N}}) \tag{4.3.3}$$

$$k = 1, .., K$$

where K is the number of DFT coefficients that are used as synopses and $K \leq N$.

The worst case is $K = N$. However,in the real world applications, this number K is often much smaller than N because only few coefficients are important. Therefore, the larger window is really beneficial for computing DFT coefficients. It can seen that except the initial step the time needed for computing the Euclidean distance between

Proof. We have

$$X_m^{old} = \frac{1}{\sqrt{M}} \sum_{i=0}^{w-1} x_i W^{mi} \tag{4.3.4}$$

where $W = e^{-\frac{2j\pi}{M}}$

We compute X_m^{new},

$$X_m^{new} = \frac{1}{\sqrt{M}} \sum_{i=0}^{w-1} x_{i+1} W^{mi} \tag{4.3.5}$$

Let $k = i + 1$, we have $X_m^{new} = \frac{1}{\sqrt{M}} \sum_{k=1}^{M} x_k W^{m(k-1)}$

As such,

$$X_m^{new} = W^{-m}(\frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} x_k W^{mk)} + \frac{1}{\sqrt{M}} W^{mM} x_m - \frac{1}{\sqrt{M}} W^{m0} x_0) \tag{4.3.6}$$

So,

$X_m^{new} = e^{\frac{j2\pi m}{w}}(X_m^{old} + \frac{x_M - x_0}{\sqrt{M}})$

□

Hassanieh et al. shows that the time needed for computing K non-zero DFT coefficients by their randomized algorithm DFT of N points is $O(K \log N)$. The computational complexity of K DFT coefficients using the fast Fourier transforms is $O(K \log N)$. It can be easily seen that from Lemma 4.3.1, the time needed for incrementally computing of K DFT coefficients is only $O(K)$.

As such, the incremental method for computing K DFT coefficients runs faster than the direct method for computing DFT coefficients $\log N$ times. Therefore, it can seen that if the larger the window width N is, the more efficient the incremental method for computing K DFT coefficients than the direct computation of DFT coefficients is.

Let T_{react} be the reacting time, which is the time needed for reacting to the change. The reacting time is computed from the time at which the change occurs to the time at which the detector makes an alarm. Reacting time includes the detection

delay and the time needed for an making alarm of the occurred change. As such, the reacting time mostly depends on the detection delay. In fact, detection delay is due to the time for computing the distance function that quantifying the change and the time for comparing this function with the detection threshold. However, the time for comparing can be ignored. In other words, detection delay depends on the time for computing the distance function. Let d be a distance function that quantifying the change. Let T_{dist} be the time needed for computing the distance function d of two windows. We note that in Algorithm 5, only the DFT coefficients in the current window is incrementally computed while the DFT coefficients in the reference window is directly computed. We consider three following cases:

1. For quantifying the difference between two windows of N points of streaming data, the time for quantifying the change is $T_{dist}(N)$.

2. For quantifying the difference between two synopses, and each synopsis consists of K of DFT coefficients, the time for quantifying the difference in a DFT-based change detector depends on the time for computing the DFT coefficients in each window. $T_{dist}(K)$. If the DFT coefficients are directly computed from both the reference window and the current window, the time for quantifying the difference $T_{dist}(K)$ can be approximated by

$$T_{diff}^{direct} \approx T_{dist}(K) + O(K \log N) + O(K \log N) \tag{4.3.7}$$

3. For the case of incremental computation of DFT coefficients in the current window, the time for quantifying the difference can be approximated by

$$T_{diff}^{incre} \approx T_{dist}(K) + O(K \log N) + O(K) \tag{4.3.8}$$

From the equations 4.3.7 and 4.3.8, it can be seen that the incremental DFT-based detector runs faster than the direct DFT-based detector. However, we cannot provide any conclusion of the detection performance in terms of running time of the detector on two windows of raw streaming data and the direct DFT-based detector as well as that of detector on two windows of raw streaming data and the incremental DFT-based detector.

4.3.2 Algorithm Description

Algorithm 5 describes the incremental DFT-based detector. The inputs to this method include a data stream S, and a given threshold D^*. One of the fundamental methods for synopses is to incrementally update synopsis with newly arriving item. Procedure $IncrementalDFT(W2, x, y)$, which implements Lemma 4.3.1, computes the DFT coefficients in the window w_2 incrementally on the basis of the

old DFT coefficients in the window w_1, the expired item x, and the newly arriving item y. Due to the unlimited nature of streaming data and the restricted resources such as memory, and computing power, data stream processing usually uses the incremental approach in order to overcome these challenges. In order to quantify the change based on synopses, we need to determine a dissimilarity function between two synopses. For example, with the Discrete Fourier Transformation, the dimension of the problem reduces from N to K.

Algorithm 5: Change detection algorithm using synopsis structure (Two Adjacent Windows)

input : A data stream S, a distance-based threshold D^*
output: The messages reporting the changes occurred, and time at t which changes occurred

Step1: **begin**
- $t \longleftarrow 0$;
- $w1 \longleftarrow$ first b points from time t;
- $W1 \longleftarrow DFT(w1)$;
- $w2 \longleftarrow$ next b points in the data stream;
- $W2 \longleftarrow DFT(w2)$;

Step2: **while** *not at the end of the stream* **do**
- **if** $d(W1, W2) > D^*$ **then**
  - $t \longleftarrow$ current time;
  - Report change occurred at time t;
  - Clear all windows and GOTO *Step 1*;
- **else**
  - $x \longleftarrow fistItem(w2, 1)$;
  - $w2 \longleftarrow sliding(w2, 1)$;
  - $y \longleftarrow lastItem(w2)$;
  - $W2 \longleftarrow IncrementalDFT(W2, x, y)$;

4.4 Evaluation

This section evaluates the DFT-based detectors in terms of detection accuracy, and performance metrics including running time, and memory consumption. The first group of experiments aim to demonstrate that the detection accuracy of the synopsis-based detector is preserved if the the distance quantifying the difference between two synopses is preserved. In other words, the accuracy of the synopsis-based detector is similar to that of the non-synopsis detector if the distance between

two synopses is equal to the distance between two original data samples. We implemented both the direct DFT-based detector and the incremental DFT-based detector using the Euclidean distance for quantifying the difference. The second group of experiments shows evaluates the performance of the DFT-based detectors in terms of running time, and memory consumption. This group of experiments also makes comparative evaluations of the DFT-based detectors with other detectors in terms of running time, and memory consumption.

4.4.1 Evaluation on Accuracy of Detection

Lemma 4.2.1 proves that detection performance of synopsis-based detector is the same as that of non-synopsis detector if the distance function is preserved under the synopsis transform. Such a distance-preserved transform is the Discrete Fourier Transform. This subsection empirically demonstrated Lemma 4.2.1 with DFT-based change detector. The distance between two DFT-based synopses was the Euclidean distance between two DFT-based synopses. The Euclidean distance-based detector was selected as a non-synopsis detector. The DFT-based detector was selected as synopsis-based detector. Two change detectors were run on the same data set generated by sensor 2 in the Intel Berkeley sensor data set. The fast DFT algorithms use conquer-and-divide strategy to improve the performance. As a computation of DFT of N points is recursively divided into computation of 2 smaller $N/2$ points, the number of data points N should be of 2^M. Without loss of generality, the window size was set to 128 in both the Euclidean distance-based detector and DFT-based detector in this group of experiments. This size of 2^M can be a drawback of DFT-based change detectors.

Similar to Section 3.6.2, it is assumed that the change points and unchange points detected by the absolute change detector are the actual change points and the actual unchange points. The raw threshold that is the difference between two consecutive points was fixed to $30°C$.

The selection of the Euclidean distance-based threshold was based on the histogram of changes represented by the Euclidean distances as shown in Figure 3.11. The Euclidean distance-based threshold was varied in the range 50,100,150,200,250,300,350,400,450,500,550,600,650,700, 750,800,850,900. For each distance-based threshold, a confusion matrix was generated. From these confusion matrixes, the performance metrics of the detectors corresponding to the distance-based thresholds were computed. As the Euclidean distance is preserved under the Discrete Fourier Transform, the distance-based threshold of the DFT-based detector was also changed in the range 50,100,150,200,250,300,350,400,450,500,550,600,650,700, 750,800,850,900. For each distance-based threshold, a confusion matrix was generated. From these confusion matrixes, the performance metrics of the detectors corresponding to the

	Detected changes	Detected unchange	Total
Actual change	127	0	127
Actual unchange	217	49325	49542
	344	49325	49699

Table 4.1: The Euclidean distance-based detector and DFT-based detector with distance-based threshold 50 and absolute threshold 30 has the same confusion matrix

distance-based thresholds were computed.

The results of the experiments on the Euclidean-based detector and DFT-based detector demonstrated that, the detection performance of the Euclidean distance-based detector was similar to that of the DFT-based detector.

Table 4.1 illustrates that, the Euclidean distance-based detector and DFT-based detector generated the same confusion matrix with the distance-based threshold of 50 and the absolute threshold 30. From this confusion matrix, the performance measures of both detectors computed from this confusion matrix were computed. In particular, both detectors have the probability of detection (hit rate) 1.0, the false alarm rate 0.00438, Precision 0.369, Recall 1.0, the detection accuracy 0.9956, specificity 0.9956, and F-score 0.5393. Similarly, the performance measures of two detectors were computed with the other value of distance-based threshold. Figure 4.2 showed that, two Precision-Recall curves of both detectors are the same. In summary, our theoretical and empirical results show that, given a synopsis transform that preserves the distance measure, the performance measures of non-synopsis change detector and synopsis-based change detector are similar in terms of detection accuracy metrics including the hit rate, the false alarm rate, Precision, Recall, as well as the corresponding derived metrics (F-score, accuracy, specificity).

4.4.2 Evaluation on Performance

In this group of experiments, we evaluate the performance of both direct and incremental DFT-based detectors in terms of running time on the same entire data stream sensor 2. We ran the all the experiments with the detectors using two adjacent windows.

Figure 4.4 shows that the running time of both the direct and incremental DFT based detectors on the same entire data stream. As DFT-based detectors uses the adjacent windows model to detect the changes in data stream. Therefore, every time a change is detected, the detector is reset. Therefore, the smaller window width is, the more often the detector resets. In other words, the smaller the window width is, the larger the time needed for directly computing DFT coefficients in the refer-

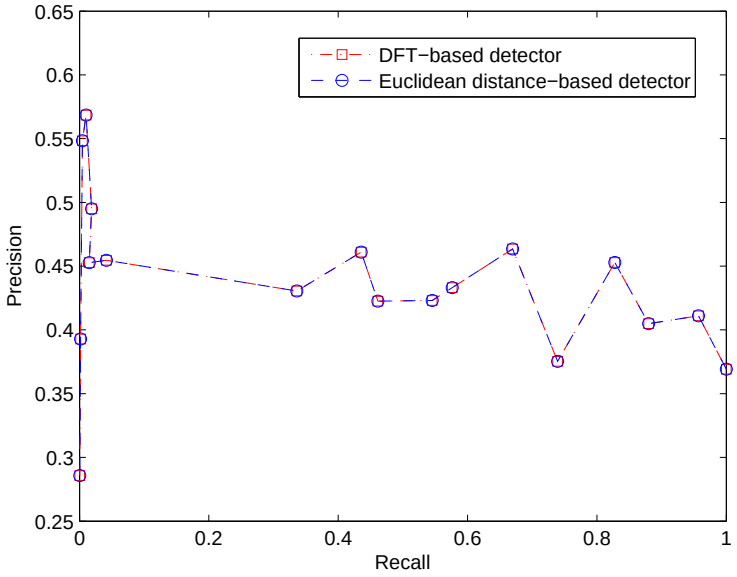


Figure 4.2: Comparison of the Euclidean distance-based detector and DFT-based detector in PR space

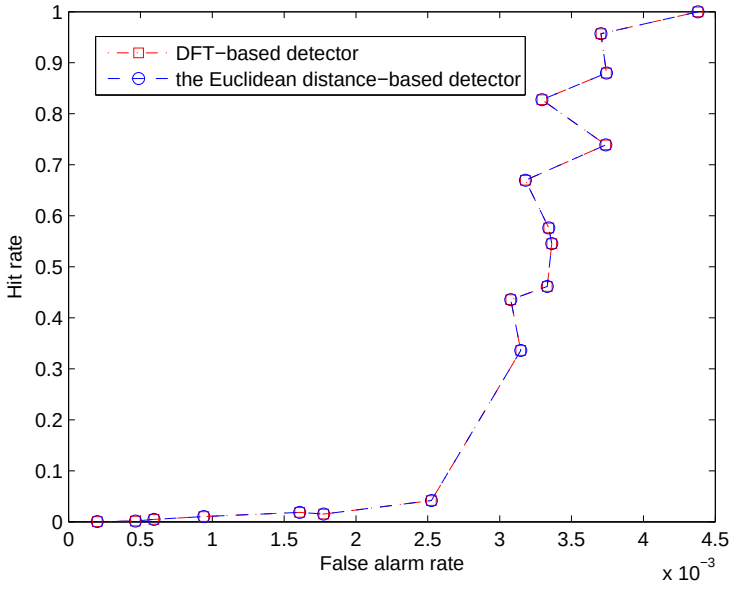


Figure 4.3: Comparison of the Euclidean distance-based detector and DFT-based detector in ROC space

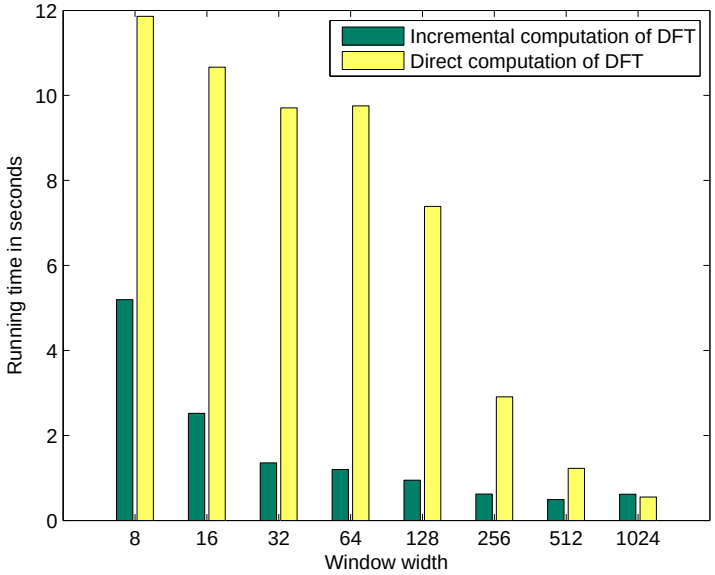


Figure 4.4: Running time of both direct and incremental DFT-based detectors on the entire data stream

ence window consumes. As Section 4.3.1 Figure 4.4, the incremental DFT-based detector ran faster than the direct DFT-based detector.

We also compared the performance of direct DFT-based detector and incremental DFT-based detector in terms of memory consumption and running time. The window width was set to 128. The raw threshold was 30. The Euclidean distance-based threshold was 150. The sliding step was 1, that means the reference window and the current window are overlapping by 127 positions. The experiment results shows that, the incremental DFT-based change detector (552 ms) runs faster than the direct DFT-based change detector (1863 ms). The incremental DFT-based change detector requires 2.46 MB memory while the direct DF-based change detector needs 2.57 MB memory for detecting changes on the entire data stream sensor 2. As such, there is no much difference in terms of memory required by both the incremental DFT-based change detector and the direct DFT-based change detector. As we expect in Lemma 4.3.1, the computing power of incremental computation of DFT in sliding window further improve than the direct computation of DFT.

We repeated the experiment with the Euclidean distance-based change detector on the same data stream sensor 2 with the same parameter setting. The running time of the Euclidean distance-based change detector was 1024 ms. The memory consumption was 2.725 MB.

Table 4.4.2 summarizes this group of experiments. As such, among three detectors, the incremental DFT-based change detector runs fastest, and requires least

Dissimilarity measure	Time (ms)	Memory (Mb)
Detector on raw data	1024 ms	2.725 MB
Direct DFT-based detector	1863 ms	2.57 MB
Incremental DFT-based detector	552 ms	2.46 MB

Table 4.2: Running time and memory consumption of the Euclidean distance-based detector, direct DFT-based detector, and incremental DFT-based detector on the entire data stream sensor 2

memory. As we can incrementally compute the DFT coefficients in the window and then we compute the Euclidean distance on smaller data rather than the entire window, we obtained this result as expected in Lemma 4.3.1. We also note that all the DFT-based change detectors that we implemented are based on the traditional fast fourier transform algorithm. In particular, the performance of DFT-based further improves if the it performs on the periodic data streams such as ECG data, acoustic data. The recent advance in fast computation of DFT [Zou 2006, Hassanieh 2012b, Hassanieh 2012a] can motivate us to develop the faster DFT-based detectors. The reasonable foundation is that most real world data is sparse, the sparse FFT is useful. Therefore, it is sufficient to consider only a few coefficients.

We also evaluated Kolmogorov-Smirnov distance-based change detector proposed by Kifer et al.[Kifer 2004] on raw streaming data set sensor 2. We set parameters for this experiment as follows. The window width was fixed to 128. The absolute threshold was 30. The Kolmogorov-Smirnov distance varies in the range [0,1]. The distance-based was 0.5. The sliding step was 1. The detection performance of Kolmogorov-Smirnov distance-based change detector in terms of memory consumption, and running time are as follows. The time required for detecting changes in the entire data stream sensor 2 is 7201ms. The memory consumption is 2.1MB. It can be seen that even compared with the direct DFT-based change detector, the running time of the KS distance-based detector (7201ms) is much larger than that of the direct DFT-based change detector (1863 ms). This result fits with our observation discussed earlier.

4.5 Summary

Due to the need for detecting the changes of features constructed from data streams as well as the resource constraints on data stream processing and mining, a general synopsis-based change detection framework is proposed. Both theoretical and empirical analysis demonstrates that, the detection performance of synopsis-based

detector is similar to that of non-synopsis change detector if a distance function that is used to quantify the changes is preserved under the synopsis construction process. The DFT-based change detector was designed and evaluated as an example of synopsis-based change detector. As the data stream must be processed under the resource constraints with high speed, an incremental computation of DFT coefficients was exploited to further speed up the DFT-based change detector.

CHAPTER 5

Change Detection in Streaming Data by Clustering

Life is its own journey, presupposes its own change and movement, and one tries to arrest them at one's eternal peril.

(Laurens van der Post)

Contents

5.1 Introduction

Clustering evolving data streams not only finds new emerging patterns in the data, but it can also detect changes in the patterns, and data distribution. Detecting changes in the clustering structure can be seen in many real world applications.

For example, understanding how clustering model evolves can help us find out the sources responsible for spreading diseases [Barbará 2001]. Clustering evolving data streams can help to analyse the evolution of group behavior in social networks. Analysis of the evolution of clustering structure can provide the foundation for promptly making decisions. This chapter presents a method for detecting changes in multivariate streaming data by using the geometric and clustering approach. We then present method for building and maintaining of clustering by using the reactive approach in which this clustering-based change detection method is used to determine when to rebuild clustering.

The concept of change in this Chapter is understood as follows. Our change detection method uses the model fitting approach in which a change occurs when a new data item or block of data items do not fit the existing clustering.

5.2 Formal Model

The problem is formulated as follows. A data stream is an infinite sequence of elements

$$S = \{(X_1, T_1), .., (X_j, T_j), ...\} \tag{5.2.1}$$

Each element is a pair (X_j, T_j) where X_j is a d-dimensional vector $X_j = (x_1, x_2, ..., x_d)$ arriving at the time stamp T_j. Change detection can reduce to the problem of hypothesis testing below

$$\begin{cases} \mathcal{H}_0 & \neg change \\ \mathcal{H}_1 & change \end{cases} \tag{5.2.2}$$

where the hypothesis is a logical expression $change$ which indicates whether a change occurs.

Let W denote a sliding window. Let SC be the stream of clusterings that are continuously created by the stream clustering algorithm over the window W. As such, $\mathcal{A} : S, W \rightarrow SC$ is a mapping that creates a stream of clusterings SC from the stream of data points S by continuously applying the algorithm $\mathcal{A}$ on each window of data items W as the window slides on the data stream S.

The first truly scalable algorithm for clustering data stream called BIRCH [Zhang 1996] constructs a clustering structure in a single scan over the data with limited memory. BIRCH can work with limited resources. BIRCH consists of the following characteristics: incremental clustering, compact representation of clusters, and the ability to process data in a single pass. With the above characteristics, BIRCH is well suited for clustering large database as well as the evolving data streams. The underlying concept behind BIRCH called the cluster feature vector is defined as follows

Definition 5.2.1. *Clustering Feature [Zhang 1996] Given d-dimensional data points in a cluster:* $\left\{\vec{X}\right\}$ *where* $i = 1, 2, .., N$*, the Clustering Feature (CF) vector of the cluster is a triple:* $CF = (N,\ LS,\ SS)$ *where* N *is the number of data points in the cluster,* $LS = \sum_{i=0}^{N-1} X_i$ *is the linear sum of the data points in the cluster, and* $SS = \sum_{i=0}^{N-1} X_i^2$ *is the squared sum of the N data points. The cluster created by merging two above disjoint clusters* CF_1 *and,* CF_2 *has the cluster feature is defined as follows*

$$CF = (N_1 + N_2,\ LS_1 + LS_2,\ SS_1 + SS_2) \tag{5.2.3}$$

The advantages of this CF summary are:

- it does not require to store all the data points in the cluster.
- it provide sufficient information for computing all the measurements necessary for making clustering decisions [Zhang 1996].

Aggarwal et al. [Aggarwal 2003] extends this concept cluster feature vector for streaming context by adding the temporal components.

Definition 5.2.2. *Micro-cluster [Aggarwal 2003]. A micro-cluster for a set of* $d-$*dimensional points* $X_{i_1}, ..., X_{i_N}$ *with time stamps* $T_{i_1}, ..., T_{i_n}$ *is the* $(2d + 3)$*-tuple* $\left(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, N\right)$*, wherein* $\overline{CF2^x}$ *and* $\overline{CF1^x}$ *each corresponds to a vector of* d *entries. The definition of each of these entries is as follows*

- *For each dimension, the sum of the squares of the data values is maintained in* $\overline{CF2^x}$*. Thus,* $\overline{CF2^x}$ *contains* d *values. The* $p - th$ *entry of* $\overline{CF2^x}$ *is equal to* $\sum_{j=1}^{N} \left(X_{i_j}^p\right)^2$*.*
- *For each dimension, the sum of the data values is maintained in* $\overline{CF1^x}$ *. Thus,* $\overline{CF1^x}$ *contains* d *values. The* $p - th$ *entry of* $\overline{CF1^x}$ *is equal to* $\sum_{j=1}^{N} \left(X_{i_j}^p\right)$*.*
- *The sum of the squares of the time stamps* $T_{i_1}, ..., T_{i_n}$ *is maintained in* $CF2^t$*.*
- *The sum of the time stamps* $T_{i_1}, ..., T_{i_N}$ *is maintained in* $CF1^t$*.*
- *The number of data points is maintained in* N*.*

Definition 5.2.2 is the first definition of micro-cluster. In the later work, variants of micro-clusters are proposed to meet the specific requirements of the stream clustering algorithms. For example, in DenStream [Cao 2006], micro-clusters fall

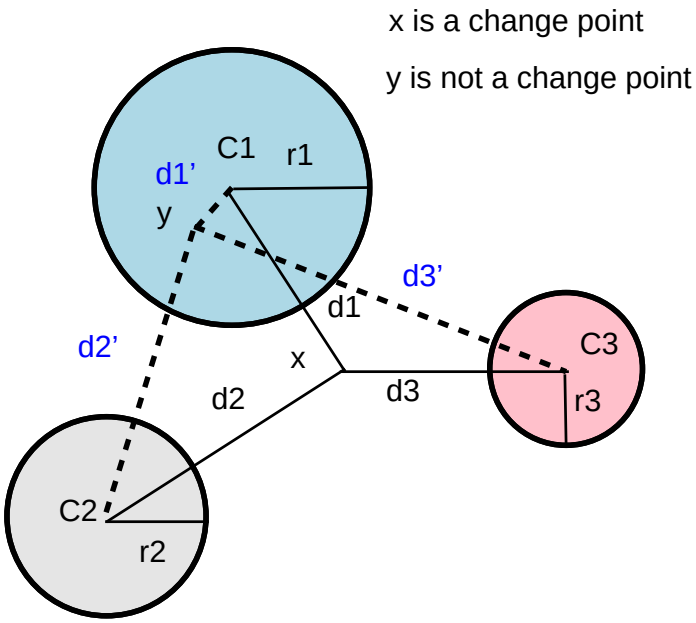


Figure 5.1: Change detection by clustering

into types such as core-micro-clusters, potential-c-micro-clusters, or outlier micro-clusters, which are used for density-based clustering.

Recently, Agarwal et al. have introduced the concept mergable summaries that are useful for many problems [Agarwal 2012]. As micro clusters can be merged into new one based on the additive property, micro clusters are mergable summaries that are useful for the problem of clustering streaming data as well as for the other problems.

5.3 Automated Change Detection by Clustering

This section presents an automated change detection algorithm in streaming multivariate data by clustering.

5.3.1 Automated Change Detection

As described in Section 5.2, depending on how to establish the logical expression $change$, we can derive the different change detection methods. This section presents an automated method for detecting changes in multivariate streaming data by using clustering and geometric approach.

Let x denote a recently arriving data item. A data point can be considered a change point if it is not a member of any cluster. In other words, data point x is a

change point if the following logical expression is true:

$$change = \bigwedge_{i=1}^{K} [d(x, center(C_i)) > radius(C_i)] \quad (5.3.1)$$

where $d(x, center(C_i))$ is the distance between a newly incoming data point x and the center of cluster C_i, and $radius(C_i)$ is the radius of the cluster C_i, for $i \in 1, .., K$.

Figure 5.1 illustrates how the clustering-based method for detecting changes works. There is a clustering of three clusters $C1, C2, C3$ in the reference window. As all the distances from data point x to three centers of clusters $C1, C2, C3$ are greater than the radiuses of three corresponding clusters, data point x is a change point while data point y is non-change point, because it is a member of cluster $C1$. In terms of model fitting, data point x does not fit to the existing clustering while data point y fits to the existing clustering.

Expression 5.3.1 shows how to check whether a data point is a change point. However, this logical expression is only used for one incoming data point. We now design a logical expression for checking whether the changes occur when sliding step is a block of b data points $x_1, x_2, .., x_b$. A block of data points is considered change if at least one change point belongs to this block. Therefore, the logical expression for checking whether a block of data points change in comparison with the reference window is as follows.

$$change(x_1, x_2, .., x_b) = change(x_1) \vee change(x_2) .. \vee change(x_b) \quad (5.3.2)$$

where $change(x_i)$ is determined by Expression 5.3.1.

All the data points are encoded as micro-clusters. A micro-cluster is a temporal cluster feature vector that is extended from the cluster feature vector. The center and the radius of each cluster can be easily computed from its micro-cluster. As described in Section 5.3.1, as the shape of cluster should be sphere cluster, the clustering algorithm used in the reference window is $K-means$, and the similarity measure is the Euclidean distance. The clustering-based algorithm for detecting change works as follows.

- *Initialization:* Read the first N data items from the incoming stream into the reference window w_1. The current window is the content of the reference window that slides one step to capture new data item. Windows $w1$ and $w2$ are overlapping by $N - 1$ data points. The next step is building a clustering the reference window w_1.

- *Continuous monitoring:* Check whether a change occurs by testing criteria for change as in Expression 5.3.1 or 5.3.2. If a change is detected, the detector makes an alarm, at the current time t_c, and the current window becomes

the reference window, construct the new clustering for the new reference window. The current window $w2$ always slides one step forward whether the recently arriving data item changes or does not change.

Algorithm 6 describes an clustering-based change detection algorithm with the arbitrary sliding step $step$, where $1 < step < N - 1$ and N is the window width. The input to the algorithm is a multivariate data stream S and the number of clusters K. If a change occurs, detector reports the change point. Our clustering-based change detector is based on the overlapping windows model. As the value of the data decreases over time, instead of storing for later analysis, data is immediately analyzed as it is produced. In particular, there are two windows: the reference window $w1$ and the current window $w2$. The current window is used to capture new items.

We note that the boolean function $change(C_1, blk)$ determined by Expression 5.3.2 checking whether at least a change point exists in the block of data points blk.

In a naive approach, we would compute the distances of the data points in the window with a newly arriving data item. Based these distances, we determine whether the recently arriving data point is a change point. As such, if the window of size N, we need to computes the distance function N times.

By using change condition specified in Expression 5.3.1 and 5.3.2, the number of comparisons reduces from N to K, where N is the size of sliding window, K is the number of clusters, $K \ll N$. The clustering-based method for detecting change is an automated method. Furthermore ,it can detect the changes in multivariate streaming data. If the changes frequently occur, we need to run the clustering algorithm $K - means$ in order to create the clustering from the reference window many times. This algorithm is efficient for the rarely occurring changes such as anomaly changes.

If the purpose is only to report whether a change exists in the recently incoming block of data, we can improve the performance of the clustering-based change detection as follows. Whenever the first change occurs in the newly incoming block, the algorithm makes an alarm that a change occurs without checking the rest of the block.

5.3.2 Maintenance of Clustering using Reactive Approach

The goal of clustering maintenance is to maintain a streaming clustering structure undergoing insertion and deletion of items when the sliding window moves on the data stream. This section describes how to builds and to maintain the clusterings emerging from data stream over a sliding window of fixed size using the reactive approach.

Algorithm 6: Clustering-based algorithm for detecting changes by using overlapping windows model

input : A data stream S, N is the window width, $slide$ is sliding step, K is the number of clusters

output: The messages reporting the changes occurred, and time t at which changes occurred

1. Initialization:

begin

- $t \longleftarrow 0$;
- $w1 \longleftarrow$ first N points from time t; `/* Each data point is encoded as a micro-cluster */`
- $\mathcal{C}_1 \longleftarrow Kmeans(w1, K)$;
- $w2 \longleftarrow slide(w1, step)$;
- $blk \longleftarrow newItemBlock(w2)$;

2. Continuous monitoring:

while *not at the end of the stream* **do**

- **if** $change(C_1, blk)$ **then**
  - $t \longleftarrow$ current time;
  - Report change occurred at time t;
  - $w1 \longleftarrow w2$;
  - $\mathcal{C}_1 \longleftarrow Kmeans(w1, K)$;
- $w2 \longleftarrow slide(w2, step)$;
- $blk \longleftarrow newItemBlock(w2)$;

A difficult problem in clustering of streaming data is that the underlying distribution of data stream evolves overtime. These changes can induce more or less changes in the clustering structure emerging from the data stream. Algorithm 7 describes how to build and maintain a clustering over sliding window by using the reactive approach. The input to the algorithm is a multivariate data stream S. If a change occurs, it rebuilds a new clustering. This clustering is returned as the result of some clustering query, or is stored in the history of clustering in streaming data warehouse.

Algorithm 7: Algorithm for building and maintenance of clustering over sliding window using the reactive approach

input : A data stream S and number of clusters K, the sliding step $step$, the window width N

output: Return clustering over sliding window at anytime

1. Initialization:

begin

$t \longleftarrow 0$;

$w1 \longleftarrow$ first N points from time t ; `/* Each data point is encoded as a micro-cluster */`

$\mathcal{C}_1 \longleftarrow K - means(w1, K)$;

2. Continuous maintenance of clustering:

while *not at the end of the stream* **do**

$w2 \longleftarrow slide(w1, step)$;

$x \longleftarrow newItemBlock(w2)$;

if *change* **then**

$t \longleftarrow$ current time;

$w1 \longleftarrow w2$;

$\mathcal{C}_1 \longleftarrow Kmeans(w1, K)$;

5.4 Related Work

The clustering-based change detection method proposed here is related to the work on automatic change detection and change detection in multivariate streaming data, and clustering-based change detection, and the reactive work on building and maintaing of model (as introduced in Section 1.3.2. Related work on the clustering-based methods for detecting changes were presented in Section 1.1.2. The following sections present related work on automatic change detection and change detection in multivariate streaming data.

5.4.1 Automated Change Detection

As automated systems require the capability of realtime processing, and adapting to the changing environments, automated change detection plays an important role in many automated systems. One of the models of realtime processing is data stream processing. For example, sensor networks need the automated change detection methods in which detection threshold must be adaptive to these changes of the environment. Automated change detection method also plays important role in many mobile robotic applications [Neuman 2011]. For example, Neuman et al. [Neuman 2011] have proposed an online change detection method for mobile robots based on the segmentation approach. Recently Hirte et al. [Hirte 2012] have developed Data3, a Kinect interface for human motion detection. In fact, Data3 is a kind of system capable of detecting the changes in spatial-temporal streaming data.

Automated systems should be capable of automatically detecting the changes without the given detection threshold. Some change detection methods can automatically tune the detection thresholds so that the rate of false alarms is not greater than a given rate of false alarms. Gustafson and Palmquist deal with the problem of automated tuning of change detectors with given false alarm rate [Gustafsson 1997]. Their approach computes the detection threshold by estimating a parametric distribution. The advantage of this method is that they can predict detection threshold with no or few false alarms from the used data. However, it is parametric method.

The automatic selection of threshold is of special importance. Alippi et al. [Alippi 2012] have presented an automated change detection method for streaming data based on Hidden Markov Models. This HMM-based method is an automated change detection method by thresholding. Their algorithm consists of the following steps: model the relationships among data streams a sequence of time invariant linear dynamic system; model the evolution of the estimated parameters of the model by Hidden Markov Model; evaluate the likelihood of new parameters; detect change based on a given threshold. If the likelihood is less than a given threshold, a change is detected.

5.4.2 Change Detection in Multivariate Streaming Data

Most methods for detecting changes in streaming multivariate data are based on the multivariate tests. As the problem of testing statistical hypotheses in high dimensional data is particularly challenging and the change detection in streaming data requires to respond to the changes in nearly real-time, change detection in streaming multivariate data is challenging.

There are two approaches to dealing with the problem of change detection in streaming multivariate data. The first approach is based on the transformation that

converts a multivariate data stream into a univariate data stream. Change detection is then performed on the univariate data stream. Dasu et al. [Dasu 2009] have used this approach to design a change detection method for streaming multivariate data. In particular, a multivariate data stream is converted into a univariate data stream. The change detection task is performed by using Kolmogorov-Smirnov test. Similarly, Kim et al. [Kim 2009] have recently proposed the concept called the detection stream. A detection stream is a univariate stream that is generated by mapping a multivariate stream into stream of dissimilarity measures quantifying the difference between two windows.

The second approach is developing new methods for detecting the changes in multivariate streaming data. Kuncheva has recently presented a general method for detecting changes in multivariate streaming data by using likelihood ratio-based test [Kuncheva 2011]. Closely related to our work is the segment-based method for detecting in multivariate data stream proposed by Chen et al. [CHEN 2009]. Gretton et al. [Gretton 2012] present a kernel two-sample test that can check whether two multivariate samples coming from the same distribution. Furthermore, a kernel two-sample test with the linear computational complexity suitable for streaming environment is proposed.

In contrast to the previous work, Section 5.3 introduces a new method for detecting the changes in multivariate streaming data by using the geometric and clustering approach.

5.5 Evaluation

The clustering-based change detector was written in Java. An algorithm for detecting changes in streaming data should be evaluated in three aspects scalability, accuracy, and monitoring capability. The detection accuracy of a change detection method depends on the window width and the number of clusters. The experiments on change detector using clustering were divided into two groups in terms of detection accuracy, running time, and memory consumption. The first group of experiments evaluated the effectiveness of window width on the clustering-based change detector. The next group of experiments studied the effectiveness of the number of clusters on the performance of the clustering-based change detector. We analyzed the effect of the window width, number of clusters on the number of change points detected by the clustering-based detectors for change.

One of the challenges in assessing a change detection algorithm is the lack of ground truth data. In particular, the evaluation of a method for detecting changes in multivariate streaming data is more challenging. Therefore, the synthetic data was used for evaluating the performance of the change detection algorithms. The synthetic data set to evaluate the accuracy of our clustering-based change detec-

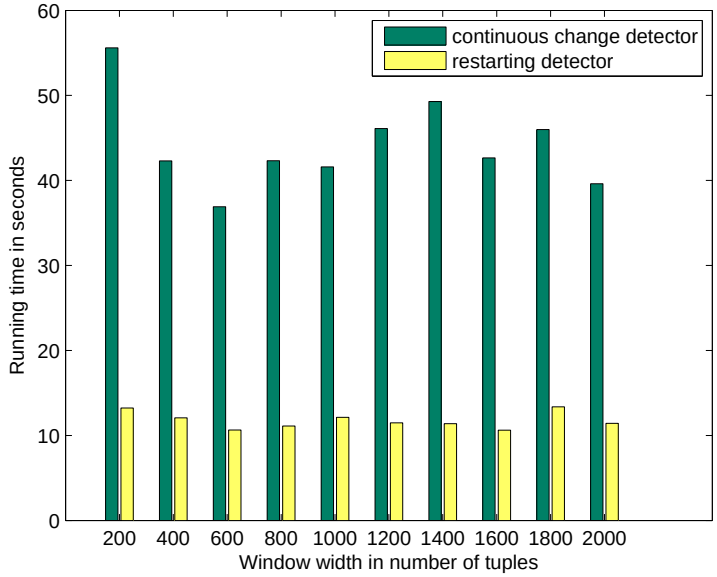


Figure 5.2: Effectiveness of window width on the performance of clustering-based method for detecting changes in terms of running time

tion algorithm is the streaming data set HyperP [Zhu 2010]. HyperP stream is a synthetic data stream of gradually evolving (drifting) concepts. The Hyper Plane stream consists of 100000 instances, and each instance consists of 10 attributes. These instances may fall into one of 5 classes (clusters). To see the advantages of the continuous detector of change over the restarting detector of change, we ran both continuous detector and restarting detector. Restarting detection of changes means that a detector will be reset if a change is detected while non-restarting detector continuously detects the changes. Non-restarting detection of change is also called continuous change detection.

We ran the detectors by clustering for two cases. First, to see the effect of the window width on detection performance of the detectors, we fixed the number of clusters while changing the window width. Second, to see the effect of the number of clusters on the detection performance of the detectors, we fixed the window width while the number of clusters changes.

5.5.1 Effectiveness of Window Width

To study the effectiveness of window width on the performance of clustering-based change detector, the number of clusters was fixed to 5, and the window width was varied in the range 200,400,600,800,1000,1200,1400,1600,1800,200.

Figure 5.2 shows that the restarting detector runs faster than the continuous

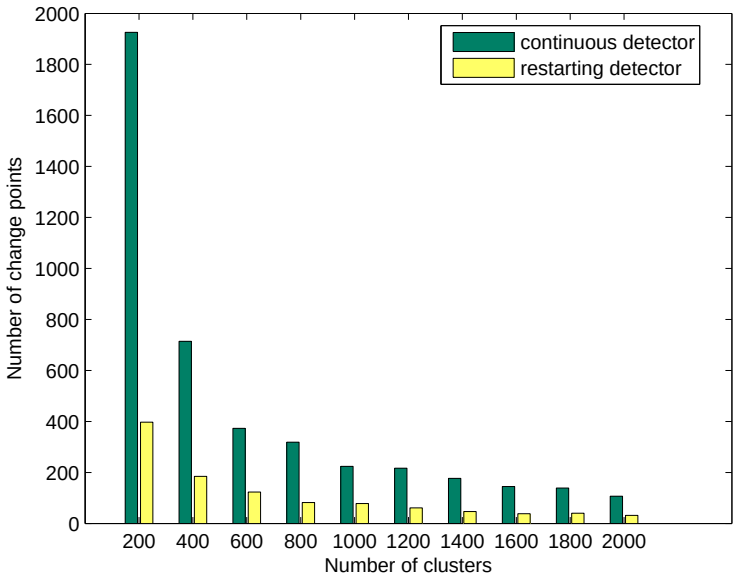


Figure 5.3: Effect of the window width on the number of change points detected by detectors

detector because the restarting detector was reset every time a change is detected. The speed of restarting detector comes at the price of its detection accuracy. As Figure 5.3 shows, the number of detected change points reduced when the window width increased. The continuous detector detected much more change points than the restarting detector. Because the restarting detector was reset every time a change was detected, many points in the reference window in the resetting phase were ignored by detectors.

Figure 5.7 shows that, the change point detection algorithm by clustering required more than the interval-based change detection. As shown in Figure 5.4, the amount of memory used by clustering-based change detection scales with the window width.

5.5.2 Effectiveness of Cluster Number

To assess the effectiveness of cluster number on this clustering-based change detection, the window width was fixed to 1000 tuples, and the number of clusters was varied in the range $(2, 3, 4, 5, 6, 7, 8, 9, 10)$. The results of this experiment (Figure 5.5) shows that, the running time of a clustering-based change detector increases with the increasing number of clusters. The time needed to determine whether a newly incoming data point is a change point increases, as the number of comparisons between the distance from the newly incoming point to the centers of clusters

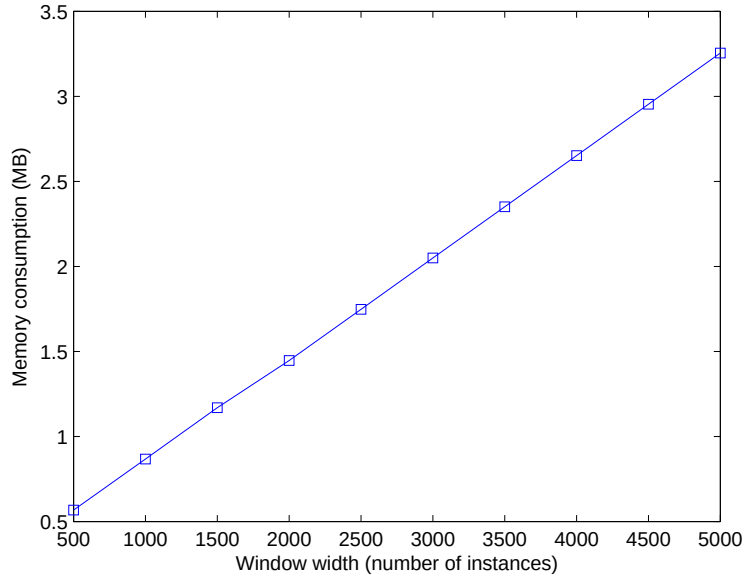


Figure 5.4: Effectiveness of window width on the performance of clustering-based change detection method in terms of memory consumption

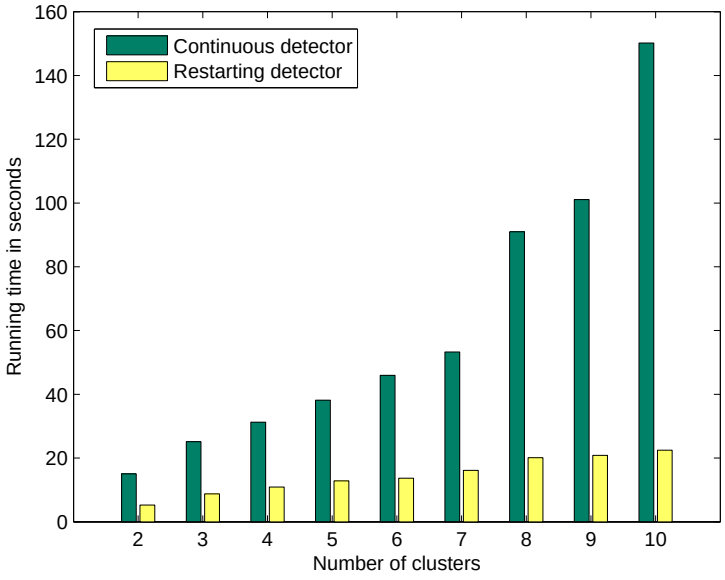


Figure 5.5: Effectiveness of the cluster number on running time of the clustering-based change detection method

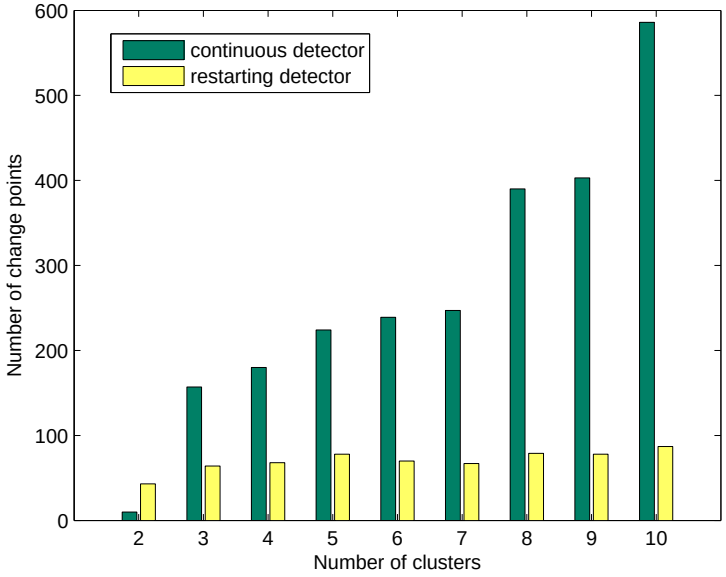


Figure 5.6: Effectiveness of number of clusters on the change points detected by the clustering-based change detector

and the corresponding radiuses of clusters increases with the number of clusters. Figure 5.6 shows that, the number of change points increases when the number of clusters increases.

If the changes frequently occur, we need to run the clustering algorithm in order to create the clustering from the reference window many times. Because the clustering process consumes a lot of time. Therefore running time of a clustering-based change detector increases with the increasing number of change points. As such, the efficiency of the clustering-based change detection method heavily depends the selection of the number of clusters.

This could be unsuitable for streaming environment with high data speed. The performance of our algorithm would be much better if an incremental clustering were used to generate the clustering in the reference window. It should be better to use an unsupervised clustering algorithm with the number of clusters is unknown in advance such as DBSCAN. As such, the clustering-based change detection algorithm will adapt to the environmental changes.

5.5.3 Effectiveness of Sliding Step

This group of experiments study the effect of sliding steps on the performance of clustering-based change detection method with the arbitrary step of sliding. We fixed the window width to 500 tupless. The number of clusters was set to 5. We ran

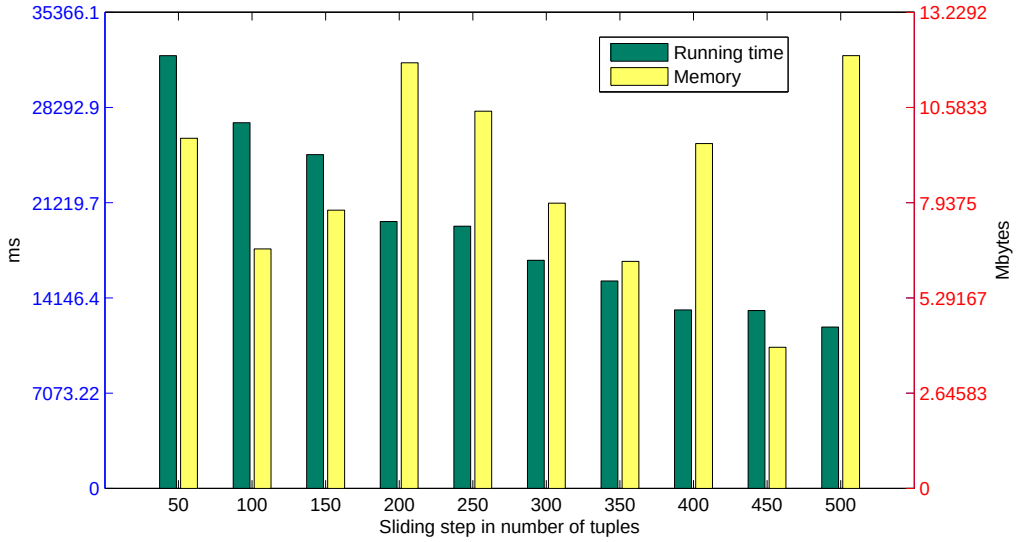


Figure 5.7: Effectiveness of sliding step on the performance of clustering-based change detection method in terms of running time and memory

the clustering-based change detector with the sliding step varying from 50, 100, 150, 200, 250, 300, 400, 450, 500 (tuples).

5.5.4 Evaluation on Clusterings using Reactive Approach

This group of experiments shows the effectiveness of building and maintenance of clusterings using the reactive approach on the detection performance of clustering-based change detection algorithm. The experiments were executed on the synthetic data streams HyperP. The window size was fixed to 500, and the number of clusters was varied from 4 to 7. Table 5.1 shows the results of interest include the number of detected change points, the number of clusterings that are generated, and the running time. The running time increases with the number of clusters because the larger number of clusters is, the more number of comparisons must be done. The number of clusterings depends on the number of clusters. In this group of experiments, we compare the performance of two approaches to building and maintaining clustering over sliding window: periodic approach and reactive approach. The results we want to determine include: running times of two methods, the number of clusterings that are generated. Results of reactive method for building and maintaining clustering over sliding window using the reactive approach are as follows: number of clusterings that are generated is 1305;running time is 57330 ms ;used memory is bytes is 10 MB. Table 5.1 shows that the number of clusterings generated by the reactive approach is equal to the number of change points plus one. In

Cluster number	Number of changes	Number of clusterings	Running time
4	897	898	35567 ms
5	1304	1305	54913 ms
6	1269	1270	59422 ms
7	1336	1337	65998 ms

Table 5.1: Effect of cluster number on the number of detected change points and the number of clusterings

addition to the initial clustering, if a change occurs, a new clustering is generated. In general, the number of change points increases with the number of clusters.

5.6 Summary

In this chapter, we have proposed a novel change detection method for streaming multivariate data by using clustering. Our change detection method uses the model fitting approach in which a change occurs when a new data item or block of data items do not fit the existing clustering. The salient features of clustering-based detector are that it can work well with the multivariate streaming data, and it is an automated change detection method for streaming data. We have presented an algorithm for building and maintaining clustering emerging from the evolving data streams by using the reactive approach. The method for building and maintaining clustering in this chapter can be extended for distributed environment.

Part III

Distributed Detection of Changes in Streaming Data

CHAPTER 6

Distributed Detection of Changes in Streaming Data

Sometimes it's the smallest decisions that can change your life forever.
(Keri Russell)

Contents

6.1 Introduction

Distributed data processing is necessary because of the following reasons. Fist, it is economical to process data that is distributed on many nodes. Second, it is highly scalable to process distributed queries. Third, by distributed processing, processing load is equally distributed to every node. This chapter focuses on the problem

of distributed change detection in streaming data. Some commonly used models for distributed streaming computation include one-shot computation, gossip-based computation, continuous computation [Cormode 2005].

The advent of technology has eased the development of small, low-power sensors. Sensor networks have been deployed in many applications such as military, environmental, health, and commercial applications [Akyildiz 2002]. The fundamental tasks of sensor network are to sense events, to detect the events, and to transmit the detected information to the sink node. A well-known and important problem is to detect changes in sensor data, which is considered one of the core issues when designing a sensor network [VISION 2007]. Advances in the sensor technology results in a massive amount of streaming sensor data. In wireless sensor networks, data streams are often generated from sensors scattered in different locations.

Change detection algorithms contribute to the success of emerging sensor networks. Designing and developing a change detector in a sensor network poses the following challenges. Sensor data streams produced by sensor networks must to be processed in real time. Change detection in sensor streaming data is a special case of change detection from distributed data streams. To make matters worse, in sensor network context, computing on data streams occurs in resource-limited conditions such as low power batteries, small memory, and small processing power.

The scenario of wireless sensor network for wildfire warning systems presented in Section 1.1.1 will be used to illustrate our distributed framework for detecting changes in streaming data that is created from multiple sources.

To solve the problem of distributed change detection, the tradeoff among the detection accuracy, the model size, and communication complexity must be considered [Palpanas 2003]. Three criteria for evaluating the performance of a distributed computation framework include: communication-efficiency, space-efficiency, and time-efficiency. The major contributions of this chapter are as follows

- We present a distributed framework for continuously detecting changes in streaming data that is created from multiple sources. The proposed framework consists of the remote-site processes and the coordinator-site process. Each remote-site detects the changes in streaming data , and transmits the local decision on detected changes to the coordinator. The coordinator-site process receives the local decisions from the remote sites, and fuse these local decisions in order to reach the global decision on detected changes.

- We consider a case study: change detection in streaming data that is created by a wireless sensor network. We present a distributed change detection framework by decision fusion based on gossiping [1].

[1]This chapter uses the gossiping based model for fusing decisions, which is the joint work

Our proposed framework is flexible since we can derive many different instances of the framework by choosing different local change detection algorithms at the node-level and different strategies for making decisions at the fusion center. As different applications need to construct different modules of the framework, we can not enumerate all the combinations of the framework.

A salient advantage of our framework over the previous work lies in the fusion and decision making process at the fusion center. Distributed change detection algorithm based on decision fusion has the following advantages:

- Improve detection performance at the coordinator by fusing local decisions from local change detectors at different locations. Local decisions that are received by the coordinator can be missed, incorrect due to the local change detection algorithms, and the communication process. The distributed change detection framework based on decision fusion can be fault tolerant by fusing local decisions made by change detectors at the sensor nodes near the target source (fire).

- Reduce the communication overhead, and thus increase the lifetime of the sensor network.

- Separate decision fusion model with the information exchange and dissemination process, where local nodes cares only how to improve the accuracy of global detection with the decision fusion model; while the communication cost is decided by information exchange protocols.

We formulate the problem of detecting changes in streaming sensor data in Section 6.2. Section 6.3.1 describes a distributed detection method for streaming sensor data by using gossip scheme. Simulation and analysis are presented in Section 6.4. We summarize this chapter in 6.6.

6.2 Problem Formulation

Our distributed framework for detecting changes in streaming data includes two functions as follows:

- *Local change detection*: Due to the unlimited nature of data streams and the restriction of computational resources and bandwidth, the sliding window-based approach is the best choice for the change detection algorithm in the data stream.

---

presented at MDM 2011 with the coauthors Jin Yang and Kai-Uwe Sattler

- *Online decision fusion:* We note that in the wildfire warning system, only a portion of nodes sense, and detect the events. To detect a fire event, only a portion of nodes detect the abnormal changes of the measures such as temperature, or humidity. Therefore, there will be a group of sensor nodes sense and detect the changes. This group includes a cluster head, and member nodes. The cluster head is some node which is elected based on its energy level [Heinzelman 2000]. The cluster head of this group of sensors communicates with the rest of wireless sensor network in order to propagate the event. Assume that each node reaches the local detection decision by its local change detector. All the local decisions made by the group of sensed nodes are transmitted to the coordinator in order to get the global decision.

6.2.1 The Coordinator

This section presents the objective of the coordinator and describes the function of the coordinator. The goal here is to make the global decision at the fusion center based on the local decisions made by the local change detectors at the nodes. The task of the coordinator is to decide whether a global change occurs based on the local detection decisions received from the remote sites as soon as possible. As global detection decision greatly depends on the frequency of updates from the local change detectors, we must answer the question how much and how often information should be propagated from sensor to the fusion center.

There are two approaches to the information exchange between the coordinator site and remote sites. In the data fusion approach, sensor senses and transmits information of its ambient environment to the fusion center. Fusion center is responsible for making decision, and further relaying information. This approach incurs high communication cost. In the decision fusion approach, the information exchange is the exchange of soft or hard decisions. Each node makes decision based on its local observations and broadcasts the decision to all the other nodes. Each node updates its decision based on the received information and broadcasts again its new decision. This process repeats until consensus among the nodes is reached. Compared with data fusion, decision fusion can reduce communication cost because sensors only transmit the local decisions of small data structures.

The design of optimal decision rules for distributed change detection is one of the central issues. In other words, the problem we should deal with is how to determine the global test statistic at the coordinator. There are two approaches to designing optimal decision rules for distributed change detection [Veeravalli 2012]. The first approach is based on the assumption that the detection performances of the local change detectors are known to the fusion center[Varshney 1986]. Our framework works under the assumption that the channel is ideal. In another words, the decisions received at the fusion center are the same as the original decisions made

by local change detectors. We note that the coordinator has complete knowledge of the hit rate and the false alarm rate of each sensor as described in Chapter 3 and Chapter 4. In particular, the change detector using the absolute difference of two consecutive data items is called the ground-truth change detector. From the ground-truth change detector, and the distance-based change detector, the rate of hit and the rate of false alarm can be determined. However, it is difficult to compute the hit rate and the false alarm rate in real-world. The second approach does not require the detection performances of the local change detectors. The fusion center uses the counting rule to reach the detection decision on the system level [Ray 2011].

A problem that arises is how to determine the period of decision fusion. There are two possible solutions to decision fusion at the coordinator. The first solution called periodic fusion uses the width of sliding window as the periodic time for fusing data. The second solution called change-based fusion fuses decisions only when detecting the changes.

Let $D = (D_i, .., D_N)$ denotes the global vector of local detection decisions. Having received all the data, the coordinator make a final decision about the presence or absence of a global change. Each local change detector uses a decision rule to make a decision D_i, for $i = 1, .., N$. The coordinator determines the global detection decision for the whole system based on the individual decisions.

6.2.2 The Remote Site

Local change detector at each remote site is an important component in our distributed framework for detecting changes in streaming data. Chapters 3 and 4 describe the methods for detecting changes in a single data stream in detail. Depending on the requirements of each application, we can select appropriate methods such as the Euclidean distance-based detector, the Manhattan distance-based detector, DFT-based change detector, or clustering-based change detector as described in Chapters 3, 4, and 5.

Without losing generality, we assume that there is a local change detector i at the remote site i, for $i = 1, .., N$, where N is the number of sensor nodes. In the parallel decision fusion, each remote site sends its local decision to the coordinator directly.

6.2.3 Decision Structure

This subsection describes the decision data structures including local decision, and global decision. As mentioned earlier, there are two kinds of decision fusion schemes: decision fusion with the information on detection accuracy, and decision without the information on detection accuracy. Distributed detection of changes

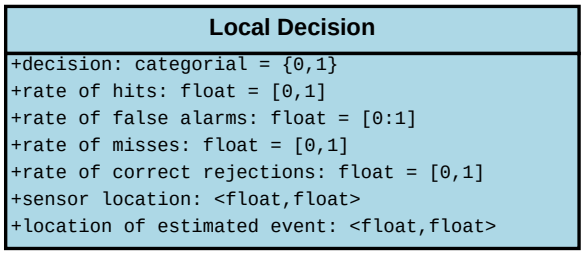


Figure 6.1: Data structure of a decision made by local change detector

presented in Section 6.3.2 does not require the information of detection accuracy of the local change detector. Therefore, local decision is a binary variable representing the decision on change. We will describe the decision structures for distributed change detection with information on detection accuracy of the local change detectors.

In addition to detecting changes, remote sites should encode the information of detected changes in an appropriate format to transmit over the network reliably and efficiently, and to make decisions. A change detector can make decisions on the detected changes based on its own information, or based on its own information in conjunction with the information from its neighbors.

As Figure 6.1 shows, a local detection decision consists of the following attributes: *decision* is a binary variable that describes the state of observation detected by a local change detector. If a change occurs then $decision = 1$, otherwise $decision = 0$; *rate of hit*: H_i, false alarm rate: FA_i, miss rate: MR_i; correct rejection: CR_i; *sensor location:*; $< x_i, y_i >$; Estimated location of event e: $< x_e, y_e >$;*estimated location of event*. The information encoded a local decision structure is used for the later fusion process at the coordinator as described in Formula 6.3.1.

Each remote site sends its local decision to the coordinator. Each local decision made by a local change detector should be fused and forwarded back to the sink as a global change detection decision. The coordinator computes the global decision on the global change based on the local decisions received from remote sites by using some decision rule as described in Section **??**.

The data structure of the global detection decision include three components: estimated position of event $< X, Y >$, global decision of detection, which includes detection result, and detection accuracy determined by local decisions.

6.3 Distributed Change Detection

As discussed earlier, the salient feature of local change detection algorithm is that it is flexible. Based on the selection of network architecture and decision making

scheme, we can develop various distributed change detection schemes. In this section, we present three possible approaches to reaching the global decision at the coordinator: decision fusion using gossip protocol, decision fusion scheme using false discovery rate, and distributed detection using clustering.

6.3.1 Distributed Change Detection by Gossiping

This section describes a distributed framework for detecting of changes in streaming data by gossiping-based fusion. A distributed change detection framework should be fault-tolerant. Fault-tolerance is that a distributed system works well in case of the failure of the node or the communication link. Developed by Xerox corporation in 1987, gossip protocol used for domain name services is very simple but robust (requiring very few guarantees from the underlying communication systems), yet it assures that the effect of updates is really reflected in the replicas [Demers 1987]. In fact, a wireless sensor network can be seen as a distributed database system, thus gossip protocol can be exploited for wireless sensor network in some important tasks such as routing problem, and data fusion. Hence, the gossip protocol will be used as a scheme for quickly spreading changing information sensed by the sensor nodes in this work.

To detect event, two fundamental tasks must be performed: estimation of event location, and decision fusion. The communication protocol we use is the multi-source multi-sink information protocol. As a node can move from this location to others, a new node can be added to the network, and a battery-depleted node can be removed from the network, thus a change detection scheme in wireless sensor network must adapt to the location changes. To estimate the location of events, we can use the location information sensed by sensor networks by the location estimation methods such GPS-based estimation of location, or the positioning algorithm Trilateration. It is assumed that the location of an event lies in the sensing range of the sensor network. In decision fusion, each sensor performs local change detection using some local change algorithm and updates its decision based on the received information and broadcasts again its new decision. This process repeats until consensus among the nodes is reached. By aggregating the results from different sensor nodes, the location of an event can be estimated more accurately.

Depending on the distance between the location of an event and sensor nodes, a sensor may or may not detect the changes of the target. Let $detectable$ be a boolean variable that represents the ability to detect a change of the event. If a sensor cannot detect the changes of the target due to some reasons, for example, the event lies outside of the sensing range of the sensor, $detecable = false$. If $detectable = true$, the event lies in the sensing range of the sensor. The detection result of the change of the event returns 1 with some detection accuracy (including the rate of hits, the rate of false alarms, the rate of missed detection, and the rate of correct re-

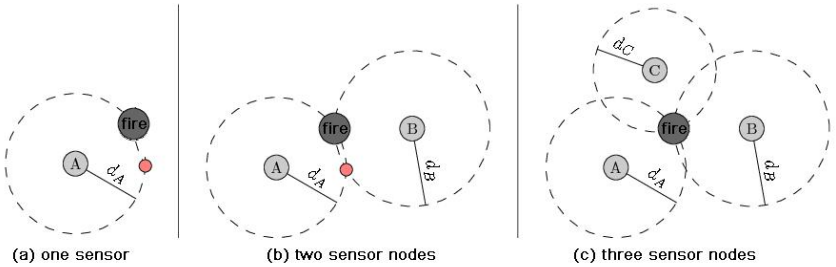


Figure 6.2: Location estimation with different number of sensor nodes

jection), otherwise it returns 0 (no change occurs). Detection accuracy depends on the change detection algorithm that is selected at the sensor node. Global detection decisions are built from the local decisions made by the sensors.

To reach the global detection decision at the coordinator from the local decisions D_i, we use a fusion framework of three components: decision fusion model at sensor node; algorithm for clustering sensor nodes, and the multi-source multi sink communication protocol. The models, algorithms, and protocols will be described in detail in the next sections.

6.3.1.1 Decision Fusion Model

Decision fusion model runs on local sensor nodes to fuse local detection decisions of change detectors from the different sensor nodes. The decision fusion model runs as a plug-in on each sensor node, which can be implemented according to different requirements. This decision fusion model should be application specific. A decision fusion scheme at each sensor node aggregates the decisions by different sensor nodes, and estimates the location of the event. From the estimated location of the event, the system can determine more accurately whether this event is a true event or a false alarm. The quality of sensor observations depends on the distances between the sensors and the event location. Depending on the requirements of estimation accuracy and the available information from nearby sensor nodes, we may implement the different models of location estimation on sensor nodes. There are some estimation techniques such triangulation techniques, or estimation technique considering the interference of obstacles that result in a region where the anomaly event occurs [**?**]. To estimate the location of fire in the scenario of wildfire warning system as shown in Figure 1.1, all the information from sensor nodes such as local decisions made by sensor nodes, and the sensing ranges of sensor nodes must be available for aggregation. We assume that the intersection of sensing ranges of the sensor nodes is the location of fire. Figure 6.2 shows that, the more sensor nodes are used to sense the event, the more accurate the estimated location of the event is.

To deal with the malfunction of sensor nodes or the false alarms of the local change detector at sensor nodes, each sensor node should aggregate its local decision made by its local change detector with a decision model. In order to overcome the occasional local failure either in value or in change detection, the sensor nodes should aggregate the local detection result with a decision model. Two any remote sites can directly communicate with each other.

The decision aggregation model works as a plug-in according to different applications. In the scenario of fire detection and warning, we use a history detection accuracy based weighted method to fuse all the detection results near the reported event location. Each sensor node i uses a queue to maintain all of the local decisions made by change detectors of sensors j near the the location of the event. We use the optimal decision rule proposed by Varshney et al. [Varshney 1986] to create the global decision rule at the coordinator. The optimal fusion is given by

$$\Lambda = \sum_{i=1}^{N} \left[u_i \ln \frac{p_{d_i}}{p_{fa_i}} + (1 - u_i) \ln \frac{1 - p_{d_i}}{1 - p_{fa_i}} \right] \tag{6.3.1}$$

where u_i is the local detection decision at the remote site i with the detection accuracy represented by the probability of detection p_{d_i} and the probability of false alarm p_{fa_i} respectively. Decision fusion is a continuous process. The more messages a node receives from its neighbors, the more accurate the fusion result is.

The decision fusion process should be finished near the event location in order to reduce communication cost, and it only makes sense to fuse the local change detection results which are within the detection range of the location of event. In other words, it is necessary to cluster of sensor nodes that are around the event. To reduce the amount of information transferred in wireless sensor network, and to distribute energy dissipation among the nodes, methods for clustering sensor nodes will be used. The method for clustering sensor nodes should be triggered by the first sensor node which detects the event. The problem of clustering sensor nodes based on the similarity of sensor readings will be discussed in Chapter 7.

6.3.1.2 Information Dissemination

The local detection result of the warning should be disseminated among the network. To reduce the redundant messages, the gossip protocols are used. Each process running at sensor node consists of two threads: active thread and passive thread.

Continuous queries can be classified into the following categories: push-based query processing model, pull-based processing model, and push-pull processing model [Madden 2002]. Push-based query processing model is a model in which remote sites publish/advertise their information to the coordinator site for further processing. Push-based query processing model is used query processing in sensor

network. Pull-based query processing model is a model in which the coordinator disseminates the interests of the user to a set of remote sites. In this case, the interest may be related to an attribute of the physical field (query) or an event of interest to be observed (task). Based on the user interest, the remote sites respond with the requested information. Push-pull query processing is a model in which both push- and pull- based query approaches can be used in a network, in which both the remote sites and the coordinator site are actively involved in query processing. Madden and Franklin have combined both push-based and pull-based query processing model in Fjords, an architecture for queries over streaming sensor data [Madden 2002]. Therefore, we propose three types of information transmission by using gossip scheme as below

- *Push-gossip scheme:* In the push-gossip scheme, information owner proactively sends its new data to its selected neighbors, but do not exchange information with the selected neighbors.

- *Pull-gossip* In pull-gossip scheme, the owner of new information waits passively until it receives the requests for data.

- *Push-pull-gossip scheme:* Each process at each node consists of two types of threads: active thread and passive thread. An active thread is executed once in each consecutive time unit at a randomly picked time. A passive thread receives messages from its neighbors, and sends its own message to its neighbors from whom it received information.

We use the push-gossip, pull-gossip, push-pull-gossip, and application-layer broadcast as plug-ins in the framework to meet requirements of different applications. The different features of these gossip protocols are discussed more detailed in [Yang 2011].

The framework proposed here uses algorithm 8 and algorithm 9 to fuse local decisions and disseminate results over the sensor network. The network protocol used in the gossip scheme is comprised of many parallel topologies, where each node may play both the remote site role and the coordinator role. At some time, a sensor node serves as a coordinator. The coordinator collect the neighbors' states including the estimated locations of the event and the local decisions. Based on its own state and the neighbors'states, the coordinator makes its own decision, and continues propagating its detection decision over the rest of network. The push-based gossip protocol uses the active threads to collect the local detection decisions. The pull-based gossip protocol uses the passive threads to collect the local detection decisions. The push-pull-gossip protocol uses both the active threads and the passive threads to collect the local detection decisions. The active threads proactively select the coordinator's neighbors and propagates its detection decision

over the network. In contrast, the passive threads passively continuously receives the states of its neighbors and propagates the detection decision over network. In chapter 3 and chapter 4, a local change detector was evaluated without the neighbors' state information. The procedures used in Algorithm 8 and algorithm 9

Algorithm 8: Decision fusion using gossip to exchange information: active thread

input : The states of the node p and the states of the neighbors of p
output: The new state of node p

foreach *consecutive iteration* **do**
- $state_q \longleftarrow receiveState()$;
- $state_p \longleftarrow localChangeDetector()$;
- send $state_p$ to every node q;
- $decision_p \longleftarrow decisionFusion(decision_p, decision_q)$
- $state_p \longleftarrow updateState(decision_p)$

Algorithm 9: Decision fusion using gossip to exchange information: passive thread

input : The states of the node p and the states of the neighbors of p
output: The new state of node p

while *true* **do**
- $state_q \longleftarrow receiveState()$;
- $state_p \longleftarrow localChangeDetector()$;
- send $state_p$ to every node q;
- $decision_p \longleftarrow decisionFusion(decision_p, decision_q)$
- $state_p \longleftarrow updateState(decision_p)$

include $receiveState()$, $localChangeDector()$, $decisionFusion()$, $send()$, and $updateState()$. The procedure $receiveState()$ implements the state-receiving process of the coordinator. The procedure $localChangeDector()$ codes some change detection method introduced in Chapter 3 and 4. The procedure $send()$ implements the process sending the state of some node to other nodes. Finally, the procedure $updateState()$ is used to update the state of a sensor node. The procedure $decisionFusion(dec_p, dec_q)$ generates the decision for node p from its own decision and the decisions of its neighbors q.

6.3.2 Distributed Detection by False Discovery Rate

Our proposed framework can control the false discovery rate in order to achieve the better performance of the fusion and decision making at the fusion center.

To reach a global detection consensus and assure the accuracy of the performance of the global change detection, the previous approaches require that the local decisions accompany their corresponding probability of detection sand probability of false alarm. In real world applications, determining the probability of detection and false alarms is difficult. To overcome this drawback, Ray and Varshney [Ray 2008] have recently presented a novel approach to the problem of distributed detection in wireless sensor networks using dynamic sensor thresholds.This approach is concerned with the distributed detection with the assumption that the received signal power decays as the distance between fusion center and target increases.

Let $u = (u_1(t), ..., u_M(t))$ be the decision vector used to reach the consensus. To find the final decision on global change, we count the local detection consensuses by using the well-known basic counting algorithm in the data stream in sliding window of size M where M is the number of sensors. Therefore, the problem of continuously detecting changes in multiple distributed data streams becomes more challenging. A data stream that is generated at the coordinator is an asynchronous binary stream.

Continuous distributed monitoring of changes in streaming data is related to asynchronous data streams. Now the problem of distributed change detection in the parallel fusion architecture casts to the problem of testing simultaneously testing M hypotheses. We suppose that M_0 hypotheses are true, that means M_0 sensors detect changes locally. After receiving local decisions from the sensors, the fusion center makes a final decision about the global change by using the fusion rule:

$$\begin{cases} \mathcal{H}_0 & \Lambda \leq T \\ \mathcal{H}_1 & \Lambda > T \end{cases} \tag{6.3.2}$$

where $\Lambda = \sum_{i=1}^{M} u_i$ and T is the global threshold. In this work, we use a fusion rule proposed by Niu et al [Niu 2004]. This fusion rule uses the total number of detections (encoded as 1) that come from local change detectors at sensors. The false alarm rate is given by

$$P_{FA} = P(\sum_{i=1}^{M} u_i \geq T \mid \mathcal{H}_0) \tag{6.3.3}$$

	Declared $\mathcal{H}_0$	Declared $\mathcal{H}_1$	Total
$\mathcal{H}_0$ True	U	V	M_0
$\mathcal{H}_1$ True	T	S	$M - M_0$
Total	M-R	R	M

Table 6.1: The result matrix of M local change detectors

6.3.2.1 False Discovery Rate

Benjamini and Hochberg present a method for testing multiple hypotheses able to control the false discovery rate. False discovery rate (FDR) is defined as the fraction of false rejections among those hypotheses rejected. False Discovery Rate method is a scalable approach to hypothesis testing [Benjamini 2005]. Each statistic test consists of statistic test Z_i and p-value P_i.

As such, the false discovery rate is given by

$$L = \frac{V}{V + S} \tag{6.3.4}$$

The probability of false alarm is given by $P_{FA} = P(\sum_{i=1}^{M} u_i \geq T \mid \mathcal{H}_0)$. The advantage of this approach is that we are uninterested in how local change detector works as well as the probability of hit and probability of false alarm at each sensor.

6.3.2.2 Estimation of Global Decision

As mentioned above, the first step in making global decision at the fusion center is to count the number of local detections. Since the order in which local decisions arrive at the fusion centers may be different from the order in which they were generated by local change detectors, the problem of continuously counting the number of detections reduces to the problem of computing the sum of elements in an asynchronous binary data stream in a sliding window. To solve this problem, we use an algorithm proposed by Bush and Tirthapura [Busch 2007]. The goal of this algorithm is to maintain a sketch which can return the sum of values of streaming elements in the sliding window. Continuous distributed monitoring of changes in streaming data is related to asynchronous data streams. Sensor nodes make their local decisions. The coordinator fuses these local decisions in order to make the global decision on the occurred event. The received order of local decisions at the coordinator may differ from the time order in which local decisions are made. Let $u = u_1, u_2, .., u_i, ..$ be an asynchronous data stream arriving at the coordinator where $u_i \in 0, 1$ Their algorithm works on the basis of a data structure called splittable histogram. The space and time required by this algorithm is given by the below theorem.

Theorem 6.3.1. *[Busch 2007] The worst case space required by the data structure for the sum is $O(logW.logB.(logW + logB)/\varepsilon)$ bits where B is an upper bound on the value of the sum, W is an upper bound on the width of the sliding window M, and ε is the desired upper bound on the relative error. The worst case time taken by the algorithm for processing a new element arriving is $O(logW.logB)$, and the time taken to answer a query for the sum is $O(logB + (logW)/\varepsilon)$.*

Let M be the number of sensors. At the fusion center, we use the well-known approach called False Discovery Rate to improve the performance of the decision fusion process.

6.3.2.3 Related Work

In fact, their problem is a change detection problem in which a change is a state transition from the in-control state to the out-of-control state or vice versa. There are many such transitions in the process control schemes. In other words, this change detection method is a continuous or non-restarting change detection method [Gandy 2013]. We note that restarting detection of changes means that a detector will be reset if a change is detected while non-restarting detector continuously detects the changes. Continuous change detection is also called non-restarting change detection.

6.3.3 Distributed Detection of Change by Clustering

In Chapter 5, we presented a reactive method for building and maintaining of clustering from multivariate streaming data based change detection. This section describes a continuous distributed algorithm for both detecting changes in multivariate streaming data and monitoring of clustering built from multivariate streaming data. In particular, we extend the clustering-based change detection method in Chapter 5.

6.3.3.1 Formal Model

Figure 6.3 depicts a framework for both change detection and clustering of streaming. Our algorithm includes two fundamental processes: remote-site process and coordinator-site process. Each remote site cannot directly communicate with other remote sites. Therefore it propagates the signal of change to all the remote sites through the coordinator. We assume that our distributed clustering model is used for the push-based queries in sensor network. Each remote-site process can perform two tasks: change detection and clustering. Likewise, the coordinator-site process consists of two modules: global clustering and global change detector. Our clustering-based change detection method presented in Chapter 5 can be considered

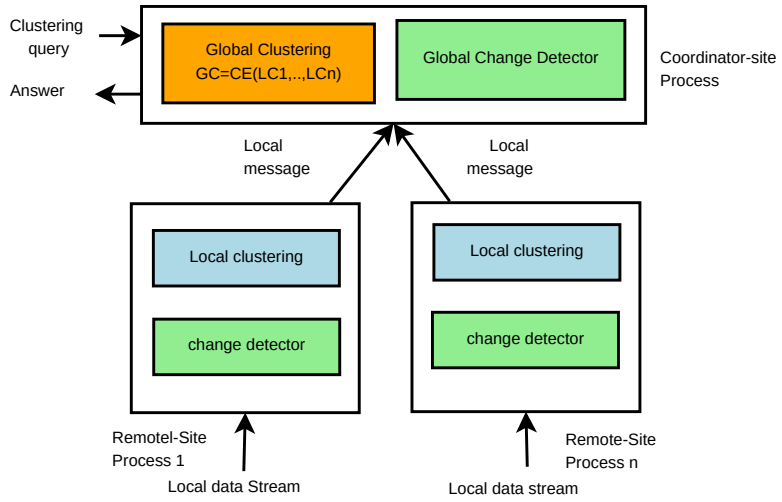


Figure 6.3: A continuous distributed framework for clustering streaming data and detecting changes in streaming data

a change detection method using model fitting approach in which a change occurs when a new block of data items does not fit the existing models. As clustering-based change detection method in Chapter 5 does not require the given threshold. Therefore, the advantage of our distributed framework is that it does not require the global threshold and local threshold.

6.3.3.2 Continuous Distributed Monitoring of Clustering

We present a reactive distributed monitoring of clustering constructed from data stream by using sliding window because the global clustering is rebuilt once a change is updated from some remote site. Reactive monitoring of model is called change-based monitoring or trigger-based monitoring of model. Our framework is a distributed continuous monitoring of clustering using trigger-based approach or change-based approach.

The goal of clustering monitoring is to continually track the global clustering. There are two types of local clustering and global clustering that need to be monitored. Continuous distributed monitoring of clustering from streaming requires the tradeoff among time-efficiency, space-efficiency, and communication-efficiency. The goal here is to minimize the communication cost yet to assure the clustering quality as well as the detection accuracy. Remote site updates its local clustering. The coordinator Then rebuilds its global clustering. Clustering sensor data stream is one of the model-driven data acquisition methods. The tradeoff of continuous distributed clustering include communication cost, clustering quality.

As the goal is to minimize the communication overhead, the size of global clustering and local clustering should be small. As size of the global clustering is

small, the communication cost for the exchange of global clustering between the coordinator and remote site is quite small. Micro-clusters is a kind of mergable synopsis.

It consists of two stages as below.

- Initialization: This stage builds the first global clustering at the coordinator site.
- Maintenance of clustering and detection of changes: This stage maintain global clustering, and distributed detection of changes in data streams that are generated from remote sites.

There are two ways to implement distributed detection of changes in streaming multivariate data

- by using local clustering-based change detection, or local clustering clustering over sliding window?
- by sending micro-clusters to the coordinator, and coordinator detects the changes

6.3.3.3 The Remote Process

A remote-site process makes a request for the global clustering to the coordinator site by transmitting a message to the coordinator site. The remote sites request the global view represented by global clustering . The remote site should reduce the workload of the coordinator processing. The remote sites should synchronize with the coordinator in order to Client builds and maintains local clustering based on its local data streams by using sliding window. Client transmits local clustering to the coordinator. Client continuously receives newly incoming data items, and updates its local clustering if a change occurs. If the client detects the changes of the incoming data stream. It simultaneously rebuilds new local clustering and send this new local clustering to the coordinator.

Algorithm 10 describes the local clustering process at the remote site.

6.3.3.4 The Coordinator Process

The coordinator receives the updates from remote site. The coordinator builds and maintains the global clustering based on the local clusterings it receives from the remote sites. The coordinator must update its global clustering based on local clusterings it receives from the remote sites. There are two kinds of requests from the remote sites: the request for generating the global clustering, and the the request for change detection. Methods for making clustering include $initClustering()$,

Algorithm 10: Local clustering process at the remote site for a push-based query

input : The data stream S_i generated at node R_i, parameters of sliding window: sliding step $slide$, window width win_size

output: The list of messages reporting the change point compared with the global clustering

1. make connection between the remote site and the coordinator;

2. initialization:

begin

- $t \longleftarrow 0$;
- $w1 \longleftarrow$ first b points from time t;
- send $w1$ to the coordinator;
- $gC \longleftarrow receiveGlobalClustering()$;

3. Maintenance and Detection of Changes

begin

- **while** *not at the end of the stream* **do**
  - $w2 \longleftarrow slide(w1, slide)$;
  - $blk \longleftarrow newItemBlock(w2)$;
  - **if** $change(blk, gC)$ **then**
    - report the change at the remote site i to other remote sites and the coordinator;
    - send blk to the coordinator;
    - $gC \longleftarrow receiveGlobalClustering()$;
    - $w1 \longleftarrow w2$;
  - **if** *a change in the incoming data stream at some remote site* **then**
    - $gC \longleftarrow receiveGlobalClustering()$;

$reClustering()$. There may be simultaneously many clustering requests from the different remote sites to the coordinator site. The coordinator-site process is multi-threaded.

There are two ways for checking the changes in streaming data as follows

- If a change occurs in the incoming data stream at some remote site, the remote site sends block of items to the coordinator. The coordinator rebuilds the global clustering and send global clustering to all the remote sites.
- When a new block arrives, the remote site checks if the changes occurs in the new block of items by using the old global clustering. This remote site propagates the signal of change to the coordinator site, and the other remote sites. Having received the signal of change from some remote site, the coordinator rebuilds the global clustering and sends to all the remote sites. Having received the signal of change from the remote site i, the rest of remote sites knows that the global clustering has been rebuilt, they receive the new global clustering for the next loop.

As can be seen in Algorithm 12, the global clustering process is a multi-threading process.

Algorithm 11: The coordinator process of the distributed detection framework for detecting changes in streaming data by clustering

input : A continuous query for monitoring the changes in the environment
output: The global clustering GC

```
while true do
    Accept the connection from the remote site R_i;
    if clientRequest is globalClustering then
        receive data from remote site and store data in buffer;
        gC ← Kmeans(k, buffer);
        send gC to all the remote sites;
    if clientRequest is changeDetection then
        receive block of data blk from client;
        detectChange(blk, gC);
        if change occurring then
            rebuild global clustering gC;
            send gC to all the remote site;
```

Since our distributed model for clustering is used for the push-based query,the coordinator process is a passive one which continuously listens for the clustering

Algorithm 12: Global clustering process at the coordinator site for a push-based query

input : A query for the global clustering
output: Continuously output the global clustering GC

```
while true do
    if a global clustering request received then
        Accept the connection from the remote site R_i;
        dataSummary_i ← receiveDataSummary(con_i);
        Add dataSummary_i to the buffer;
        gC ← Kmeans(k, buffer);
        send gC to all the remote sites;
    if there at least change signal received from remote sites then
        gC ← Kmeans(k, buffer);
        send gC to all the remote sites;
```

requests from the remote sites. The coordinator is capable of invoking the remote clustering method on each data stream, which is the data input at the corresponding remote site.

6.4 Evaluation

Due to the complexity of the model, the simulation approach was selected to evaluate the performance of the decentralized change detection framework using gossip protocol. In particular, the global estimation error of the event location using decision fusion model was evaluated. Furthermore, the time required to disseminate the decision from the information sources to the sinks as well as the communication cost in terms of transmitted messages were also taken into account. The distributed change detection scheme using gossiping was evaluated in two ways. For the gossip-based detection, we selected the simulation approach due to the complexity of the model. To illustrate the capability of deploying the framework, we implemented a simple framework for distributed detection of changes for parallel decision fusion in the client/server model in Java.

6.4.1 Simulation

For the integration of this thesis, we present the results of Jin et al. [Yang 2011, Tran 2011b] to illustrate the communication efficiency and estimated accuracy of the event location using gossip-based fusion scheme. We simulated a scenario of

wildfire systems in order to demonstrate the function of this distributed change detection framework. The goal of this simulation is to observe how fast the change can be detected and learned by other sensor nodes, how accurate the estimated location of the event is, and what is the communication overhead for information dissemination.

In general, we demonstrate that the framework is flexible in that one can choose different communication protocols to achieve different design aims (such as low communication cost, or good time performance, or fault tolerance), and also different decision fusion models on sensor nodes according to different applications. They simulated the scenario of a wildfire warning system in OmNet++ 4.0 and its INET framework that provides 802.11 MAC layer. By using this protocol stack library, we can compare the performance of different data communication protocols at the application layer. As we simulated the scenario at the application layer, the broadcast scheme was message broadcast instead of radio broadcast at the physical layer.

We simulated a wireless sensor network of 500 sensor nodes that detect changes. In the scenario of wildfire warning, the goal is to detect the abrupt changes of the temperature that may indicate a wildfire. This sensor network is organized as a matrix. The distance between nodes is 15 meters. The sensing range of each node is 25 meters. As such, the nodes in the central area have 8 neighbors, nodes at the edges or at the corners have 5 or 3 peer neighbors. We created two event locations and switched them on at 0.1 second after the simulation starts, to simulate the event of fire. The locations of the two fire events are randomly initialized. Global detection accuracy depends on local detection accuracy.

6.4.2 Analysis

To demonstrate the flexibility of this framework in different scenarios, we compared four different protocols for exchanging messages that encode local decisions of local change detectors at sensor nodes as shown in Figure 6.4 and **??**.

Among the four data transmission protocols, PushPullGossip is the fastest to reach the lowest estimation error, because it exchanges data using both push and pull approaches; Broadcast follows PushPull with high communication cost in terms of data message numbers as shown in Figure 6.6 , PushGossip and PullGossip spend twice as long as PushPullGossip for all sensor nodes to learn the event location.

The distribution of failed nodes conforms to the binomial distribution, and the failure occurs randomly in space.

The difference compared with the normal case where no nodes fail is that the final estimation error is obvious, around 7 meters, showing that some nodes which are around the event location and can directly detect the fire, fail. But in general,

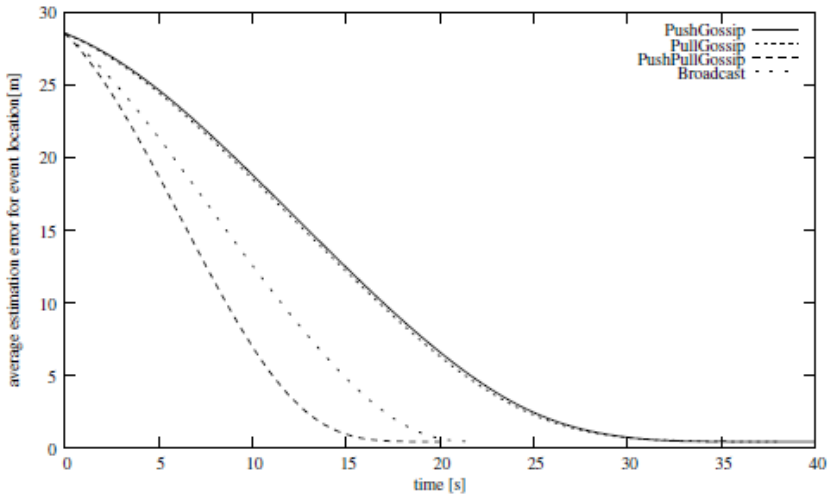


Figure 6.4: Location estimation error

the framework still functions similar to the normal situation in terms of time performance, meaning the events are detected and learned by all the nodes but with a relatively bigger range of where the fires occur.

6.4.2.1 Error in Location Estimation

Figure 6.4 and 6.5 show the estimation error for the event location. Failures of the nodes on the key path influence the time required for dissemination, and aggregation.

Figure 6.4 shows the change of average estimation error on all the nodes with the increasing time of simulation. At the beginning of the simulation, only the nodes nearby event directly detect the changes, while most of the other nodes belong to the state of N/A; with the information exchange protocol transmitting the detected information among sensor nodes and with the triangulation location algorithm executing on the improved data source, the event is learned by more and more nodes, which reduces the estimated location error. After 5 seconds of decision fusion runs, 30% sensor nodes using PushPullGossip learned the event and have knowledge on the event location with error smaller than 1 meter.

6.4.2.2 Communication Overhead

Figure 6.6 and Figure 6.7 show that the message number in the scenario where 25% nodes failed. In comparing communication cost, Figure 6.6 and Figure 6.7 show that PushGossip is the most efficient in both normal and random failure scenarios.

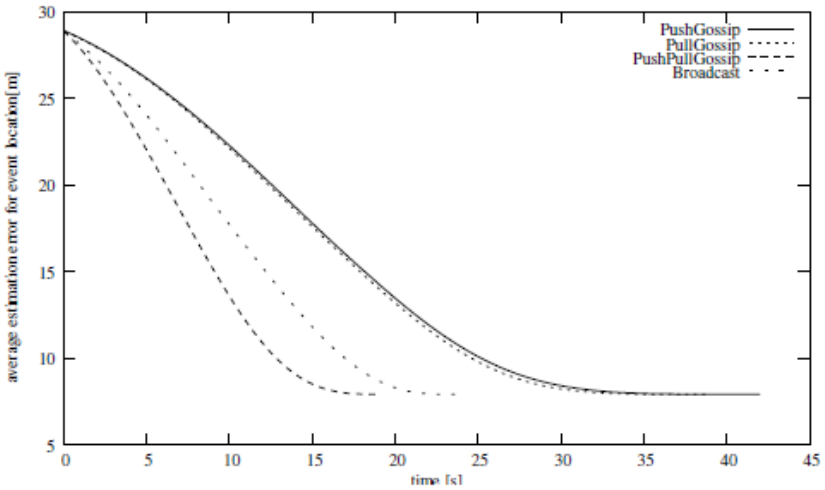


Figure 6.5: Location estimation error in 25% nodes failure scenario

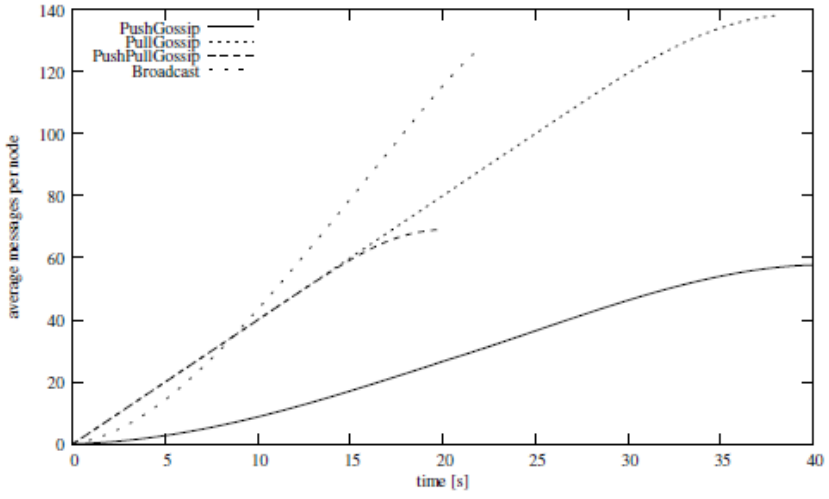


Figure 6.6: Number of massages transmitted

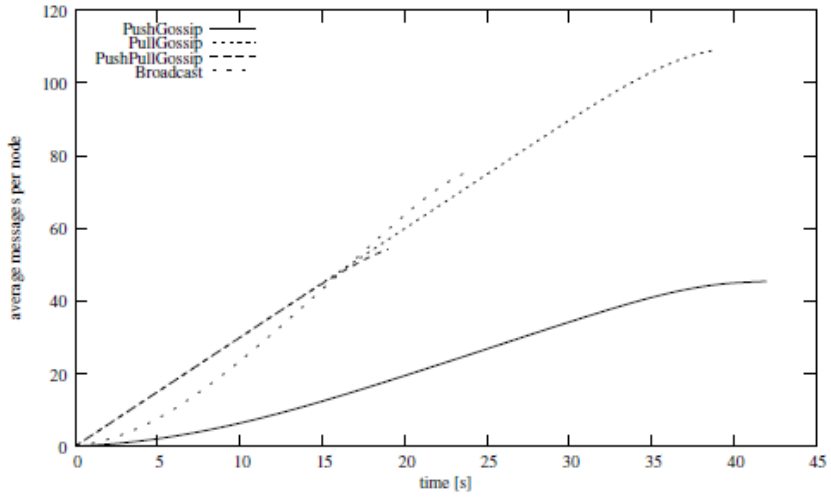


Figure 6.7: Number of massages transmitted 25% nodes failure scenario

In summary, the simulation results show that, the distributed data fusion framework can effectively fuse local event detection results to generate a more accurate event location. Besides, it demonstrates the ability to disseminate the information from the information source to the sink in a multi-source multi-sink scenario.

6.5 Related Work

Since sensor networks are distributed in nature, the problem of detecting changes in sensor data streams closely relates to the problem of decentralized detection. Over the last decades, the problem of decentralized detection has received much attention. There are two directions of research on decentralized detection. The first approach focuses on aggregating measurements from multiple sensors to test a single hypothesis. The second focuses on dealing with multiple dependent testing/estimation tasks from multiple sensors [Rajagopal 2008].

Distributed change detection usually involves a set of sensors that receive observations from the environment and then transmit those observations back to fusion center in order to reach the final consensus of detection. Decentralized detection and data fusion are therefore two closely related tasks that arise in the context of sensor networks [Niu 2006b, Niu 2006a]. Two traditional approaches to the decentralized change detection are data fusion and decision fusion. In data fusion, each node detects change and sends quantized version of its observation to a fusion center responsible for final detected decision making and further relaying information.

In contrast, in decision fusion, each node performs local change detection using some local change algorithm and updates its decision based on the received infor-

mation and broadcasts again its new decision. This process repeats until consensus among the nodes are reached. Compared to data fusion, decision fusion can reduce the communication cost because sensors need only to transmit the final local decisions represented by small data structures.

6.6 Summary

In this chapter, we have presented a distributed framework for detecting changes in streaming data that is created from multiple sources. In particular, we have proposed a framework for decentralized change detection in wireless sensor networks. This framework consists of two components: local change detector at the sensor node level and a global decision fusion model for determining the event location and the global detection decision at the fusion center.

For local change detection, we have exploited the sliding window model to detect local changes in sensor data streams in order to reduce the memory footprint of the change detector. The local change detectors are mentioned in chapter 3 and chapter 4 in detail.

The locations of events (i.e. changes in the environment) are estimated using a global decision model based on gossiping. By choosing different models of local change detection, global decision fusion, and data exchange protocols, different design goals can be achieved.

For the decentralized decision fusion framework, the decision fusion models efficiently improve the accuracy of event location estimation. Specifically, push-gossip can be implemented for power efficiency design aims, while push-pull-gossip is a good choice in reducing decision dissemination time, which is the key factor in emergency applications.

The distributed change detection scheme using gossip-based approach works with the assumption that the knowledge of detection performance of every local change detector is known to the fusion center, that means probability of detection and probability of false alarm are known to the fusion center as mentioned chapter 3 and chapter 4.

CHAPTER 7

Distributed Clustering of Streaming Data

If you want to truly understand something, try to change it.
(Kurt Lewin)

Contents

7.1 Introduction

In chapter 6, we present a distributed framework for detecting changes in streaming sensor data reflecting the environment. Besides as sensor nodes in wireless sensor networks are moving , wireless sensor network application should manage a huge amount of GPS streaming data. Therefore, the main challenge facing the problem of continuous monitoring of streaming sensor data is the restricted energy of sensor nodes [Klan 2011]. One of the solutions to this challenge is to reduce

the communication overhead by using the algorithms for clustering sensor nodes in senor network such as [Handy 2002, Younis 2004]. Clustering sensor nodes in sensor networks supports for power management and network routing. Another approach to clustering sensors in sensor networks based sensor data have been recently presented [Sagen 2010].

In many monitoring applications, one needs to track the changes of not only a single source but also the changes, or the trends of regions of sensor readings. For instance, in a coal mine surveillance application, when a gas leakage event occurs, the gas density readings measured at the sensor nodes near the source would follow a gradual increasing trend [Yin 2008]. Likewise, in the wildfire warning system, the readings measured at the sensor near the fire locations would follow a gradual increasing trend. Therefore, distributed clustering of streaming sensor data can support the distributed detection of changes in streaming sensor data.

Another example is a sensor network that continuously monitors the changes in the environment such as a building. In a naive approach, every sensor continuously sends its stream of observations (temperature, humidity, light) to the base station for processing in order to give a global view of the building [Karnstedt 2007]. However, the challenges facing this approach include: (1) the limited communication bandwidth; (2) the overlapping of observations sensed by nearby sensors that generates data redundancy.

Besides, clustering streaming data can be used to save energy in wireless sensor network by setting sensors not transmitting data into sleeping state [Sagen 2010]. This approach needs a global clustering from streaming data produced by sensor nodes. However, building and maintaining global clustering from multiple distributed data streams is challenging. In particular, building and maintaining global clustering in wireless sensor networks are even more challenging. To solve this challenge, we present a communication-efficient d exact method for clustering distributed streaming data as shown in Figure 7.1.

Mining distributed data streams has been increasingly received great attention [Sun 2006, Zaki 2002]. Data in nowadays data analysis applications is too big to gather and analyze in a single location or data is generated by the distributed sources. Design and development of data stream mining algorithms in high-performance and distributed environments thus should meet system scalability yet assure the quality of mining results. This paper studies the problem of clustering distributed streaming data.

Clustering is considered an unsupervised learning method which classifies the objects into groups of similar objects [Jain 1999]. Clustering data stream continuously produces and maintains the clustering structure from the data stream in which the data items continuously arrive in the ordered sequence [Guha 2003]. There are two types of problems of clustering data streams: (1) Clustering streaming data is to classify the data points that come from a single or multiple data streams into groups

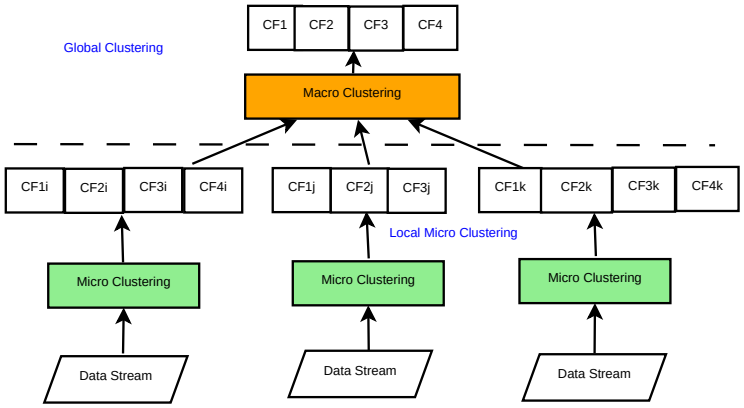


Figure 7.1: A distributed two-staged framework for clustering streaming data

of similar data points [Aggarwal 2003, Cao 2006, Kranen 2009]; (2) Clustering multiple data streams is to classify the data streams into groups of data streams of similar behavior or trend [Beringer 2006, Dai 2004]. The basic requirements for clustering data streams [Barbará 2002] includes a compact representation of clustering structures, fast, incremental processing of recently incoming data items, and clear and fast identification of "outliers". Clustering distributed streaming data is to classify the data points that come from multiple data streams generated by the distributed sources. Clustering distributed streaming data can be used to solve many real world applications. Such an example is a sensor network that continuously monitors the changes in the environment such as a building. In a naive approach, every sensor continuously sends its stream of observations (temperature, humidity, light) to the base station for processing in order to give a global view of the building [Karnstedt 2007]. However, the challenges facing this approach include: (1) the limited communication bandwidth; (2) the overlapping of observations sensed by nearby sensors that generates data redundancy. Clustering distributed streaming sensor data may be a solution to this problem. There are four approaches to the problem of clustering distributed data streams: (1) *Centralization:* All the raw data is transmitted to the coordinator site at which a clustering algorithm is applied to the received data. This approach is impossible in the distributed context because of the resource constraints. First, it is impractical to send all the data to the coordinator for processing and mining due to the limited network bandwidth. The communication constraint even becomes more critical when the energy consumption is considered in the sensor network with nodes of the power-limited batteries [Klan 2011]. Second, the transmission of all the raw data over a network can cause privacy and security concerns while the objective we want is only to understand and monitor the global view of the environment. (2) *Data summary:* First, each remote site sends the data representatives constructed from data stream by sampling method or synopsis construction method to the coordinator site. Second,

the coordinator site combines all the received data representatives into the global representative. Third, the clustering algorithm is performed on this global representative [Da Silva 2010, Yin 2008, Zhang 2008]. Compared to the centralized approach, this approach considerably reduces the volume of data to be transmitted. However, a major drawback of the summary-based clustering is that the clustering quality is affected due to the loss of data during the sampling or synopsis construction process. (3) *Local clustering:* each remote site clusters its local data stream. The local clustering is sent to the coordinator site [Cormode 2007]. The coordinator site generates the global clustering by using some cluster ensembles method. The important advantage of this approach is that it is fault tolerant. That means the framework works well despite of the malfunction of some nodes in the network.(4) *P2P stream clustering:* An important distinction and this approach with the first three approaches is that each remote site can directly communicate with each other to generate the global clustering. As each remote site must maintain the same global clustering [Bandyopadhyay 2006], this approach may incur high communication overhead.

The specific contributions of this chapter are as follows.

- We propose a distributed framework for clustering data streams. Every remote-site process generates and maintains micro clusters that represent cluster information summary, from its local data streams. Remote sites send the local micro-clusterings to the coordinator, or the coordinator invokes the remote methods in order to get the local micro-clusterings from the remote sites. Having received all the local micro-clusterings from the remote sites, the coordinator generates the global clustering by the macro-clustering method.

- We achieve the global clustering as exact as the underlying centralized algorithm. Since the local micro clusterings that are received at the coordinator from the remote sites are the same as the local micro clusterings produced by the micro-clustering method of the centralized clustering algorithm and the macro-clustering algorithms are similar in both the distributed clustering algorithm and the centralized clustering algorithm.

- Our framework achieves high scalability, and communication-efficiency by using the local micro-clustering approach.

In comparison with the previous work, our framework has the following advantages. First, as local clusterings are much smaller than the local raw data, transmission of local clusterings significantly reduces the communication cost. Second, the strongest point in our framework is that no approximation is required when producing global clustering.

We formulate the problem in Section 7.2. Section 7.3 describes the algorithms in the framework in detail. The experimental results, as well as evaluations of our framework are presented in Section 7.4. Related work is given in Section 7.5. Finally, we conclude and propose the future work in Section 7.6.

7.2 Problem Formulation

We consider a network of one coordinator site and N remote sites. Let $\{S_1, ..., S_N\}$ be the incoming data streams to the remote clustering modules at N remote sites. A data stream is an infinite sequence of elements $S_i = \{(X_1, T_1), .., (X_j, T_j), ...\}$. Each element is a pair (X_j, T_j) where X_j is a d-dimensional vector $X_j = (x_1, x_2, ..., x_d)$ arriving at the timestamp T_j. We assume that S_i^t is a block of M data items that arrives at the remote site i during each update epoch. We also assume that all the data streams arrive at the remote sites with the same data speed, that means at each update epoch t, every remote site receives the same M data items. We assume that each site knows its own clustering structure but nothing about the clustering structures at other sites. Let $LC_i^t = \{CF_i^1, .., .., CF_i^j, .., CF_i^{K_i}\}$ be a local micro-clustering generated by a clustering algorithm at remote site i at the update epoch t, where CF_i^j is the $j-th$ micro-cluster, K_i is the number of micro-clusters, for $i = 1, .., N$.

One of the fundamental properties of distributed computational systems is locality [Naor 1993]. A distributed algorithm for clustering streaming data should meet the locality. A local algorithm is defined as one whose resource consumption is independent of the system size. Local algorithms can fall into one of two categories [Datta 2006]: (1) exact local algorithms are defined as ones that produce the same results as a centralized algorithm; (2) approximate local algorithms are algorithms that produce approximations of the results that a centralized algorithm would produce. The objective here is to create and maintain the global clustering which is similar to the clustering created by the centralized stream clustering algorithm. In other words, our algorithm is an exact local algorithm.

A distributed streaming data clustering algorithm derived from a given underlying algorithm $\mathcal{A}$ is called $Dis\mathcal{A}$. The micro-clustering $Mic\mathcal{A}$ creates and maintains micro-clusters at the remote sites. The macro-clustering algorithm $Mac\mathcal{A}$ is used to produce meaningful macro-clustering at the coordinator.

7.3 Algorithm Description

Our distributed framework for clustering streaming data includes two fundamental processes: remote-site process and coordinator-site process. On the basis of type of query, we design the appropriate communication protocols as follows. First, if

the clustering query is push-based query, the remote sites send the local micro-clusterings to the coordinator. As the coordinator process is a passive process, it continuously listens for connection requests from the remote sites and receives the local micro clusterings that are sent by the remote sites. Push-based query is widely used for querying sensor network. Second, if the clustering query is pull-based query, the coordinator invokes the remote methods on the local data streams from the remote sites in order to get the local micro-clusterings. Such a pull-based query is used in the well-known protocol HTTP. In order for the coordinator to communicate with many remote sites concurrently, the global clustering process is organized as a multi-threading process.

Our algorithm uses the same data structure *Clustering* for both local clustering and global clustering. An object *Clustering* consists of many micro-clusters where each cluster is a cluster representative. Micro-cluster extends the cluster feature vector by adding the temporal components [Aggarwal 2003]. Cluster feature vector and micro-cluster are presented in detail in Section 5.2 of Chapter 5.

7.3.1 The Remote Process

This subsection describes how a local micro-clustering algorithm at the remote site works. As we assumed in Section 7.2, a remote site processes the incoming data from its local stream in an update epoch t: S_i^t. For $i = 1, .., N$, the local clustering process for the push-based type of query at the remote site i works as follows.

1. make the connection between the remote site i and the coordinator.
2. while not at the end of the data stream S_i
   (a) initialize the stream learner $Mic\mathcal{A}(S_i^t)$.
   (b) read block of data items at update epoch t
   (c) create the local micro-clustering LC_i^t by calling micro-clustering learner $Mic\mathcal{A}(S_i^t)$ on block S_i^t.
   (d) transmit the local micro-clustering to the coordinator.

The clustering algorithm for push-based query, at each update epoch, every remote site generates and transmits the local micro-clustering to the coordinator. For $i = 1, .., N$, the local clustering process for the pull-based type of query at the remote site i works as follows.

1. make the connection between the remote site i and the coordinator.
2. while not at the end of the data stream S_i
   (a) initialize the stream learner $Mic\mathcal{A}(S_i^t)$.

(b) read block of data items at update epoch e, let S_i^t denote the block of data items.

In contrast to the clustering algorithm for push-based type of query, in a clustering algorithm for pull-based query, the coordinator calls the local micro-clustering on the block of data items at epoch t from the remote-site process.

7.3.2 The Coordinator Process

This subsection describes how the coordinator process works. For each update epoch t, the coordinator works as follows.

1. for each remote site i, perform the following steps
   (a) make the connection between the remote site i and the coordinator.
   (b) receive the local micro-clustering LC_i^t from the remote site (for the push-based query), or remotely calls the local-micro clustering method from the remote site i on the block S_i^t (for the pull-based query).
   (c) add LC_i^t to a buffer buf.
2. if all the local micro-clusterings are received, the coordinator creates the global clustering GC^t by calling $Mac\mathcal{A}$ on the micro-clusters in the buffer.
3. process the global clustering GC_t.

We note that step 3 is executed on the basis of each specific context. For example, it may return the global clustering to the user as requested, or the global clustering can be stored in the history of global clusterings for some computation purpose, or it may be sent back to all the remote sites.

7.3.3 Algorithm Analysis

In this section we present some theoretical results including the global clustering quality, communication complexity, and computational complexity.

7.3.3.1 Global Clustering Quality

We shows that the global clustering produced by $Dis\mathcal{A}$ in an update epoch is similar to the clustering created by the underlying algorithm $Dis\mathcal{A}$ on the same data set. We first consider the case in which the clustering is generated by the centralized clustering algorithm $\mathcal{A}$. Let S_i^t be the streaming data block at the remote site i in an update epoch t, for $i = 1, .., N$, where N is the number of remote sites. We consider the $\mathcal{A}$ and $Dis\mathcal{A}$ in an update epoch. We also assume that all the data streams

arrive at the remote sites with the same data speed, that means at each update epoch t, every remote site receives the same data items. As such, the streaming data that the coordinator receive from all the remote sites in an update epoch is given by $S^{t+\triangle t} = \bigcup_{i=1}^{N} S_i^t$ where $\triangle t$ is the time for the coordinator to receive all the streaming blocks from the remote sites. Algorithm $\mathcal{A}$ works in two stages as follows

- *Micro clustering:* the output of the micro-clustering method is given by

$$LC_{centralized} = Mic\mathcal{A}\left(S^{t+\triangle t}\right) = \bigcup_{i=1}^{N} LC_i^t \tag{7.3.1}$$

- *Macro clustering:* the output of the macro-clustering method is given by

$$GC^t_{centralized} = Mac\mathcal{A}\left(\bigcup_{i=1}^{N} LC_i^t\right) \tag{7.3.2}$$

We now consider how our distributed stream clustering framework $Dis\mathcal{A}$ works. However, a distinction between the centralized algorithm $\mathcal{A}$ and the distributed algorithm $Dis\mathcal{A}$ is that while the both micro-clustering method and macro-clustering method of $\mathcal{A}$ are executed in the same process, the micro-clustering method of $Dis\mathcal{A}$ takes place at the remote sites, the macro-clustering method of $Dis\mathcal{A}$ occurs at the coordinator. The framework $Dis\mathcal{A}$ works as follows

- *At the remote site:* Each remote site generates the local macro-clustering LC_i, for $i = 1, .., N$. If the clustering query is pushed-based query, remote sites send its local micro-clustering to the coordinator.

- *At the coordinator:* If the clustering query is pull-based query, the coordinator invokes remote clustering methods in order to get all the local micro-clusterings. The list of local clusterings received from remote sites

$$LC^{t+\triangle t_1}_{distributed} = \bigcup_{i=1}^{N} LC_i^t \tag{7.3.3}$$

  where $\triangle t_1$ is the time needed to transmit all the local clusterings to the coordinator

  The global clustering $GC^t_{distributed}$ is created by the macro clustering method Mac up to time $t + \triangle t_1 + \triangle t_2$ is given by

$$GC^t_{distributed} = Mac\mathcal{A}\left(LC_{distributed}\right) = Mac\mathcal{A}\left(\bigcup_{i=1}^{N} LC_i^t\right) \tag{7.3.4}$$

  where $\triangle t_2$ is the time needed to produce global macro-clustering (global clustering).

From the equations 7.3.1, 7.3.2 and 7.3.3, 7.3.4, we can conclude that the global macro-clustering produced by the distributed framework $Dis\mathcal{A}$ in an update epoch t is similar to the macro-clustering produced by the centralized version of the two-phased stream clustering algorithm $\mathcal{A}$.

7.3.3.2 Communication Complexity

One of the fundamental issues of a distributed computing algorithm is to evaluate the communication complexity. The communication complexity of a distributed framework for clustering is the minimum communication cost so that it produces the global clustering. Let b_i^t be the total bits sent between remote site i and the coordinator. The size of a local micro-clustering $|LC_i|$ is defined as the product of the size of each micro-cluster $|CF|$ by the number of clusters K_i in the local clustering structure $|LC_i| = |CF|K_i$. The number of bits used to transmit a local micro-clustering produced by the remote site i to the coordinator site is $\log_2 |LC_i^t|)$. As such, the communication cost of all local clusterings from the remote sites to the coordinator is given by.

$$\sum_{i=1}^{N} \log_2 |CF|K_i = \sum_{i=0}^{N} \log_2 K_i + \frac{(N+1)N}{2} \log_2 |CF| \tag{7.3.5}$$

Let DisClustream be the distributed version of Clustream. If the DisClustream is used, and the number of clusters at all the remote sites and the coordinator are the same, the communication cost needed to answer a clustering query at an update epoch t is given by

$$N \log_2 K + \frac{(N+1)N}{2} \log_2 |CF| \tag{7.3.6}$$

where K is the number of micro-clusters in a local micro-clustering, and $|CF|$ is the size of a micro-cluster.

7.3.3.3 Computational Complexity

This section answers the question how much time is needed to compute the global clustering. Let $\mathcal{T}_{mic}$ and $\mathcal{T}_{mac}$ denote the time needed to produce a micro-clustering and a macro-clustering respectively. The time needed to generate the clustering by an underlying two-phased stream clustering $\mathcal{T}_{centralized}$ is given by

$$\mathcal{T}_{centralized} = N\mathcal{T}_{mic} + \mathcal{T}_{mac} \tag{7.3.7}$$

The time needed to generate the global clustering is computed from the time at which all the local micro-clustering algorithms start until the time at which the global macro-clustering is generated. Therefore, the time needed to compute the global clustering $\mathcal{T}_{distributed}$ is given by

$$\mathcal{T}_{distributed} = \mathcal{T}_{mic} + \mathcal{T}_{transmit}^{allLC} + \mathcal{T}_{mac} \tag{7.3.8}$$

Let $\mathcal{T}_{transmit}^{LC_i}$ denote the time needed for transmitting the local clustering from the remote site i to the coordinator. As our framework is multi-threaded, the total time needed to transmit all the local clusterings $\mathcal{T}_{transmit}^{allLC}$ is less than or equal

to the sum of time needed to transmit each local clustering $\mathcal{T}_{transmit}^{LC_i}$, that means $\mathcal{T}_{transmit}^{all\,LC} \leq \sum_{i=1}^{N} \mathcal{T}_{transmit}^{LC_i}$. From the equation 7.3.8, we can reduce the time to produce the global clustering at the coordinator by the following ways: (1) speed up the local clustering algorithm (reduce $\mathcal{T}_{clust}^{LC}$); (2) reduce the time to send the local clustering $\mathcal{T}_{send}^{LC}$; (3) speed up the cluster ensemble algorithm at the coordinator site. The first task depends on the selection of stream clustering algorithm at remote sites. For the second task we reduce the size of local clusters transmitted. The third task depends on the selection of the ensemble cluster approach. The time to send a local clustering to the coordinator site depends on the network bandwidth, and the size of local clustering. As shown in the experiment part, the local micro-cluster is small. The speedup of our framework is given by

$$speedup = \frac{\mathcal{T}_{centralized}}{\mathcal{T}_{distributed}} \tag{7.3.9}$$

where $\mathcal{T}^{centralized}$ is the time needed to generate the clustering by the underlying two-phased stream clustering algorithm and $\mathcal{T}_{distributed}$ is the time needed to generate the global clustering by our framework.

From the equations 7.3.7, 7.3.8, and 7.3.9, we have

$$speedup = \frac{N\mathcal{T}_{mic} + \mathcal{T}_{mac}}{\mathcal{T}_{mic} + \mathcal{T}_{transmit}^{allLC} + \mathcal{T}_{mac}} \tag{7.3.10}$$

Based on the experiment in Subsection 7.4.3.2, the average values of $\mathcal{T}_{mic}$, $\mathcal{T}_{mac}$, and $\mathcal{T}_{transmit}^{LC_i}$ were computed. From these parameters, the function speedup was approximately given by $speedup = 0.98N$. In conclusion, the speed of our framework scales up with the increasing number of remote sites.

7.4 Empirical Results

All the experiments were performed on a Intel Pentium (R) $2.00GHz$ computer with $1.00GB$ memory, running Windows XP professional. We implemented an instance of the proposed framework, which we call DisClustream (Distributed Clustream). DisClustream was developed on the basis of the underlying stream clustering algorithm Clustream [Bifet 2010b]. The streaming data sets (Sensor Stream, Powersupply Stream, and Kddcup99) that were used in our experiments were from the Stream Data Mining Repository [Zhu 2010]. Depending on the purpose of each group of experiments, we selected some data sets of the above data sets.

We sought to answer two empirical questions about our framework: (1) How accurate is our clustering framework for distributed data streams in comparison with the central clustering approach on the same data set? (2) How scalable is our framework? All the empirical evaluations were done in an update epoch.

7.4.1 Evaluation on Global Clustering

This group of experiments evaluated aspects of the global clustering in terms of clustering quality compared with the centralized clustering algorithm, time needed to create the global clustering. The goal of these experiments is to compare the clusterings produced by centralized and distributed stream clustering algorithms. We expect that the global clustering produced by distributed stream clustering will be almost the same as the clustering produced by centralized stream clustering algorithm. For ease of comparison, we chose a data set in which the number of attributes and the number of classes are small sufficient to visualize clusterings. The appropriate data set is for this task is Powersupply used to predict which hour the current supply belongs to. This data set includes 29928 records, each record consists of 2 attributes. These records can fall into one of 24 classes. To obtain clusterings, we ran the centralized version of ClusStream, and the distributed version DisClustream. For the centralized CluStream, the entire data set can be seen as the incoming data stream. For the distributed DisCluStream, we divided the data set into two data sets of the same size. There were two remote sites, each remote site produced a local micro clustering from one of two above data sets.

Figure 7.2 illustrates two clusterings produced by centralized (marked by square) and distributed (marked by circle) stream clustering algorithms. Figure 7.2 illustrates two clusterings produced by centralized (marked by square) and distributed (marked by circle) stream clustering algorithms.

As we expect, the clustering produced by distributed clustering algorithm DisClustream was almost the same as the clustering produced by centralized one CluStream (Figure 7.2). In conclusion, the global clustering produced by our distributed clustering framework is almost similar to the clustering produced by centralized one. The quality of an underlying local stream clustering affects the overall quality of the global clustering.

We should distinguish between the time needed to produce the global clustering and the time needed to run the macro-clustering algorithm. The time needed to generate the global clustering in the distributed framework is the duration from the time at which all the local micro-clustering algorithms start until the time at which the global macro-clustering is generated. The time needed to generate micro clusters at the remote site is much greater than the time needed to run macro clustering algorithm in order to generate the global clustering at the coordinator. This is a direct consequence of the two-phased stream clustering approach in which the online phase consumes more time than the off-line one [Aggarwal 2003]. Our the experiment with KDDCup99 data set shows that time to produce micro clusters at the remote site (14021 instances, and number of micro-clusters 100, time needed is 25701 ms) is much greater than the time needed to produce macro-clustering (391 ms) at the coordinator. Therefore, in order to speed up the algorithm, we should

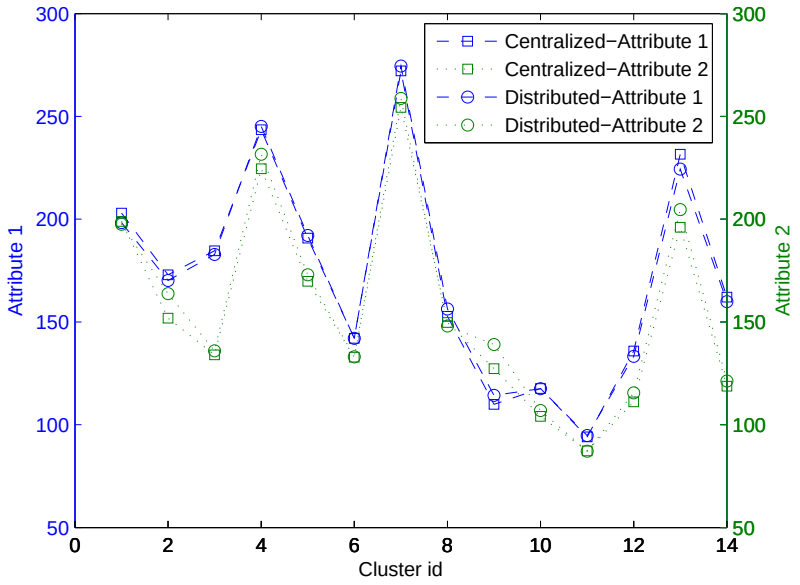


Figure 7.2: A comparison of two clusterings that were created by distributed stream clustering algorithm and the centralized one on Power Supply data set

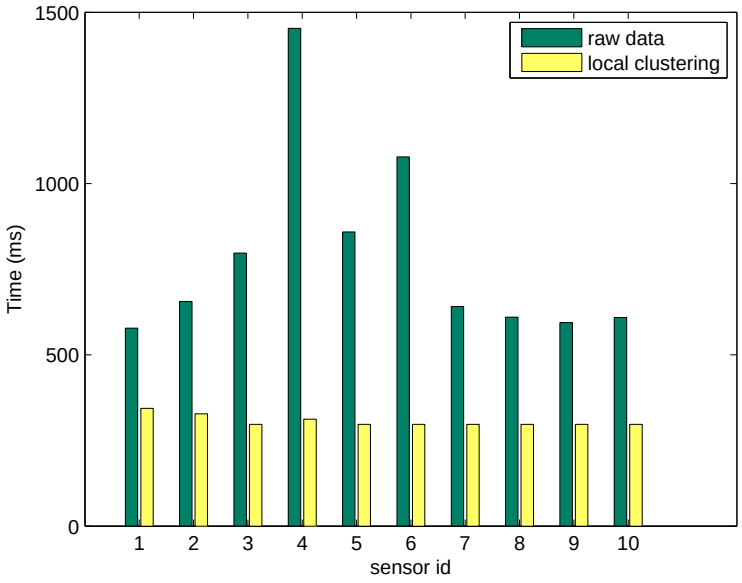


Figure 7.3: Time to transmit block of streaming data vs. time to transmit local-micro clustering

select, or develop the fast and high-quality micro-clustering approach.

7.4.2 Evaluation on Communication Efficiency

The goal of this subsection is to evaluate the time needed to transmit block of streaming data with the time needed to transmit the local micro-clustering built from the corresponding the block of streaming data. As shown in Section 7.3.3.2, the communication overhead needed to transmit local micro clusterings would be negligible compared to the raw data transmission involved. In other word, the time needed to transmit local micro-clustering would be much smaller than the time needed to transmit the raw streaming data.

We execute this groups of experiments with DisClustream on the Intel data set. We divided this data set into many data sets based on the sensor id. Without loss of generality, the sensor data set from sensor 1 to 10 was used as data streams at the remote sites. The number of micro-clusters was set to default number (100 kernels, in Clustream). For simplicity, we set up 10 experiments with 10 sensor data sets from 1 to 10 separately, that means each experiment include one coordinator and one remote site. Figure 7.3 shows that, the time needed to transmit raw streaming data depends on the size of streaming data block. The time needed to transmit micro-clusters is almost invariant as the number of micro-clusters were set to the same number 100 in all experiments. Our experiments shows that the micro-clusters produced by remote sites are small sufficient for meeting the requirement of communication efficiency.

The goal of this experiment is to determine the size of local clustering. The DisCluStream was used in this experiment. We used KDD Cup99 for this experiment. The framework consists of one remote site and one coordinator. At the remote site, the number of micro-clusters was fixed to 100. The coordinator received the local clustering and wrote it to file. We changed the number of instances in the window (an update epoch) in the range 1000, 2000, 3000, 4000, 5000. The size of local clustering file was invariant and equals to 13.2KB. In fact, the actual size of local clustering in memory may be smaller than 13.2 KB as the local clustering file includes the size of file format. The size of local clustering is invariant because we fixed the number of micro-clusters to 100. Therefore, the size of local clustering in DisClustream is independent of the number of instances that a remote site receives in an update epoch.

7.4.3 Effect of Block Width and Number of Micro-clusters

The success of a distributed framework for clustering streaming data depends on the scalability of the local stream clustering algorithm and the scalability in term of number of remote sites. The scalability of the micro-clustering algorithm used

at the remote sites is mandatory because it must process the large volume of incoming data. The scalability of the micro-clustering algorithm Clustream in the number of data dimensions, and the number of clusters was thoroughly evaluated [Aggarwal 2003] in term of the time needed to produce clustering. In contrast, we evaluated the scalability of DisClustream in terms of communication time by increasing the number of micro-clusters, and the number of instances in the update block.

7.4.3.1 Effect of Block Width

This group of experiments studied the scalability in the increasing number of instance in the block. The underlying stream clustering algorithm was used to evaluate the scalability of our proposed framework is Clustream. As such, the micro-clustering algorithm at the remote sites the micro-clustering method of Clustream while the macro-clustering algorithm at the coordinator is K-means. Figure 7.4 shows how our distributed stream clustering framework scale with the number of instances in an update epoch. As we experimented with the sensor data set in which each record consisted of two attributes, we can observe the time to transmit the local clustering to the coordinator in Figure 7.4 . We also tested the scalability in window size with KDD Cup data set. The result of this experiment demonstrated that, the time needed to create local clustering scales with the number of instances in an update epoch. However, the time needed to transmit the local clustering is hardly varied. For example, with KDD Cup data set, the time needed to create the local clustering (18,11 seconds) was much larger than the time needed to transmit the local clustering (46 miliseconds) with the number of instances in an update epoch 10000. As the number of instances in an update epoch increases 10 times (100000), the time needed to create local clustering was almost 3 (minutes) while the time needed to transmit local clustering was only 47 (milliseconds).

7.4.3.2 Effect of Number of Micro-clusters

To evaluate the scalability of our algorithm in the increasing number of micro clusters. Evaluation on the scalability of our algorithm was done on two data sets Intel sensor data set and KDDCup 99 data set. We set up experiment as follows. The number of instances was fixed to 10000 instances. There were nine remote sites connected to the coordinator sites. The number of micro clusters was selected from the range $100, 150, 250, 350, 400, 450, 500$ as shown in Figure 7.5. An increase in the number of micro-clusters resulted in the increasing time to transmit micro-clusters In our experiments, time required to produce micro-clusters is much larger than the time required to transmit local micro-clusters.

Our experiments on the scalability with DisClustream showed that the time

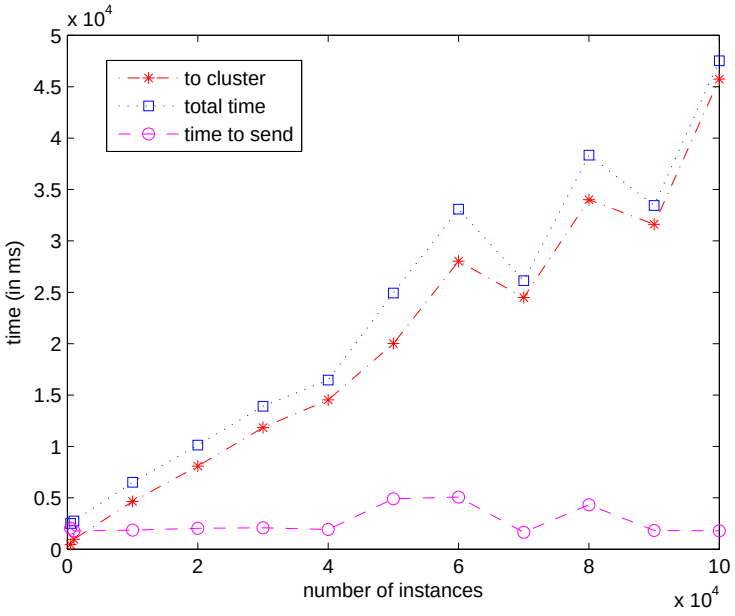


Figure 7.4: Effect of block width

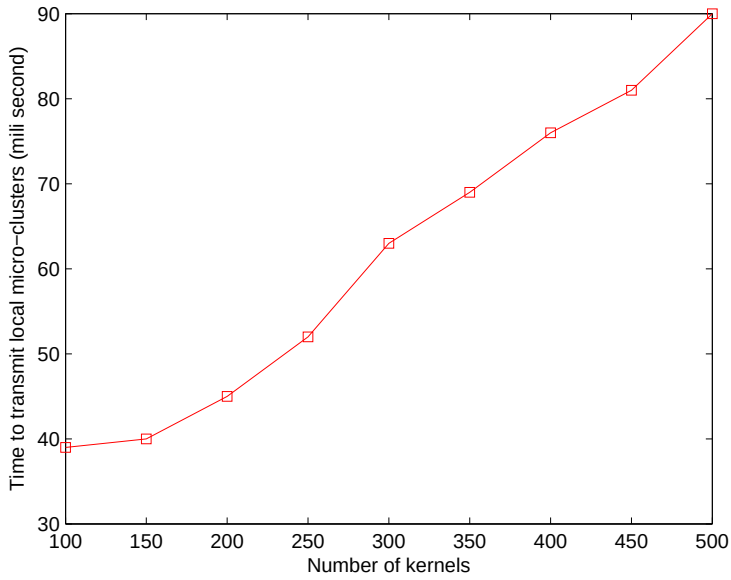


Figure 7.5: Effect of number of micro-clusters

needed to create local clustering in an update epoch scales with the increasing number of instances in the window, and the increasing number of micro-clusters while the time needed to transmit a local clustering is almost invariant. The data stream clustering algorithms that are fast sufficient to deploy in resource-limited applications such as sensor networks are still open questions.

7.5 Related Work

The first two-phased stream clustering algorithm Clustream introduced by Aggarwal et al [Aggarwal 2003] has two separate parts: online and off-line components. The role of the online component is to process and to extract summary information from data stream while the off-line builds the meaningful clustering structure from the extracted summary information.

Clustream can efficiently maintain a very large number of micro-clusters, which are then used to build macro-clusters. Further, clustering structure can be observed in arbitrary time horizons by using pyramidal time model. Hence, Clustream is capable of monitoring the evolution of clustering structure. However, one of the main drawbacks of Clustream is that it requires the number of clusters in advance. To overcome this drawback, Cao et al. [Cao 2006] have proposed a density-based clustering data streams. This advantage of this approach is that it can handle outliers in data streams, and adapt to the changes of clusters. However, the major limitation of DenStream is that it frequently runs the off-line component in order to detect the changes of clusters. As such, the off-line component consumes the cost of computing clusters much. In the experimental evaluation, we use two well-known algorithms Clustream [Aggarwal 2003], and DenStream [Cao 2006] as the local stream clustering algorithms at the remote sites.

Cormode et al. [Cormode 2007] presented a suite of k-center-based algorithms for clustering distributed data streams. The communication topology consists of N remote sites, and one coordinator. Any two remote sites can directly communicate with each other. Their approach can provide a global clustering that is as good as the centralized clusterings. However, this approach provide a approximate clustering by using the underlying k-centers stream clustering algorithm. Zhou et al. [Zhou 2007] considered the problem of clustering distributed data stream by using EM-based approach. The advantage of this approach is that it can deal with the noisy and incomplete data streams. To deal with the evolving data streams, they used the reactive approach to rebuild the local clustering and global clustering when it no longer suits the data. However, it is not scalable in the large number of remote sites such as in a sensor network of thousands of nodes. Although this drawback can be overcome by model-merging technique, the global clustering quality considerably reduces. Furthermore, the memory consumption at the remote site increases

with the increasing number of data dimensionality and the increasing number of models. In contrast to the work introduced by Zhou et al. [Zhou 2007], Zhang et al.[Zhang 2008] introduced a suite of k-medians-based algorithms for clustering distributed data streams, which work on the more general topologies: topology-oblivious algorithm, height-aware algorithm, and path-aware algorithm. To deal with the evolving data streams, they selected periodic approach in which the global clustering at the root site is continuously updated every period called update epoch. Similar to [Cormode 2007], the global clustering is approximately computed on the summaries that are received from the internal and leaf nodes.

7.6 Summary

We have proposed a distributed framework for clustering streaming data by extending the two-phased stream clustering approach that is widely used to cluster a single data stream. While the remote sites create and maintain micro-clusters by using the micro-clustering method, the coordinator creates and maintains the global clustering by using the macro-clustering algorithm. We have theoretically analyzed our framework in the following aspects. The global clustering created in an update epoch is similar to the clustering created by the centralized stream clustering algorithm. We also estimated the communication cost as well as proved that the speed of our framework scales with the number of remote sites. Our empirical results demonstrated that the global clustering generated by our distributed framework was almost the same as the clustering generated by the underlying centralized algorithm on the same data set with low communication cost. In conclusion, by using the local micro-clustering approach, our framework achieves scalability, and communication-efficiency while assuring the global clustering quality. Our work along with other work can be seen as one of the first steps towards a novel distributed framework for mining streaming data by using micro-clustering method as an efficient and exact method of data summarization.

CHAPTER 8

Conclusion

Our only security is our ability to change. (John Lilly)

Contents

This dissertation has addressed the problem of change detection in streaming data by studying a wildfire warning system using wireless sensor network as a motivating scenario. This scenario poses the challenges for the problem of change detection in streaming data as follows. These challenges include those coming from the problems of change detection, those arising from the data stream model, those originating from the distributed nature of sensor streaming data, and those resulting from the restricted resources of wireless sensor network such as limited range of sensing, small memory, low power of computing, and limited energy of battery. From the motivating scenario and its challenges, the solution to the problem of change detection in streaming has been formulated. Change detection in wireless sensor network has been thoroughly studied for a long time. However, much effort has been only focused on one-shot change detection, that means detecting and reacting to a change once it occurs. Only recently, the continuous distributed monitoring model has been received attention from research community [Cormode 2013]. Along with this line of research, this dissertation presents a distributed framework for detecting changes in streaming data by using the local approach. Sec 8.1 summarizes the key contributions of this dissertation. The possibilities of extending this work in promising directions are presented in Section 8.2.

8.1 Key Contributions

1. As data stream is infinite in nature and the systems resources are limited, this dissertation proposes the methods for detecting changes by quantifying the difference between the reference window and the current one. This

approach is based on the work proposed by Kifet et al. [Kifer 2004]. An important distinction between our approach and theirs lies in the choice of dissimilarity measures quantifying the change. While Kifet et al. [Kifer 2004] use Kolmogorov-Smirnov distance with high complexity of computation, we used the geometric distances such as the Euclidean distance, and Manhattan distance for quantifying the change. Besides, by changing the relative positions of the reference and sliding windows, the change detection algorithms for different purposes are developed such as the problem of change point detection using overlapping windows model, or interval-based change detection using adjacent windows model. The detection performance of a change detection method using sliding windows model depends on the factors such as the window width, the distance measure, and the detection threshold. While the previous work on change detection only evaluates the detection accuracy in a snap-shot of a data stream, it is the first time we can evaluate the detection performance of our change detectors at anytime or the entire data stream (assume that the data stream is finite). We compare detection accuracy of two change detectors in both ROC-space and PR-space. Two well-known models for processing data stream include time-based model (in Oracle CEP model), and tuple-based model (in StreamBase) [Jain 2008]. Our methods for detecting changes are only for tuple-based model. Another limitation is that our change detection methods use the fixed size window that makes the choice of window width difficult.

2. Raw streaming data is defined as data that has not been transformed by any transformation. In Chapter 3, we uses two windows to truncate two segments of a data stream. The original streaming data in these two segments preserves. Therefore, change detection methods presented in Chapter 3 are change detection method in raw streaming data. As the sliding window size can be too large to store in memory, a general synopsis-based change detection framework is proposed. Besides, data comes from streaming data sources incrementally, thus incremental change detection is necessary. Discrete Fourier Transform converts a data stream in time domain into frequency domain and vice versa. Discrete Fourier Transform can be used to reduce the data dimension. Besides there exists many fast Fourier transform algorithms that can meet the requirement for fast computing in streaming data. Thus, the DFT-based change detector is designed and evaluated as an example of synopsis-based change detector. As the data stream must be processed under the resource constraints with high speed, an incremental method of computing DFT coefficients was exploited to further speed up the DFT-based change detector.

3. A question that arises is how the detection accuracy of synopsis-based

change detection methods is compared with the non-synopsis change detection methods. Theoretical and empirical analysis shows that the detection performance of synopsis-based detector is similar to that of non-synopsis change detector if a distance function quantifying the changes is preserved under the synopsis construction process. In addition to DFT transformation, there are the similar techniques such Haar Wavelet transformation [Chan 1999], Johnson-Lindenstrauss transformation [Achlioptas 2003].

4. As automated systems require the capability of realtime processing, and adapting to the changing environments. For instance, sensor networks need the automated change detection methods in which detection threshold must adapt to these changes of the environment. Besides, the change detection methods presented in Chapter 3 and Chapter 4 can be only applied for univariate data stream, or single dimensional data stream. Hence, we have proposed an automated change detection algorithm in streaming multivariate data by clustering. This change detection algorithm can automatically detect the changes in streaming multivariate data without requiring the detection thresholds. We present a method for building and maintaining clustering over sliding window by using reactive approach in which the clustering is rebuilt once a change in the data distribution is detected. The drawback of the change detection methods presented in Chapters 3, 4, and 5 is that the window width is fixed. The selection of window width is challenging and application-specific dependent.

5. In real-world applications, streaming data may come from multiple sources such as continuous data streams from sensor networks. A framework for decentralized change detection in wireless sensor networks is proposed, and an instance of this model is presented and simulated by using Gossip scheme. Chapter 6 proposes a distributed framework for detecting changes in streaming data that is created from multiple sources. The distributed detection framework consists of three components: local change detectors, decision fusion schemes, and detection network of network protocol and communication protocol. In particular, we have proposed a framework for decentralized change detection in wireless sensor networks. The locations of events are estimated using a global decision model based on gossiping. By choosing different models of local change detection, global decision fusion, and data exchange protocols, different design goals can be achieved. For the decentralized decision fusion framework, the decision fusion models efficiently improve the accuracy of event location estimation. Specifically, push-gossip can be implemented for power efficiency design aims, while push-pull-gossip is a good choice in reducing decision dissemination time, which is the key factor in emergency applications. The distributed change detection scheme

using gossip-based approach works with the assumption that the knowledge of detection performance of every local change detector is known to the fusion center, that means probability of detection and probability of false alarm are known to the fusion center.

6. A distributed framework for clustering streaming data is proposed by extending the two-phased stream clustering approach which is widely used to cluster a single data stream. While the remote sites create and maintain micro-clusters by using the local micro-clustering method, the coordinator creates and maintains the global clustering by using the macro-clustering algorithm. The theoretical analysis and empirical experiments show that, the global clustering which is created in an update epoch is similar to the clustering which is created by the centralized stream clustering algorithm. We also estimated the communication cost as well as showed that the speed of our framework scales with the number of remote sites. By using the local micro-clustering approach, our framework achieves scalability, and communication-efficiency while assuring the global clustering quality. In chapter 5, we describes how to detect changes in streaming multivariate data, hence in Chapter 7, we present a distributed clustering method for streaming data, and then extend it for the distributed detection of changes in streaming multivariate data. In particular, remote sites will summarize streaming multivariate data by using online phase, and send it to the coordinator, coordinator will detect the changes by using the clustering-based change detection presented in Chapter 5.

8.2 Future Directions

Our results presented in this dissertation suggest the future directions of research as follows.

- Although change detection methods proposed in this thesis meet the certain requirements such the ability to work in the environment with restricted resources, they need to further improve the processing performance in order to keep up with the high speed of the modern data streams. A widely used approach to dealing with the high speed of data stream is sampling techniques [Vitter 1985]. As the goal of change detection is only to detect, localize, and report the change, the approximate summary built from data stream is acceptable for change detection. Recent work on change detection by sampling have been proposed by Cho and Ntoulas [Cho 2002], Li et al. [Li 2007]. However, the approach proposed by Li et al. [Li 2007] quantify the changes by the statistical distance that is quite complex A distance

in [Kifer 2004]. Therefore, fast detection of changes in streaming data by sampling use the geometric distances L_p^p is necessary. In particular, Cohen and Kaplan [Cohen 2012] have recently proposed the novel approaches to estimating the changes from the samples for the Manhattan and Euclidean distances instead of computing these distances on original data.

- Change detection methods using two overlapping windows can be incorporated with the incremental query processing to improve the performance of incremental query processing proposed in [Liarou 2012, Liarou 2013].

- In many real-world applications, streaming data is sparse, thus it is important to develop change detection methods for sparse streaming data. Based on the results of Chapter 3 and Chapter 4, and the recent advances in compressive sensing [Hassanieh 2012a, Hassanieh 2012b, Davenport 2006], we can develop change detection methods for sparse streaming data by using compressive sensing. The change detection problem is in fact a binary classification problem. One of the challenges facing a classification problem is the unbalanced class problem, that means the class distribution is unbalanced. For example, for the rare change detection problems, the number of change points is much smaller than the number of non-change points. An open research direction should be devoted to the problem of rare change detection using compressive sensing.

- Micro-clustering approach is used in many tasks of data mining. For instance, micro-clustering method is used to summarize cluster information for the data stream classification [Masud 2008]. In chapters 5, 6.3.3, and 7, we used micro-clusters as synopses that are used for clustering, change detection. In chapter 7, our theoretical and empirical results show that we can develop a general framework for mining distributed streaming data by using micro-clusters. The remote sites create and maintain micro-clusters or the variants of micro-clusters. The coordinator can generate and maintain the global pattern from the micro-clusters that are received from the remote sites by using the micro-clusters based data mining such as classification, clustering, and frequent pattern mining. The reasonable foundation behind this local micro-clustering approach to data stream mining is that micro-clusters maintain statistics at a sufficiently high level of granularity (temporal and spatial)[Aggarwal 2003]. As such, micro-clustering algorithms can be seen as a data summarization method. For the problem of clustering distributed streaming data, there are some open questions such as: How to create and maintain the global clustering by cluster ensembles method on the micro-clusters that are received from the remote sites? How to build and maintain the global clustering in sliding window in a distributed environment by using

the local micro-clustering algorithm?

- We can extend the distributed framework which were presented in Chapter 6 for other network topologies. The main challenge facing the problem of distributed detection of changes in streaming data presented in Chapter 6is that the detection performance of local change detections are impossible to compute in the real-world application because of the lack of ground truth data. Therefore, a framework that can control the false discovery rate in order to achieve the better performance of the fusion and decision making at the fusion center can be developed by extending the recent work of Ray and Varney [Ray 2011] for the streaming model.

- The motivating scenario for wildfire warning system was used to illustrate the need for the change detection methods presented in this thesis. However, these methods can be also used for the other applications such as medicare monitoring system [Sun 2010, Huang 2013]. In particular, DFT-based change detection is very useful for physiological streaming data in medicare monitoring system such [Huang 2013] since physiological data is periodic.

- Similar to clustering, and classification of streaming data, the evaluation of detection accuracy, and performance of the change detectors is very important. Although many change detection methods have been proposed and developed, a standard benchmark for change detectors in streaming data is still open issue. Benchmarking for change detection in streaming data faces the following challenges. The first challenge here is that in order to evaluate detection accuracy of detectors, we need to know how to generate groundtruth data that allows us to determine the true change point, true unchange points. The second challenge is the metrics for evaluating detection accuracy. Recently, some work on performance of change detectors for streaming data have been proposed in [Dasu 2011, Tran 2011b]. The third challenge is the evaluation of detection accuracy should be performed continuously due to the theoretically infinite nature of data streams. In this thesis, to our best knowledge, we have continuously evaluated the detection accuracy despite the length of data stream. We also made a comparative study of two detectors: the Euclidean detector and the Manhattan detector. The results of these comparisons shows that depending on each metric space (ROC or PR), we can obtain different conclusions on the quality of each change detector. Besides, there has been a benchmark for change detection in the field of video processing [1]. Based on these initial results, we can develop a benchmark for change detection in streaming data. Chapter 3 presents two change detectors

[1] http://www.changedetection.net/

using the Euclidean distance, and the Manhattan distance. The future work is dedicated to various distance measures, and compare the detection accuracy of these detectors by developing a benchmark for change detection in streaming data.

- Change detection methods proposed in this thesis can be seen as the first steps for developing the real world environment monitoring systems such wildfire monitoring system presented in this thesis, building monitoring system [Karnstedt 2007], or medicare monitoring systems [Sun 2010, Huang 2013]. The reasonable foundation is that most of the algorithms presented in this thesis are implemented Java. Besides, many data stream engines are available. For instance, we can integrate these change detection methods into the continuous monitoring systems by using the emerging technology DataTurbine [Fountain 2012].

Change detection in streaming data is an exciting and challenging area of research. It provides a challenging but promising source of research problems. As the nature of streaming data is dynamic, the success of many streaming applications mostly depends on the ability of change detection algorithms for detecting changes in streaming data. There are of course many open and challenging questions of the change detection in streaming data. We believe that the contributions of this dissertation provide insights that will be valuable in addressing the challenges of the data deluge.

Bibliography

[Achlioptas 2003] D. Achlioptas. *Database-friendly random projections: Johnson-Lindenstrauss with binary coins*. Journal of computer and System Sciences, vol. 66, no. 4, pages 671–687, 2003. (Cited on page 145.)

[Agarwal 2012] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei and K. Yi. *Mergeable summaries*. In Proceedings of the 31st symposium on Principles of Database Systems, pages 23–34. ACM, 2012. (Cited on page 86.)

[Aggarwal 2003] C.C. Aggarwal, J. Han, J. Wang and P.S. Yu. *A framework for clustering evolving data streams*. In Proceedings of the 29th international conference on Very large data bases-Volume 29, pages 81–92. VLDB Endowment, 2003. (Cited on pages 85, 127, 130, 135, 138, 140 and 147.)

[Aggarwal 2006] C. Aggarwal and P. Yu. *On clustering techniques for change diagnosis in data streams*. Advances in Web Mining and Web Usage Analysis, pages 139–157, 2006. (Cited on page 10.)

[Aggarwal 2007] C.C. Aggarwal and P.S. Yu. *A SURVEY OF SYNOPSIS CONSTRUCTION IN DATA STREAMS*. Data streams: models and algorithms, page 169, 2007. (Cited on pages 21, 25 and 67.)

[Aggarwal 2012] C.C Aggarwal. *A segment-based framework for modeling and mining data streams*. Knowledge and information systems, vol. 30, no. 1, pages 1–29, 2012. (Cited on page 9.)

[Akyildiz 2002] IF Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci. *Wireless sensor networks: a survey*. Computer networks, vol. 38, no. 4, pages 393–422, 2002. (Cited on page 102.)

[Alippi 2012] C. Alippi, S. Ntalampiras and M. Roveri. *An HMM-based change detection method for intelligent embedded sensors*. In Neural Networks (IJCNN), The 2012 International Joint Conference on, pages 1–7. IEEE, 2012. (Cited on page 91.)

[Babcock 2002] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom. *Models and issues in data stream systems*. In Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systemsbabcock2002models, pages 1–16. ACM, 2002. (Cited on page 67.)

[Bandyopadhyay 2006] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu and S. Datta. *Clustering distributed data streams in peer-to-peer environments*. Information Sciences, vol. 176, no. 14, pages 1952–1985, 2006. (Cited on page 128.)

[Barbará 2001] D. Barbará and P. Chen. *Tracking clusters in evolving data sets*. In Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference, pages 239–243. AAAI Press, 2001. (Cited on page 84.)

[Barbará 2002] D. Barbará. *Requirements for clustering data streams*. ACM SIGKDD Explorations Newsletter, vol. 3, no. 2, pages 23–27, 2002. (Cited on page 127.)

[Benjamini 2005] Y. Benjamini and M. Leshno. *Statistical methods for data mining*. Data Mining and Knowledge Discovery Handbook, pages 565–587, 2005. (Cited on page 113.)

[Beringer 2006] J. Beringer and E. Hullermeier. *Online clustering of parallel data streams*. Data & Knowledge Engineering, vol. 58, no. 2, pages 180–204, 2006. (Cited on page 127.)

[Bhaduri 2008] K. Bhaduri and H. Kargupta. *An efficient local algorithm for distributed multivariate regression in peer-to-peer networks*. Proceedings of SDM 08, pages 153–164, 2008. (Cited on page 15.)

[Bifet 2007] A. Bifet and R. Gavalda. *Learning from time-changing data with adaptive windowing*. In SIAM International Conference on Data Mining, pages 443–448. Citeseer, 2007. (Cited on page 42.)

[Bifet 2010a] A. Bifet. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. In Proceedings of the 2010 conference on adaptive stream mining: Pattern learning and mining from evolving data streams, pages 1–212. IOS Press, 2010. (Cited on page 15.)

[Bifet 2010b] A. Bifet, G. Holmes, R. Kirkby and B. Pfahringer. *Moa: Massive online analysis*. The Journal of Machine Learning Research, vol. 11, pages 1601–1604, 2010. (Cited on page 134.)

[Bondu 2011] A Bondu and M Boullé. *A supervised approach for change detection in data streams*. In Neural Networks (IJCNN), The 2011 International Joint Conference on, pages 519–526. IEEE, 2011. (Cited on page 22.)

[Boriah 2008] S. Boriah, V. Kumar, M. Steinbach, C. Potter and S. Klooster. *Land cover change detection: a case study*. In Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 857–865. ACM, 2008. (Cited on page 24.)

[Botan 2012] I. Botan, P. M Fischer, D. Kossmann and N. Tatbul. *Transactional stream processing*. In Proceedings of the 15th International Conference on Extending Database Technology, pages 204–215. ACM, 2012. (Cited on page 29.)

[Busch 2007] C. Busch and S. Tirthapura. *A deterministic algorithm for summarizing asynchronous streams over a sliding window*. STACS 2007, pages 465–476, 2007. (Cited on pages 113 and 114.)

[Cabanes 2012] G. Cabanes and Y. Bennani. *Change detection in data streams through unsupervised learning*. In Neural Networks (IJCNN), The 2012 International Joint Conference on, pages 1–6. IEEE, 2012. (Cited on page 23.)

[Cao 2006] F. Cao, M. Ester, W. Qian and A. Zhou. *Density-based clustering over an evolving data stream with noise*. In Proceedings of the 2006 SIAM International Conference on Data Mining, pages 328–339, 2006. (Cited on pages 85, 127 and 140.)

[Casolari 2012] S. Casolari, S. Tosi and F. Lo Presti. *An adaptive model for online detection of relevant state changes in internet-based systems*. Performance Evaluation, vol. 69, no. 5, pages 206–226, 2012. (Cited on page 67.)

[Chan 1999] K.P. Chan and A.W.C. Fu. *Efficient time series matching by wavelets*. In Data Engineering, 1999. Proceedings., 15th International Conference on, pages 126–133. IEEE, 1999. (Cited on pages 71 and 145.)

[Chang 2003] J.H. Chang and W.S. Lee. *estWin: adaptively monitoring the recent change of frequent itemsets over online data streams*. In Proceedings of the twelfth international conference on Information and knowledge management, pages 536–539. ACM, 2003. (Cited on pages 24 and 26.)

[Chen] K. Chen and L. Liu. *HE-Tree: a framework for detecting changes in clustering structure for categorical data streams*. The VLDB Journal, pages 1–20. (Cited on pages 26 and 68.)

[CHEN 2009] T. CHEN, C. YUAN, A.S. SHEIKH and C. NEUBAUER. *SEGMENT-BASED CHANGE DETECTION METHOD IN MULTIVARIATE DATA STREAM*, April 9 2009. WO Patent WO/2009/045,312. (Cited on pages 9, 10, 67 and 92.)

[Cho 2002] J. Cho and A. Ntoulas. *Effective change detection using sampling*. In Proceedings of the 28th international conference on Very Large Data Bases, pages 514–525. VLDB Endowment, 2002. (Cited on page 146.)

[Cohen 2012] E. Cohen and H. Kaplan. *How to estimate change from samples*. arXiv preprint arXiv:1203.4903, 2012. (Cited on pages 48, 52 and 147.)

[Cormode 2005] G. Cormode and M. Garofalakis. *Efficient strategies for continuous distributed tracking tasks*. IEEE Data Engineering Bulletin, vol. 28, no. 1, pages 33–39, 2005. (Cited on pages 16 and 102.)

[Cormode 2007] G. Cormode, S. Muthukrishnan and W. Zhuang. *Conquering the divide: Continuous clustering of distributed data streams*. In Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, pages 1036–1045. IEEE, 2007. (Cited on pages 128, 140 and 141.)

[Cormode 2013] G. Cormode. *The Continuous Distributed Monitoring Model*. SIGMOD Record, vol. 42, no. 1, page 5, 2013. (Cited on pages 4 and 143.)

[Curry 2007] C. Curry, R.L. Grossman, D. Locke, S. Vejcik and J. Bugajski. *Detecting changes in large data sets of payment card data: a case study*. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1018–1022. ACM, 2007. (Cited on page 68.)

[Da Silva 2010] A. Da Silva, R. Chiky and G. Hebrail. *CLUSMASTER: A Clustering Approach for Sampling Data Streams in Sensor Networks*. In Data Mining (ICDM), 2010 IEEE 10th International Conference on, pages 98–107. IEEE, 2010. (Cited on page 128.)

[Dai 2004] B.R. Dai, J.W. Huang, M.Y. Yeh and M.S. Chen. *Clustering on demand for multiple data streams*. In Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on, pages 367–370. IEEE, 2004. (Cited on page 127.)

[Das 2009] K. Das, K. Bhaduri, S. Arora, W. Griffin, K. Borne, C. Giannella and H. Kargupta. *Scalable Distributed Change Detection from Astronomy Data Streams using Local, Asynchronous Eigen Monitoring Algorithms*. In SIAM International Conference on Data Mining, Nevada, 2009. (Cited on pages 16, 27, 28 and 29.)

[Dasu 2005] T. Dasu, S. Krishnan, S. Venkatasubramanian and K. Yi. *An information-theoretic approach to detecting changes in multi-dimensional data streams*. In 38th Symposium on the Interface of Statistics, Computing Science, and Applications. Citeseer, 2005. (Cited on page 24.)

[Dasu 2009] T. Dasu, S. Krishnan, D. Lin, S. Venkatasubramanian and K. Yi. *Change (Detection) You Can Believe in: Finding Distributional Shifts in Data Streams*. In Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII, page 34. Springer, 2009. (Cited on pages 7, 22, 26 and 92.)

[Dasu 2011] T. Dasu, S. Krishnan and G.M. Pomann. *Robustness of change detection algorithms*. In Advances in Intelligent Data Analysis X, pages 125–137. Springer, 2011. (Cited on page 148.)

[Datta 2006] S. Datta, K. Bhaduri, C. Giannella, R. Wolff and H. Kargupta. *Distributed data mining in peer-to-peer networks*. IEEE Internet Computing, pages 18–26, 2006. (Cited on pages 29 and 129.)

[Davenport 2006] M. A Davenport, M. B Wakin and R. G Baraniuk. *Detection and estimation with compressive measurements*. Dept. of ECE, Rice University, Tech. Rep, 2006. (Cited on page 147.)

[Davis 2006] J. Davis and M. Goadrich. *The relationship between Precision-Recall and ROC curves*. In Proceedings of the 23rd international conference on Machine learning, pages 233–240. ACM, 2006. (Cited on pages 34 and 35.)

[Dean 2008] J. Dean and S. Ghemawat. *MapReduce: Simplified data processing on large clusters*. Communications of the ACM, vol. 51, no. 1, pages 107–113, 2008. (Cited on page 3.)

[Demers 1987] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry. *Epidemic algorithms for replicated database maintenance*. In Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, pages 1–12. ACM, 1987. (Cited on page 107.)

[Dindar 2011] N. Dindar, P. M Fischer, M. Soner and N. Tatbul. *Efficiently correlating complex events over live and archived data streams*. In ACM DEBS Conference, 2011. (Cited on page 24.)

[Dindar 2012] N. Dindar, N. Tatbul, R. J Miller, Laura M. Haas and I. Botan. *Modeling the execution semantics of stream processing engines with SECRET*. The VLDB Journal, pages 1–26, 2012. (Cited on page 41.)

[Dong] G. Dong, J. Han, L.V.S. Lakshmanan, J. Pei, H. Wang and P.S. Yu. *Online mining of changes from data streams: Research problems and preliminary results*. Citeseer. (Cited on page 20.)

[Draisbach 2012] U. Draisbach, F. Naumann, S. Szott and O. Wonneberg. *Adaptive windows for duplicate detection*. In Data Engineering (ICDE), 2012 IEEE 28th International Conference on, pages 1073–1083. IEEE, 2012. (Cited on page 42.)

[Fawcett 2004] T. Fawcett. *ROC graphs: Notes and practical considerations for researchers*. Machine Learning, vol. 31, pages 1–38, 2004. (Cited on pages 35 and 59.)

[Fountain 2012] T. Fountain, S. Tilak, P. Shin and M. Nekrasov. *The open source dataturbine initiative: empowering the scientific community with streaming data middleware*. Bulletin of the Ecological Society of America, vol. 93, no. 3, pages 242–252, 2012. (Cited on pages xiii, 6 and 149.)

[Gaber 2006] M.M. Gaber and P.S. Yu. *Classification of Changes in Evolving Data Streams using Online Clustering Result Deviation*. In Proceedings of the third International Workshop on Knowledge Discovery in Data Streams June, volume 29. Citeseer, 2006. (Cited on pages 9, 65 and 68.)

[Gama 2007] J. Gama and M. M. Gaber. Learning from data streams: processing techniques in sensor networks, volume 21. Springer, 2007. (Cited on page 8.)

[Gandy 2013] A. Gandy and F. D. Lau. *Non-restarting cumulative sum charts and control of the false discovery rate*. Biometrika, vol. 100, no. 1, pages 261–268, 2013. (Cited on page 114.)

[Ganguly 2008] A. R Ganguly, J. Gama, O. A Omitaomu, M. M. Gaber and R. R. Vatsavai. Knowledge discovery from sensor data, volume 7. CRC, 2008. (Cited on page 29.)

[Ganti 1999] V. Ganti, J. Gehrke and R. Ramakrishnan. *A framework for measuring changes in data characteristics*. In Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 126–137. ACM, 1999. (Cited on pages 25 and 68.)

[Ganti 2002] V. Ganti, J. Gehrke, R. Ramakrishnan and W.Y. Loh. *A framework for measuring differences in data characteristics*. Journal of Computer and System Sciences, vol. 64, no. 3, pages 542–578, 2002. (Cited on pages 25 and 68.)

[Geisler 2010] S. Geisler, C. Quix and S. Schiffer. *A data stream-based evaluation framework for traffic information systems*. In Proceedings of the ACM SIGSPATIAL International Workshop on GeoStreaming, pages 11–18. ACM, 2010. (Cited on page 19.)

[Girod 2007] L. Girod, Y. Mei, R. Newton, S. Rost, A. Thiagarajan, H. Balakrishnan and S. Madden. *The case for a signal-oriented data stream management system*. In Proc. CIDR. Citeseer, 2007. (Cited on page 73.)

[Girod 2008] L. Girod, Y. Mei, R. Newton, S. Rost, A. Thiagarajan, H. Balakrishnan and S. Madden. *Xstream: A signal-oriented data stream management system*. In Proc. ICDE, pages 1180–1189. IEEE, 2008. (Cited on page 73.)

[Golab 2003] L. Golab and M.T. Özsu. *Issues in data stream management*. ACM Sigmod Record, vol. 32, no. 2, pages 5–14, 2003. (Cited on page 41.)

[Golab 2009] L. Golab, T. Johnson, J. S. Seidel and V. Shkapenyuk. *Stream warehousing with DataDepot*. In Proceedings of the 35th SIGMOD international conference on Management of data, pages 847–854. ACM, 2009. (Cited on pages 7 and 25.)

[Gretton 2012] A. Gretton, K.M. Borgwardt, M.J. Rasch, B. Schölkopf and A. Smola. *A kernel two-sample test*. The Journal of Machine Learning Research, vol. 13, pages 723–773, 2012. (Cited on page 92.)

[Guha 2003] S. Guha, A. Meyerson, N. Mishra, R. Motwani and L. O'Callaghan. *Clustering data streams: Theory and practice*. Knowledge and Data Engineering, IEEE Transactions on, vol. 15, no. 3, pages 515–528, 2003. (Cited on page 126.)

[Gustafsson 1997] F. Gustafsson and J. Palmqvist. *Change detection design for low false alarm rates*. threshold, vol. 10, page 4, 1997. (Cited on page 91.)

[Gustafsson 2001] F. Gustafsson and F. Gustafsson. Adaptive filtering and change detection. Wiley Londres, 2001. (Cited on page 68.)

[Handy 2002] MJ Handy, M. Haase and D. Timmermann. *Low energy adaptive clustering hierarchy with deterministic cluster-head selection*. In Mobile and Wireless Communications Network, 2002. 4th International Workshop on, pages 368–372. IEEE, 2002. (Cited on pages 11 and 126.)

[Hassanieh 2012a] H. Hassanieh, P. Indyk, D. Katabi and E. Price. *Nearly optimal sparse Fourier transform*. In Proceedings of the 44th symposium on Theory of Computing, pages 563–578. ACM, 2012. (Cited on pages 73, 81 and 147.)

[Hassanieh 2012b] H. Hassanieh, P. Indyk, D. Katabi and E. Price. *Simple and practical algorithm for sparse Fourier transform*. In Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1183–1194. SIAM, 2012. (Cited on pages 73, 81 and 147.)

[Heinzelman 2000] W. R. Heinzelman, A. Chandrakasan and H. Balakrishnan. *Energy-efficient communication protocol for wireless microsensor networks*. In System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on, pages 10–pp. IEEE, 2000. (Cited on page 104.)

[Hernández-Orallo 2012] J. Hernández-Orallo, P. Flach and C. Ferri. *A unified view of performance metrics: translating threshold choice into expected classification loss*. Journal of Machine Learning Research, vol. 13, pages 2813–2869, 2012. (Cited on page 53.)

[Hey 2009] A. JG Hey, S. Tansley and K. M. Tolle. The fourth paradigm: data-intensive scientific discovery. Microsoft Research Redmond, WA, 2009. (Cited on page 3.)

[Hido 2008] S. Hido, T. Idé, H. Kashima, H. Kubo and H. Matsuzawa. *Unsupervised change analysis using supervised learning*. In Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining, pages 148–159. Springer-Verlag, 2008. (Cited on page 19.)

[Hirte 2012] S. Hirte, A. Seifert, S. Baumann, D. Klan and K. U. Sattler. *Data3–A Kinect Interface for OLAP Using Complex Event Processing*. In Data Engineering (ICDE), 2012 IEEE 28th International Conference on, pages 1297–1300. IEEE, 2012. (Cited on page 91.)

[Ho 2007] S.S. Ho and H. Wechsler. *Detecting changes in unlabeled data streams using martingale*. In Proceedings of the 20th international joint conference on Artifical intelligence, pages 1912–1917. Morgan Kaufmann Publishers Inc., 2007. (Cited on page 23.)

[Huang 2003] W. Huang, E.R. Omiecinski and L. Mark. *Evolution in Data Streams*. 2003. (Cited on pages 21 and 25.)

[Huang 2009] Weiyun Huang, Edward Omiecinski, Leo Mark and Minh Nguyen. *History Guided Low-Cost Change Detection in Streams*. Data Warehousing and Knowledge Discovery, pages 75–86, 2009. (Cited on page 25.)

[Huang 2013] G. Huang, J. He, J. Cao, Z. Qiao, M. Steyn& and K. Taraporewalla&6. *A Real-Time Abnormality Detection System for Intensive Care Management*. In Proceeding of ICDE 2013 Conference, 2013. (Cited on pages 148 and 149.)

[Ikonomovska 2009] E. Ikonomovska, J. Gama, R. Sebastião and D. Gjorgjevik. *Regression trees from data streams with drift detection*. In Discovery Science, pages 121–135. Springer, 2009. (Cited on page 26.)

[Jain 1999] A.K. Jain, M.N. Murty and P.J. Flynn. *Data clustering: a review*. ACM computing surveys (CSUR), vol. 31, no. 3, pages 264–323, 1999. (Cited on page 126.)

[Jain 2008] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts and S. Zdonik. *Towards a streaming SQL standard*. Proceedings of the VLDB Endowment, vol. 1, no. 2, pages 1379–1390, 2008. (Cited on pages 41, 42 and 144.)

[Juang 2002] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh and D. Rubenstein. *Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet*. In ACM Sigplan Notices, volume 37, pages 96–107. ACM, 2002. (Cited on page 12.)

[Kargupta 2001] H. Kargupta and B.H. Park. *Mining decision trees from data streams in a mobile environment*. In Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, pages 281–288. IEEE, 2001. (Cited on page 71.)

[Karnstedt 2007] M. Karnstedt, K.-U. Sattler and J. Quasebarth. *Incremental Mining for Facility Management*. LWA 2007 Lernen–Wissen–Adaption, page 183, 2007. (Cited on pages 126, 127 and 149.)

[Karnstedt 2009] M. Karnstedt, D. Klan, C. Pölitz, K.-U. Sattler and C. Franke. *Adaptive burst detection in a stream engine*. In Proceedings of the 2009 ACM symposium on Applied Computing, pages 1511–1515. ACM, 2009. (Cited on page 20.)

[Kawahara 2009] Y. Kawahara and M. Sugiyama. *Change-point detection in time-series data by direct density-ratio estimation*. In Proceedings of 2009 SIAM International Conference on Data Mining (SDM2009), pages 389–400, 2009. (Cited on page 19.)

[Kifer 2004] D. Kifer, S. Ben-David and J. Gehrke. *Detecting change in data streams*. In Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, page 191. VLDB Endowment, 2004. (Cited on pages 14, 23, 24, 43, 81, 144 and 147.)

[Kim 2008] Y. M. Kim, G. Zheng, S. H. Sohn and J. M. Kim. *An alternative energy detection using sliding window for cognitive radio system*. In Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on, volume 1, pages 481–485. IEEE, 2008. (Cited on page 66.)

[Kim 2009] A.Y. Kim, C. Marzban, D.B. Percival and W. Stuetzle. *Using labeled data to evaluate change detectors in a multivariate streaming environment*. Signal Processing, vol. 89, no. 12, pages 2529–2536, 2009. (Cited on pages 22 and 92.)

[Klan 2011] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann and K.U. Sattler. *Stream engines meet wireless sensor networks: cost-based planning and processing of complex queries in AnduIN*. Distributed and Parallel Databases, vol. 29, no. 1, pages 151–183, 2011. (Cited on pages 11, 67, 125 and 127.)

[Kranen 2009] P. Kranen, I. Assent, C. Baldauf and T. Seidl. *Self-adaptive anytime stream clustering*. In Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on, pages 249–258. IEEE, 2009. (Cited on page 127.)

[Krishnamurthy 2003] B. Krishnamurthy, S. Sen, Y. Zhang and Y. Chen. *Sketch-based change detection: Methods, evaluation, and applications*. In Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, pages 234–247. ACM New York, NY, USA, 2003. (Cited on pages 22 and 67.)

[Kuncheva 2011] L. Kuncheva. *Change Detection in Streaming Multivariate Data Using Likelihood Detectors*. Knowledge and Data Engineering, IEEE Transactions on, no. 99, pages 1–1, 2011. (Cited on pages 22, 23 and 92.)

[Lai 2008] L. Lai, Y. Fan and H.V. Poor. *Quickest detection in cognitive radio: A sequential change detection framework*. In Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, pages 1–5. IEEE, 2008. (Cited on page 73.)

[Li 2006] Y. Li, Z. Wang and Y. Song. *Wireless sensor network design for wildfire monitoring*. In Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on, volume 1, pages 109–113. IEEE, 2006. (Cited on pages xiii and 5.)

[Li 2007] W. Li, X. Jin and X. Ye. *Detecting change in data stream: Using sampling technique*. In Natural Computation, 2007. ICNC 2007. Third International Conference on, volume 1, pages 130–134. IEEE, 2007. (Cited on pages 67 and 146.)

[Liarou 2012] E. Liarou, S. Idreos, S. Manegold and M. Kersten. *MonetDB/DataCell: online analytics in a streaming column-store*. Proceedings of the VLDB Endowment, vol. 5, no. 12, pages 1910–1913, 2012. (Cited on pages 42 and 147.)

[Liarou 2013] Erietta Liarou, Stratos Idreos, Stefan Manegold and Martin Kersten. *Enhanced stream processing in a DBMS kernel*. In Proceedings of the 16th International Conference on Extending Database Technology, pages 501–512. ACM, 2013. (Cited on pages 42 and 147.)

[Liu 2010] X. Liu, X. Wu, H. Wang, R. Zhang, J. Bailey and K. Ramamohanarao. *Mining distribution change in stock order streams*. Prof. of ICDE, pages 105–108, 2010. (Cited on pages 20, 23 and 58.)

[Macmillan 2005] N.A. Macmillan and C.D. Creelman. Detection theory: A user's guide. Lawrence Erlbaum, 2005. (Cited on page 31.)

[Madden 2002] S. Madden and M. J Franklin. *Fjording the stream: An architecture for queries over streaming sensor data*. In Data Engineering, 2002. Proceedings. 18th International Conference on, pages 555–566. IEEE, 2002. (Cited on pages 109 and 110.)

[Manku 2002] G.S. Manku and R. Motwani. *Approximate frequency counts over data streams*. In Proceedings of the 28th international conference on Very Large Data Bases, pages 346–357. VLDB Endowment, 2002. (Cited on page 24.)

[Manyika 2011] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh and A.H. Byers. *Big data: The next frontier for innovation, competition and productivity*. McKinsey Global Institute, May, 2011. (Cited on page 3.)

[Maslov 2012] A. Maslov, M. Pechenizkiy, T. Kärkkäinen and M. Tähtinen. *Quantile index for gradual and abrupt change detection from CFB boiler sensor data in online settings*. In Proceedings of the Sixth International Workshop on Knowledge Discovery from Sensor Data, pages 25–33. ACM, 2012. (Cited on page 24.)

[Massey Jr 1951] Frank J Massey Jr. *The Kolmogorov-Smirnov test for goodness of fit*. Journal of the American statistical Association, vol. 46, no. 253, pages 68–78, 1951. (Cited on page 14.)

[Masud 2008] M.M. Masud, J. Gao, L. Khan, J. Han and B. Thuraisingham. *A practical approach to classify evolving data streams: Training with limited amount of labeled data*. In Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, pages 929–934. IEEE, 2008. (Cited on page 147.)

[Mörchen 2003] F. Mörchen. Time series feature extraction for data mining using dwt and dft. 2003. (Cited on page 71.)

[Muthukrishnan 2005] S. Muthukrishnan. Data streams: Algorithms and applications. Now Publishers Inc, 2005. (Cited on pages 22 and 23.)

[Muthukrishnan 2007] S. Muthukrishnan, E. van den Berg and Y. Wu. *Sequential change detection on data streams*. ICDM Workshops, 2007. (Cited on page 23.)

[Naor 1993] M. Naor and L. Stockmeyer. *What can be computed locally?* pages 184–193, 1993. (Cited on pages 29 and 129.)

[Nelson 2012] J. Nelson. *Sketching and streaming algorithms for processing massive data*. XRDS: Crossroads, The ACM Magazine for Students, vol. 19, no. 1, pages 14–19, 2012. (Cited on page 67.)

[Neuman 2011] B. Neuman, B. Sofman, A. Stentz and J.A. Bagnell. *Segmentation-based online change detection for mobile robots*. In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 5427–5434. IEEE, 2011. (Cited on page 91.)

[Neyman 1933] J. Neyman and E.S. Pearson. *On the problem of the most efficient tests of statistical hypotheses*. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, vol. 231, no. 694-706, page 289, 1933. (Cited on page 33.)

[Ng 2008a] W. Ng and M. Dash. *A change detector for mining frequent patterns over evolving data streams*. In Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on, pages 2407–2412. IEEE, 2008. (Cited on page 26.)

[Ng 2008b] W. Ng and M. Dash. *A test paradigm for detecting changes in transactional data streams*. In Database Systems for Advanced Applications, pages 204–219. Springer, 2008. (Cited on page 24.)

[Nikovski 2010] D. Nikovski and A. Jain. *Fast adaptive algorithms for abrupt change detection*. Machine learning, vol. 79, no. 3, pages 283–306, 2010. (Cited on page 24.)

[Niu 2004] R. Niu, M. Moore and D. Klamer. *Decision fusion in a wireless sensor network with a large number of sensors*. 2004. (Cited on page 112.)

[Niu 2006a] R. Niu and P.K. Varshney. *Performance evaluation of decision fusion in wireless sensor networks*. In Information Sciences and Systems, 2006 40th Annual Conference on, pages 69–74. IEEE, 2006. (Cited on pages 16 and 123.)

[Niu 2006b] R. Niu, P.K. Varshney and Q. Cheng. *Distributed detection in a large wireless sensor network*. Information Fusion, vol. 7, no. 4, pages 380–394, 2006. (Cited on pages 16 and 123.)

[Oppenheim 1989] A.V. Oppenheim, R.W. Schafer, J.R. Buck*et al.* Discrete-time signal processing, volume 2. Prentice hall Englewood Cliffs, NJ:, 1989. (Cited on page 72.)

[Palpanas 2003] T. Palpanas, D. Papadopoulos, V. Kalogeraki and D. Gunopulos. *Distributed deviation detection in sensor networks*. ACM SIGMOD Record, vol. 32, no. 4, pages 77–82, 2003. (Cited on pages 16, 28, 29 and 102.)

[Pham 2012] D.-S. Pham, S. Venkatesh, M. Lazarescu and S. Budhaditya. *Anomaly detection in large-scale data stream networks*. Data Mining and Knowledge Discovery, pages 1–45, 2012. (Cited on page 16.)

[Rajagopal 2008] R. Rajagopal, X.L. Nguyen, S.C. Ergen and P. Varaiya. *Distributed online simultaneous fault detection for multiple sensors*. In Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on, pages 133–144. IEEE, 2008. (Cited on pages 15 and 123.)

[Ray 2008] P. Ray and P.K.V. Fellow. *Distributed detection in wireless sensor networks using dynamic sensor thresholds*. International Journal of Distributed Sensor Networks, vol. 4, no. 1, pages 5–12, 2008. (Cited on page 112.)

[Ray 2011] P. Ray and P. K Varshney. *False Discovery Rate Based Sensor Decision Rules for the Network-Wide Distributed Detection Problem*. Aerospace and Electronic Systems, IEEE Transactions on, vol. 47, no. 3, pages 1785–1799, 2011. (Cited on pages 105 and 148.)

[Roddick 2000] J.F. Roddick, L. Al-Jadir, L. Bertossi, M. Dumas, H. Gregersen, K. Hornsby, J. Lufter, F. Mandreoli, T. Mannisto, E. Mayol*et al.* *Evolution and change in data management:issues and directions*. ACM Sigmod Record, vol. 29, no. 1, pages 21–25, 2000. (Cited on pages 18 and 20.)

[Ross 2009] G.J. Ross, D.K. Tasoulis and N.M. Adams. *Online annotation and prediction for regime switching data streams*. In Proceedings of the 2009 ACM symposium on Applied Computing, pages 1501–1505. ACM, 2009. (Cited on pages 4 and 21.)

[Sagen 2010] A. Sagen, C. Labbé, M. M. Gaber, S. Krishnaswamy, A. B. Waluyo and S. Loke. *A Data Clustering Approach to Energy Conservation in Wireless Sensor Networks*. In Workshop on Ubiquitous Data Mining, pages 13–17, 2010. (Cited on pages 11 and 126.)

[Sebastião 2007] R. Sebastião and J. Gama. *Change detection in learning histograms from data streams*. Progress in Artificial Intelligence, pages 112–123, 2007. (Cited on page 67.)

[Severo 2006] M. Severo and J. Gama. *Change detection with kalman filter and CUSUM*. In Proceedings of the 9th international conference on Discovery Science, pages 243–254. Springer-Verlag, 2006. (Cited on page 68.)

[Singh 1989] A. Singh. *Review Article Digital change detection techniques using remotely-sensed data*. International Journal of Remote Sensing, vol. 10, no. 6, pages 989–1003, 1989. (Cited on page 19.)

[Song 2007] X. Song, M. Wu, C. Jermaine and S. Ranka. *Statistical change detection for multi-dimensional data*. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 667–676. ACM, 2007. (Cited on pages 24 and 49.)

[Sun 2006] J. Sun, S. Papadimitriou and C. Faloutsos. *Distributed pattern discovery in multiple streams*. Advances in Knowledge Discovery and Data Mining, pages 713–718, 2006. (Cited on page 126.)

[Sun 2010] J. Sun, D. Sow, J. Hu and S. Ebadollahi. *A system for mining temporal physiological data streams for advanced prognostic decision support*. In Data Mining (ICDM), 2010 IEEE 10th International Conference on, pages 1061–1066. IEEE, 2010. (Cited on pages 148 and 149.)

[Tao 2009] Y. Tao and M T. Ozsu. *Mining data streams with periodically changing distributions*. In Proceedings of the 18th ACM conference on Information and knowledge management, pages 887–896. ACM, 2009. (Cited on page 19.)

[Tran 2011a] D.-H. Tran and K.-U. Sattler. *On detection of changes in sensor data streams*. In Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, pages 50–57. ACM, 2011. (Cited on pages 24 and 29.)

[Tran 2011b] D.-H. Tran, J. Yang and K.-U Sattler. *Decentralized change detection in wireless sensor network using dft-based synopsis*. In Mobile Data Management (MDM), 2011 12th IEEE International Conference on, volume 1, pages 226–235. IEEE, 2011. (Cited on pages 12, 29, 119 and 148.)

[Tschumitschew 2010] Katharina Tschumitschew and Frank Klawonn. *Incremental quantile estimation*. Evolving Systems, vol. 1, pages 253–264, 2010. (Cited on page 73.)

[Valizadegan 2007] H. Valizadegan and P.N. Tan. *A Prototype-driven Framework for Change Detection in Data Stream Classification*. In IEEE Symposium on Computational Intelligence and Data Mining, 2007. CIDM 2007, pages 88–95. Citeseer, 2007. (Cited on page 68.)

[van Leeuwen 2008] M. van Leeuwen and A. Siebes. *Streamkrimp: Detecting change in data streams*. Machine Learning and Knowledge Discovery in Databases, pages 672–687, 2008. (Cited on page 24.)

[Varshney 1986] PK Varshney*et al. Optimal data fusion in multiple sensor detection systems*. Aerospace and Electronic Systems, IEEE Transactions on, no. 1, pages 98–101, 1986. (Cited on pages 104 and 109.)

[Varshney 1997] P.K. Varshney and CS Burrus. Distributed detection and data fusion. Springer Verlag, 1997. (Cited on page 28.)

[Veeravalli 2012] V.V. Veeravalli and P.K. Varshney. *Distributed inference in wireless sensor networks*. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 370, no. 1958, pages 100–117, 2012. (Cited on pages 27, 28 and 104.)

[VISION 2007] AR VISION. *Wireless Sensors in Distributed Detection Applications*. IEEE SIGNAL PROCESSING MAGAZINE, vol. 1053, no. 5888/07, 2007. (Cited on page 102.)

[Vitter 1985] J. S Vitter. *Random sampling with a reservoir*. ACM Transactions on Mathematical Software (TOMS), vol. 11, no. 1, pages 37–57, 1985. (Cited on page 146.)

[Yang 2011] J. Yang, T. Simon, C. Mueller, D. Klan and K.U Sattler. *Comparing and Refining Gossip Protocols for Fault Tolerance in Wireless P2P Systems*. In Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on, pages 595–599. IEEE, 2011. (Cited on pages 12, 110 and 119.)

[Yin 2008] J. Yin and M. M. Gaber. *Clustering distributed time series in sensor networks*. In Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, pages 678–687. IEEE, 2008. (Cited on pages 126 and 128.)

[Younis 2004] O. Younis and S. Fahmy. *HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks*. Mobile Computing, IEEE Transactions on, vol. 3, no. 4, pages 366–379, 2004. (Cited on pages 11 and 126.)

[Zaki 2002] M.J. Zaki and Y. Pan. *Introduction: recent developments in parallel and distributed data mining*. Distributed and Parallel Databases, vol. 11, no. 2, pages 123–127, 2002. (Cited on page 126.)

[Zhang 1996] T. Zhang, R. Ramakrishnan and M. Livny. *BIRCH: an efficient data clustering method for very large databases*. ACM SIGMOD Record, vol. 25, no. 2, pages 103–114, 1996. (Cited on pages 84 and 85.)

[Zhang 2008] Q. Zhang, J. Liu and W. Wang. *Approximate Clustering on Distributed Data Streams*. In ICDE, pages 1131–1139, 2008. (Cited on pages 128 and 141.)

[Zhou 2007] A. Zhou, F. Cao, Y. Yan, C. Sha and X. He. *Distributed data stream clustering: A fast EM-based approach*. In Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, pages 736–745. Ieee, 2007. (Cited on pages 140 and 141.)

[Zhou 2008] A. Zhou, F. Cao, W. Qian and C. Jin. *Tracking clusters in evolving data streams over sliding windows*. Knowledge and Information Systems, vol. 15, no. 2, pages 181–214, 2008. (Cited on page 26.)

[Zhu 2002] Y. Zhu and D. Shasha. *Statstream: Statistical monitoring of thousands of data streams in real time*. In Proceedings of the 28th international conference on Very Large Data Bases, pages 358–369. VLDB Endowment, 2002. (Cited on pages 14, 72 and 73.)

[Zhu 2010] X. Zhu. *Stream Data Mining Repository*. http://www.cse.fau.edu/ xqzhu/stream.html, 2010., 2010. (Cited on pages 93 and 134.)

[Zliobaite and 2012] I. Zliobaite and and B. Gabrys. *Adaptive Preprocessing for Streaming Data*. Knowledge and Data Engineering, IEEE Transactions on, vol. PP99, no. 99, page 1, 2012. (Cited on page 68.)

[Zou 2006] J. Zou, A. Gilbert, M. Strauss and I. Daubechies. *Theoretical and experimental analysis of a randomized algorithm for sparse Fourier transform analysis*. Journal of Computational physics, vol. 211, no. 2, pages 572–595, 2006. (Cited on pages 73 and 81.)

[Zouari 2008] R. Zouari, L. Mevel and M. Basseville. *An Adaptive Statistical Approach to Flutter Monitoring*. In World Congress, volume 17, pages 12024–12029, 2008. (Cited on page 19.)