ONLINE OUTLIER DETECTION OVER DATA STREAMS

by

Hongyin Cui
B.Sc., Simon Fraser University, 2002

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

In the School
of
Computing Science

SIMON FRASER UNIVERSITY

Fall 2005

APPROVAL

Name: **Hongyin Cui**

Degree: **Master of Science**

Title of Thesis: **Online Outlier Detection Over Data Streams**

Examining Committee:

Chair: **Qianping Gu**
Professor of School of Computing Science

Dr. Martin Ester
Senior Supervisor
Associate Professor of School of Computing Science

Dr. Jian Pei
Supervisor
Assistant Professor of School of Computing Science

Dr. Jiangchuan Liu
Examiner
Assistant Professor of School of Computing Science

Date Defended/Approved: Dec. 1/2005

ABSTRACT

Outlier detection is an important data mining task. Recently, online discovering outlier under data stream model has attracted attention for many emerging applications, such as network intrusion detection. Because the algorithms on data streams are restricted to fulfil their works with only one pass over data sets and limited resources, it is a very challenging problem to detect outliers over streams.

In this paper, we present an unsupervised outlier detection approach to online network intrusion detection over data streams. Our method continuously maintains online summary and obtains a set of clusters, and those small clusters far away from big clusters are regarded as outlier clusters. We also propose a novel definition of outlier degree to measure the outlying degree of each cluster. When a new data arrives, it is considered as an outlier if it lies in the top-k outlier clusters.

Experiment results demonstrate the effectiveness of our method.

Keyword

Outlier detection, network intrusion detection, data mining, data stream

DEDICATION

To Andy

ACKNOWLEDGEMENTS

I would like to thank my senior supervisor, Dr. Martin Ester. He has provided me with ideas, guidance and encouragement as a supervisor. He has also given me strength and support in my most difficult time as a friend. I feel exceedingly appreciated to have had his guidance and I owe him a great many heartfelt thanks.

I would also like to thank my supervisor, Dr. Jian Pei. His suggestions and input are very helpful. Thank Dr. Jiangchuan Liu for being my examiner of this thesis.

My appreciation also goes to Heather Muckart for the administration procedure.

My deepest gratitude and appreciation is reserved for my parents, my wife and other family members. To My beloved wife Ping Wang, thank you for your love and unlimited support both on the study and on the daily life. You are my biggest treasure. To my parents, thank you for your endless love, your help, your support and your patience. Thank you for taking care of Andy in the period of my studying. To my brother and sister in law, thank you for giving us both spiritual and financial support. You make us feel that we are not alone here.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1:INTRODUCTION

1.1 Background Information and Motivation

An outlier in a dataset is defined informally as an observation that is considerably different from the remainder as if it is generated by a different mechanism. Outlier detection is an important data mining research direction with numerous applications, including credit card fraud detection, discovery of criminal activities in electronic commerce, weather prediction, marketing and customer segmentation.

Recently, online discovering outliers under data stream model have attracted attention for many emerging applications, such as network intrusion detection. With the increased usage of computer networks, security becomes a critical issue. A network intrusion can cause severe disruption to networks. Therefore the development of a robust and reliable network intrusion detection system is increasingly important.

Traditional methods for intrusion detection employ signature-based detection. These methods extract features from data streams, and detect intrusions by comparing the feature values to a set of attack signatures provided by human experts. Such methods can only detect previously known intrusions since these intrusions have a corresponding signature. The signature database has to be manually revised for each new type of attack that is discovered and until this revision, systems are vulnerable to these attacks.

Because of this, applying data mining techniques is a promising approach. In data mining community, intrusion detection can be solved by outlier detection over data streams.

A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. For many recent applications, the concept of *data stream* is more appropriate than a dataset. By nature, a stored static dataset is an appropriate model when significant portions of the data are queried again and again, and updates are relatively infrequent. In contrast, a data stream is an appropriate model when a large volume of data arriving continuously and it is either unnecessary or impractical to store the data in some form of memory. Data streams are also appropriate as a model of access to large datasets stored in secondary memory where performance requirements necessitate linear scans [15].

Outlier detection has been widely studied in static data sets. *Distance-based* outlier is first presented by Knorr and Ng [6]. A distance-based outlier in a dataset D is a data object with at lease $p\%$ of the objects in D having a distance of more than d_{min} away from it. It is further extended in [7] based on the distance of a point from its k^{th} nearest neighbour. All the data objects are ranked by the distance to their k^{th} nearest neighbour, and the top-k data objects are identified as outliers. Alternatively, in the algorithm proposed by Angiuli and Pizzuti [9], each data point is ranked by the sum of distances from its k nearest neighbours. Distance-based outliers are meaningful and adequate under certain conditions, but not satisfactory for the general case when clusters of different densities exist.

To avoid the shortcomings in distance-based outlier definition, Breunig [8] introduced the concept of "*local outliers*", which are the objects outlying relative to their local neighbourhoods, particularly with respect to the densities of the neighbourhoods. The outlier rank of a data object is determined by "local outlier factor (LOF)". Although the concept of local outliers is a useful one, the computation of LOF value for every data object requires a large number of k-nearest neighbours searches and can be computationally expensive.

Aggarwal and Yu [10] discussed a technique for outlier detection, which finds outliers by observing the density distribution of projections from the data. That is, their definition considers a point to be an outlier, if in some lower dimensional projection it is present in a local region of abnormally low density. It also has an expensive computation.

Clustering-based outlier detection techniques regard small clusters as outliers [14]. It first performs fixed-width clustering over the points with a radius of w, and then sort the clusters based on the size. The points in the small clusters are labelled as outliers. This algorithm requires only one pass through the dataset.

Although outlier detection has been widely studied, however, there has not been much research work conducted on detecting outliers in dynamic data streams, and most existing methods are not appropriate in data stream environment.

In data mining community there has been some work addressing data streams, but these proposals mainly solve the problems of clustering [1, 2], mining frequent patterns [3, 4], data analysis and query processing [5], and do not address the problem of online outlier detection over data stream.

In the data stream model, data points can only be accessed in the order of their arrivals and random access is disallowed. The space available to store information is supposed to be small comparing to the huge size of the unbounded data streams. Thus, the data mining algorithms on data streams are restricted to be able to fulfil their works with only one pass over data sets and limited resources. In addition, outliers may be the seeds of a new cluster and clusters may also become outliers in the course of the time. The data distribution is dynamically changed over data streams. So it is a very challenging problem to detect outliers in stream environment.

In this paper, we present an unsupervised outlier detection approach for online network intrusion detection over the data streams. Our method continuously keeps online summary and obtains a set of clusters over the data stream, and those small clusters far away from big clusters are regarded as outlier clusters. We also propose a novel definition of outlier degree to measure the outlying degree of each cluster. When a new data arrives, it is considered as an outlier if it lies in the top-k outlier clusters.

For this application, we make the following assumptions:

Assumption 1 *The majority of data in the stream are normal. Only a small number of data are outliers [12].*

Assumption 2 *The outliers are statistically different from normal data [13].*

These two assumptions are two different issues and both are important. If we only have Assumption 1, outliers may have the similar or same behaviour of normal data. With Assumption 2, we ensure that the outliers as a group must have a quite different behaviour from normal data.

Since our approach aggregates the online summary of clusters in the data stream, temporal decaying is not considered. Moreover, sparse data points between bursts that lie in the same spatial cluster are not likely to be network attacks. Thus we won't consider the time dimension in this thesis.

1.2 Contribution

This thesis makes the following contributions:

- Propose a novel method for online outlier detection over data stream. Our approach can immediately predict if a data object is an outlier, when it arrives.
- Propose a novel definition for outlier: *cluster-based outlier,* which is natural for data stream and has numerous applications, and introduce a corresponding outlier degree measurement.
- Present an efficient and effective grid-based online summarization algorithm that successfully satisfies the requirement in run time complexity and memory consumption for data stream.
- Besides a binary detecting answer, provide more information or description about outliers, such as the outlier degree, the means of outlier clusters, etc.

We evaluate our method using the 1999 KDD Cup data set, which is a very popular and widely used intrusion attacks data set. Our result shows the very good accuracy and recall.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

- In Chapter 2, we provide an overview of related work systematically.
- Chapter 3 defines the problem, and then presents our approach.
- Chapter 4 describes the data set and discuss the experiment results.
- In Chapter 5, we finally summarize our work and discuss future work.

CHAPTER 2:RELATED WORK

In this Chapter, we review several previous studies that are related to our work technically. Section 2.1 presents some related work on mining data stream. Section 2.2 discusses several previous researches on outlier detection. Section 2.3 provides a survey of network intrusion detection and existing solutions in network research area.

2.1 Related Work on Mining Data Streams

2.1.1 CluStream

This algorithm was introduced by Charu Agarwal, Jiawei Han and Philip Yu [1] for clustering evolving data, which evolves considerably over time. The idea is divide the clustering process into an online component that periodically stores detailed summary statistics and an offline component that uses only this summary statistics.

We first begin by defining the concept of micro-clusters.

Definition 1 A *micro-cluster* for a set of d-dimensional points $X_{i1} \ldots X_{in}$ with time stamps $T_{i1} \ldots T_{in}$ is defined as the (2d + 3) tuple ($CF2^x$, $CF1^x$, $CF2^t$, $CF1^t$, n), where in $CF2^x$ and $CF1^x$ each correspond 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 $CF2^x$. Thus, $CF2^x$ contains d values. The p-th entry of $CF2^x$ is equal to $\sum_{j=1}^{n} (x_{i_j}^p)^2$.

- For each dimension, the sum of the data values is maintained in $CF1^x$. Thus, $CF1^x$ contains d values. The p-th entry of $CF1^x$ is equal to $\sum_{j=1}^{n} x_{i_j}^p$.
- The sum of the squares of the time stamps $T_{i1} \ldots T_{in}$ is maintained in $CF2^t$.
- The sum of the time stamps $T_{i1} \ldots T_{in}$ is maintained in $CF1^t$.
- The number of data points is maintained in n.

We note that the above definition is actually a cluster feature vector. The summary information in it can be expressed in an additive way over the data points. In another words, as a new data point come and is added into a micro-cluster, the feature vector of the micro-cluster can be updated in an additive way.

Based on this definition, cluStream algorithm assumes that a total of *q* micro-clusters are maintained at any moment by the algorithm. The value of q is determined by the amount of main memory available in order to store the micro-clusters. Typically, the value of q is significantly greater than the number of clusters and smaller than the number of data points in a massive data stream. The following is the basic steps of the algorithm.

First, store *InitNumber* data points at very beginning of the data stream and offline create the initial q micro-clusters using k-means clustering algorithm.

When a new data point arrives, the micro-clusters are updated in order to reflect the change. It either needs to be absorbed by a closest existing micro-cluster, or it needs to be put in a cluster of its own if this data point does not lie within the *maximum boundary* of the nearest micro-cluster. In the later case, a new micro-cluster must be

created, and two of the old micro-clusters are merged together to limit the memory consumption.

The micro-clusters are stored at particular moments in the stream, which are referred as *snapshots*. Given the time-horizon *h* and the number of clusters *k* by the user, we can obtain a set of micro-clusters by $S(t_c) - S(t_c - h')$, where $S(t_c)$ is the snapshot at a current clock time t_c, $S(t_c\text{-}h')$ is the snapshot just before the time $t_c - h$, and h' is within a pre-specified tolerance.

At last, *k* clusters are determined by using a modification of a k-means algorithm.

This algorithm gives the user the flexibility to explore stream clusters over different time horizons. However, it also has the following shortages:

- Each new data is either absorbed in an existing micro-cluster or put into a new cluster of its own. After this, it will belong to this micro-cluster forever and won't be reassigned to other micro-cluster, even though the data distribution and micro-clusters' shape may change later.
- Although this clustering algorithm can be modified to discovering outlier clusters with the same definition and outlier degree measurement in our approach, but its original task is to find clusters, the outliers are typically ignored or tolerated in the clustering process for producing meaningful clusters, which prevents giving good results on outlier detection. E.g. outliers may be assigned to big clusters because of the above first shortage, merging clusters and k-means clustering in the initial phase.

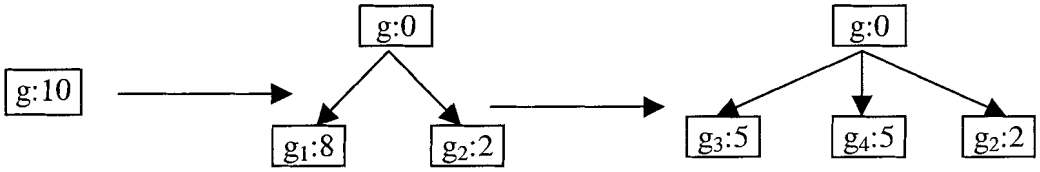
- It is a two-phase algorithm. The statistical summary are collected and stored in hard disk in first phase. Then it will do offline clustering and evolution analysis. From this point, it is not a real-time online algorithm.

2.1.2 Grid-based Clustering

Park and Lee [2] proposed a statistical grid-based approach to cluster data elements of a data stream. Initially, the data space is partitioned into a set of mutually exclusive equal-size initial cells. When the support of a cell becomes greater or equal to a predefined split support Ssplt, one dimension of the data space is chosen based on its distribution statistics and the cell is dynamically divided into two mutually exclusive intermediate cells, with respect to the selected dividing dimension. The distribution statistics of each divided cell are estimated. Similarly, when an intermediate cell itself becomes dense, it is partitioned by the same way. Eventually, a dense region of each initial cell is recursively partitioned until it becomes the smallest cell called a unit cell. A cluster in a data stream is a group of adjacent dense unit cells. Three partition methods are proposed: μ-partition (based on the average value), δ-partition (based on the standard deviation) and hybrid-partition (select the more effective one).

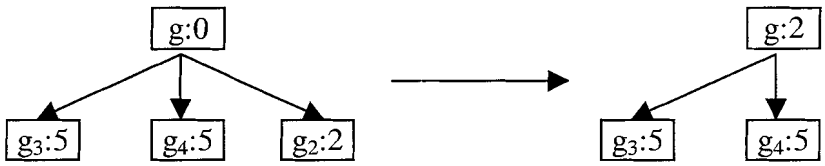
During the partition, it only maintains two-level hierarchical structure. When partitioning an initial cell g into g_1 and g_2, g_1 and g_2 are naturally the children of g. But when an intermediate cell g_1 is divided into g_3 and g_4, g_1 is replaced by g_3 and g_4. Figure 2.1 shows this example with $S_{splt} = 10$.

Figure 2.1 Partitioning Example



In this structure, it is very easy to prune an intermediate or unit cells whose support is less than a *predefined pruning support* S_{prn}. The cell is removed from the second level and its distribution statistics are returned back to its parent initial cell. The statistics of the parent cell is updated accordingly. Figure 2.2 shows a simple pruning example based on Figure 2.1 with S_{prn}= 3.

Figure 2.2 Pruning Example



This approach can also adaptively adjust the memory usage by resizing the size λ of a unit cell dynamically. Given the value of λ, if the confined memory space is full, all unit cells are pruned and their distribution statistics are added to their parent initial cells respectively. Similarly, those intermediate cells whose interval size in at least one dimension is less than 2λ are pruned by the same way. Subsequently, the value of λ is doubled and the normal operations of the proposed algorithm are resumed. Since the value of λ is doubled, the memory requirement of the proposed method is reduced while its clustering accuracy is degraded.

This clustering algorithm can also be applied to discovering outlier clusters with the same outlier definition and outlier degree measurement in our approach, but its top-down partition property prevent giving a good results on outlier detection. In top-down partition, the statistic summary can only be estimated. In addition, the distance between two clusters can only be estimated too, since sparse outlier clusters may lie in a very big rectangle.

2.2 Related Work on Outlier Detection

2.2.1 Distance-based approach

Knorr and Ng [6] proposed a distance-based approach to detect outliers in large static datasets. A distance-based outlier in a dataset T is a data object with at lease *p%* of the objects in T having a distance of more than D away from it. This definition is formally defined as follows [6]:

An object O in a dataset T is a DB(p, D)-outlier if at least fraction p of the objects in T lies greater than distance D from O.

This definition is suitable for situations where the observed distribution does not fit any standard distribution and relies on the computation of distance values.

For any dimensionality k>2, Knorr also presented a Nested-Loop (NL) algorithm for finding all outliers. The algorithm is shown in Figure 2.3. Assuming a total buffer size of B% of the dataset size, the algorithm divides the entire buffer space into two parts called the *first* and *second* array. It reads the data into the arrays, and directly computes pair-wised distances. For each object *t* in the first array, a count of its *D-neighbours* is maintained. Counting stops for a particular tuple whenever the number of *D-neighbours*

exceeds M. The run time complexity is $O(kN^2)$, where k and N are the dimensionality and size of the dataset.

Figure 2.3 Pseudo-Code for NL algorithm

Algorithm NL

1. Fill the first array (of size $\frac{B}{2}$% of the dataset) with a block of tuples from T.
2. For each tuple t_i in the first array, do:
   a. $count_i \leftarrow 0$
   b. For each tuple t_j in the first array, if $dist(t_i, t_j) \leq D$:
      Increment $count_i$ by 1. If $count_i > M$, mark t_i as a non-outlier and proceed to next t_i.
3. While blocks remain to be compared to the first array, do:
   a. Fill the second array with another block (but save a block which has never served as the first array, for last).
   b. For each unmarked tuple t_i in the first array do:
      For each tuple t_j in the second array, if $dist(t_i, t_j) \leq D$:
      Increment $count_i$ by 1. If $count_i > M$, mark t_i as a non-outlier and proceed to next t_i.
4. For each unmarked tuple t_i in the first array, report t_i as an outlier.
5. If the second array has served as the first array anytime before, stop; otherwise, swap the names of the first and second arrays and goto step 2.

This outlier definition is further extended in [7] based on the distance of a point from its k^{th} nearest neighbour. All the data objects are ranked by the distance to their k^{th} nearest neighbour, and the top-k data objects are identified as outliers. Alternatively, in the algorithm proposed by Angiuli and Pizzuti [9], each data point is ranked by the sum of distances from its k nearest neighbours.

However, for each data object, all these algorithms need a search for its neighbours, which is infeasible in stream environment. Even though some clustering algorithm over data streams can be used to approximate the neighbour search and computation, the neighbours of each data point may change over time and some data point not in the set of top-k outliers may become one of them later, but it has been thrown away since the memory can not keep all the data points.

Moreover, the above outlier definition captures only certain kinds of outliers. Because the definition takes a global view of the dataset, these outliers can be viewed as "global" outliers [8]. However, for many real-world datasets, such as data streams, which exhibit a more complex structure. Distance-based outliers are meaningful and adequate under certain conditions, but not satisfactory for the general case when clusters of different densities exist. This shortcoming is illustrated in details in [8].

2.2.2 Local outlier (LOF)

To avoid the shortcomings in distance-based outlier definition, Breunig [8] introduced the concept of "local outliers", which are the objects outlying relative to their local neighbourhoods, particularly with respect to the densities of the neighbourhoods. The outlier rank of a data object is determined by "local outlier factor (LOF)". Before introduce LOF, some notations are defined as follows.

k-distance of an object p [8]: For any positive integer k, the k-distance of object p, denoted as *k-distance(p)*, is defined as the distance $d(p,o)$ between p and an object $o \in D$ such that:

(i) for at least k objects $o' \in D \setminus \{p\}$ it holds that $d(p,o') \leq d(p,o)$, and

(ii) for at most k-1 objects $o' \in D \setminus \{p\}$ it holds that $d(p,o') < d(p,o)$.

k-distance neighbourhood of an object *p* [8]: Given the *k*-distance of *p*, the *k-distance neighbourhood* of *p* contains every object whose distance from *p* is not greater than the *k*-distance, i.e. $N_{k\text{-}distance(p)}(p) = \{ q \in D\setminus\{p\} \mid d(p, q) \leq k\text{-}distance(p) \}$. These objects *q* are called the *k*-nearest neighbors of p.

reachability distance of an object p w.r.t. object o [8]: Let *k* be a natural number. The reachability distance of object *p* with respect to object *o* is defined as

$$reach\text{-}dist_k(p, o) = \max \{ k\text{-distance}(o), d(p, o) \}.$$

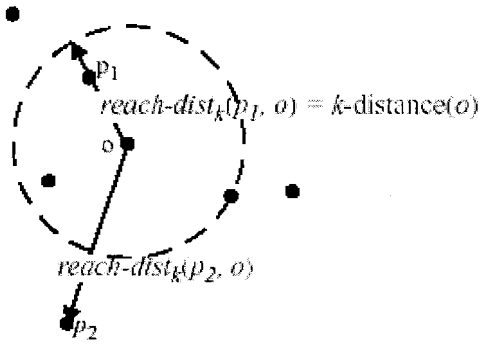
local reachability density of an object p [8]: The *local reachability density* of *p* is defined as

$$lrd_{MinPts}(p) = 1 \Big/ \left(\frac{\sum_{o \in N_{MinPts}(p)} reach\text{-}dist_{MinPts}(p, o)}{|N_{MinPts}(p)|} \right),$$

where *MinPts* is the minimum number of objects.

Figure 2.4 illustrates the idea of reachability distance with k=4. Intuitively, the local reachability density of an object *p* is the inverse of the average reachability distance based on the *MinPts*-nearest neighbours of *p*.

Figure 2.4 ***reach-dist(p_1, o)*** **and** ***reach-dist(p_2, o)*****, for** $k = 4$



Based on above definitions, **Local Outlier Factor (LOF)** [8] of an object p is defined as

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(p)} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{\left|N_{MinPts}(p)\right|}$$

The outlier factor of object p captures the degree to which we call p an outlier. It is the average of the ratio of the local reachability density of p and those of p's *MinPts*-nearest neighbours. It is easy to see that the lower p's local reachability density is, and the higher the local reachability densities of p's *MinPts*-nearest neighbours are, the higher is the LOF value of p.

Although the concept of local outliers is a useful one, the computation of LOF values for every data objects requires a large number of k-nearest neighbours searches and can be computationally expensive. This problem makes it infeasible in data stream environment.

2.2.3 Cluster-based approach

To satisfy the requirements in the application of intrusion detection, Eskin, et al. [14] investigated the effectiveness of a cluster-based algorithm to detect outliers in feature spaces.

The goal of this approach is to compute how many points are "near" each point in the feature space. For any pair of points x_1 and x_2, they are considered "near" each other if their distance is less than or equal to a fixed width w. Thus the number of points that are within w of a point x is defined as follows:

$$N(x) = |\{ s | d(x, s) \leq w \}|$$

The naïve computation of N(x) for all the points has a complexity of $O(n^2)$ where n is the size of the dataset. However, since the task is to identify the outliers, this computation can be approximated. This approximation is implemented as follows.

It first performs fixed-width clustering over the points with a radius of w. The first point is the center of the first cluster. Given a fixed width w, for every subsequent point, if it is within w of a cluster center, it is added to that cluster. Otherwise it is the center of a new cluster.

Let $N(c)$ be the number of points in a cluster c. For each point x, we approximate $N(x)$ by $N(c)$, where c is the cluster that contains x. For points in very dense areas where there is a lot of overlap between clusters, this approximation is an inaccurate estimate. However, for the outliers that lie in sparse regions, $N(c)$ is an accurate approximation of $N(x)$. Since we are only interested in outliers, the points in the dense regions will be higher than the density threshold anyway. Thus the approximation is reasonable.

After clustering the data points, the clusters are sorted by the size. The points in the small clusters are labelled as outliers.

This efficient approximation algorithm is able to process significantly large datasets, because it needs not to perform a pair-wise comparison of points and requires only one pass through the data. This property makes it appropriate and practical in stream environment.

However, this algorithm ranks the clusters only by their size. This simple outlier degree measurement may result in mislabelling lots of normal data points as outliers. For example, if the normal data points lie in the edge of a big cluster that has a dense center and sparse edge, they are outside of w and in a sparse area. Thus they may be incorrectly regarded as outliers.

2.3 A survey of network intrusion detection

Intrusion detection systems (IDS) automate the detection of security violations through computer processing of system audit information. The approaches are usually classified into two categories: Rule-Based Intrusion Detection (RBID) and Statistical-Based Intrusion Detection (SBID). The nature of training data sets of each category is different, but expensive to produce in either case.

Rule-Based Intrusion Detection (RBID) seeks to identify intrusion attempts by matching audit data with known patterns of intrusive behaviour. It compares data to known intrusive behaviour learned from training data. If the data matches the pattern of some known intrusion data, the observed data is considered intrusive. RBID systems rely

on codified rules obtained by training on a large set of data in which the attacks have been manually labelled [16]. This data is very expensive to produce because each piece of data must be labelled as either normal or some particular attack. Intrusions not represented in an RBID rule base will go undetected by these systems.

To help overcome this limitation, statistical methods have been employed to identify audit data that may *potentially* indicate intrusive or abusive behaviour. Known as Statistical-based Intrusion Detection (SBID) systems, these systems compare data to normal patterns learned from the training data. If the data deviates from normal behaviour, it is considered intrusive. These systems are popular because they are seen as a possible approach to detecting unknown or new attacks [17, 18, 19, 20]. Most of these systems require that the data used for training is purely normal and does not contain any attacks. This data can also be very expensive because the systems still require a very large amount of normal data.

In 1996, Forrest [17] introduced a novel and simple intrusion detection method based on learning the patterns of UNIX processes. This is a classic example of SBID systems. In this method, each UNIX process is represented by its *trace* – the ordered list of system calls used by that process from the beginning of its execution to the end. They gathered the normal traces and analyzed the "local (short) range ordering of system calls". They discovered that these local orderings "appears to be remarkably consistent, and this suggests a simple definition of normal behaviour".

The key idea is to build a "normal" database that contains all possible short sequences (e.g., of length 11) of system calls. The normal database is then used to examine the behaviour of a running program. If the total number or percentage of

abnormal sequences, which are those that can not be found in the normal database, is above a threshold value, then the current run is labelled as an outlier or intrusion.

2.4 Summary

Continue with Section 2.1 and Section 2.2, all proposed work on mining outliers is not appropriate for data stream model. Furthermore, existing proposals on mining data streams do not address the problem of outlier detection.

From Section 2.3, current Intrusion Detection Systems trained on data gathered from one environment may not perform well in some other environment. At the same time, the cost of generating data sets can be very expensive.

To address these issues, in this thesis, we propose an intrusion detection method that performs unsupervised online outliers detection over data streams.

CHAPTER 3:OUR APPROACH

This chapter provides an in-depth discussion of our approach for online outlier detection over data streams. In Section 3.1, a problem definition and the basic idea are given. Section 3.2 presents the grid-based clustering algorithm, and then Section 3.3 provides a new cluster-based outlier definition and a novel outlier degree measurement. Finally, Section 3.4 discusses the memory management strategies over stream environment and presents an efficient implementation.

3.1 Problem definition and basic idea

In this section, we formally define the problem of online outlier detection over data streams, and introduce our basic idea.

Problem Definition Given a data space in multiple dimensions A_1, ..., A_m with domains D_1,..., D_m respectively, let the data stream D be a sequence of data objects, where each data object $t \in D_1 \times \ldots \times D_m$. Our task is to online detect if a new coming data is an outlier.

The basic idea in our approach is as follows. It first performs an online clustering over the data stream, and then sort the clusters based on the outlier degree that measure how outlying a cluster is. We propose: the smaller a cluster is, the more outlying it is; the further a cluster is from the closest data cluster, the more outlying it is. For each new coming data, if it lies in the top-k outlying clusters, it is regarded as an outlier.

Here, the clustering algorithm can be chosen freely, as long as it can satisfy the following requirements:

- Appropriate for stream environment.
- Provide online statistic summary of each cluster.
- Produce good clustering results, e.g. not ignore small clusters

3.2 Grid-based clustering algorithm

In our approach, a grid-based clustering algorithm is used. To find clusters of similar data objects over a data stream, the distribution statistics of data objects in the data space of a data stream are carefully maintained. By keeping only the distribution statistics of data elements in a pre-partitioned grid-cell, the clusters of a data stream can be effectively found without maintaining the individual data elements physically. The maintaining of the statistics is very efficient and easy, but the partitions can not be dynamically changed.

To illustrate this clustering method, we begin with an example in a 2-dimension space. In Figure 3.1, the data space is partitioned into small *cells* according to users input. Here we simply use X, the width of the grid in each dimension, as the partition rule. Whenever a new data object arrives, it lies in a cell according to the values of its features. Then a cluster is formally defined as following:

Definition 1 Each non-empty cell is defined as a ***cluster***.

The advantage of defining each non-empty cell as a cluster is that it does not lose any small or sparse clusters. Most clustering algorithms only concentrate on providing

meaningful clustering results and often ignore small clusters. However, in the problem of outlier detection, the small clusters have more value than large ones, since most outliers lie in these small clusters. On the other side, outliers are not likely to lie in large data clusters. Thus to save memory, a group of adjacent large data clusters can be merged into one big cluster. We will discuss it in details later.

For each cluster, we maintain a statistic summary **S,** which is defined as follows:

Definition 2 The statistic summary **S** of a cluster is defined as a tuple ***<R, n, M>***, where:

- **R** is the set of cell rectangles included in this cluster.
- **n** is the support of this cluster.
- **M** is the set of means in all dimensions for the data objects in this cluster.

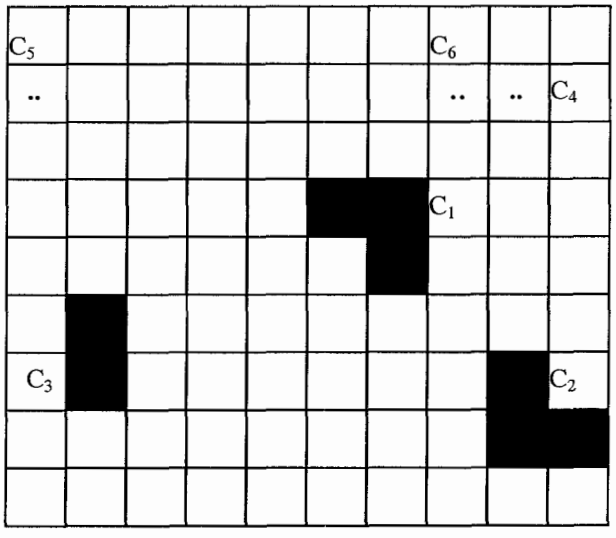
When a new data object arrives, it either needs to be absorbed by an existing cluster or needs to be put in a new cluster if it lies in an empty cell. The statistic summary S of the cluster is updated. Thus at any moment, a set of clusters can be constructed.

Figure 3.1 presents an example in 2-dimensions with 6 clusters (C_1, C_2, C_3, C_4, C_5, C_6) at some particular moment. C_1, C_2 and C_3 are constructed from several dense clusters. C_4, C_5 and C_6 are sparse clusters.

With this grid-based clustering algorithm we can easily find and merge the adjacent dense clusters by simply searching its neighbour cells in each dimension. However, the vector-based clustering algorithms must calculate pair-wise distance between clusters and find the closest clusters to merge. The distance computation is

expensive and may result in errors. Thus the grid-based clustering algorithm provides more accurate online summarization. But as the dimensions increase, the number of cells increases dramatically and the number of cluster may also increase. The space complexity of cells is $O(x^m)$, where x is the number of intervals in each dimension and m is the dimensionality; the space complexity of clusters is O(n), where n is the number of data objects in the data stream.

Figure 3.1 An example in 2-D space



3.3 Outlier definition and outlier degree

In this section, we propose a new cluster-based definition for outliers and a novel measure for outlier degree. We begin with the same example in Figure 3.1. It is very obvious that data objects in C_4, C_5 and C_6 are outliers. Intuitively we call data objects in C_4, C_5 and C_6 outliers because they are in small clusters and deviated from most data. Thus we will define outliers from the point of view of clusters and define the data objects that don't lie in *large data clusters* as outliers.

Definition 3 Let $C = \{C_1, C_2, \ldots, C_h\}$ is the set of clusters in the ordering of $|C_1| \geq |C_2| \geq \ldots \geq |C_h|$. Given two pre-specified parameters α and β, we find the minimal d as the boundary such that it can first satisfy condition (1) and then also satisfy condition (2):

$$\begin{cases} |C_1|+|C_2|+\ldots+|C_d| \geq \alpha * |D| & (1) \\ \dfrac{(|C_1|+\ldots+|C_d|)/d}{(|C_{d+1}|+\ldots+|C_h|)/(h-d)} \geq \beta & (2) \end{cases}$$

Then the set of large ***Date Cluster (DC)*** = $\{C_1, C_2, \ldots, C_d\}$, and the set of small ***Outlier Cluster (OC)*** = $\{ C_{d+1}, \ldots, C_h \}$. Thus all the data objects in OC are outliers.

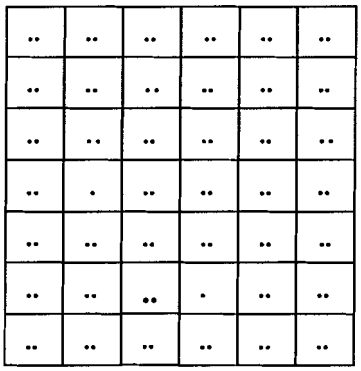
The above definition considers two heuristics:

- Outliers are just a small number of data objects, and most data objects are not outliers. It is our Assumption 1 and is represented by formula (1). E.g. $\alpha = 95\%$, then at least 95% of data objects are not outliers.
- The average size of clusters in DC should also be much bigger than that of clusters in OC. E.g. $\beta = 6$ then the average size of clusters in DC is at least 6 times of the average size of clusters in OC.

In Figure 3-1, according to Definition 3, DC = $\{C_1, C_2, C_3\}$ and OC = $\{C_4, C_5, C_6\}$. However, sometimes only condition (1) can be satisfied. Condition (2) is not satisfied since clusters have very similar support, and the boundary d can not be found by Definition 3. In this case, data points are evenly distributed in the data space, so it is

intuitively reasonable to claim that there are no outlier clusters and all clusters are data clusters, thus no outliers exist. Figure 3-2 illustrates the situation.

Figure 3.2 Boundary not exists



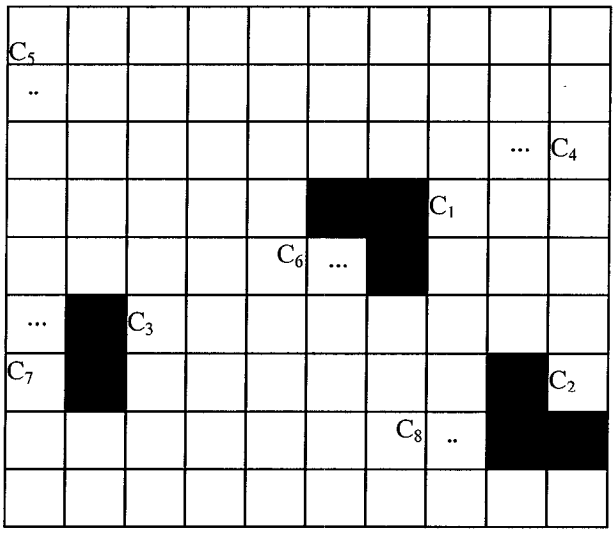
To describe an outlier cluster, it is desirable to have an outlier degree to measure how outlying it is from normal data clusters. Moreover, some outlier clusters in OC may be very close to big data clusters and only contain normal data objects. If all the data objects in OC are regarded as outliers, those normal data objects will be incorrectly labelled as outliers. To avoid this, we can simply only predict those data objects in the top-k outlier clusters as outliers, if an outlier degree measurement is defined.

Figure 3.3 illustrates this situation. There are 8 clusters (C_1, to C_8) at some particular moment. According to Definition 2, OC = {C_4, C_5, C_6, C_7, C_8} and DC = {C_1, C_2, C_3}. However, all the data objects in C_6, C_7 and C_8 are normal data, and only C_4 and C_5 are real outlier clusters. So if we have an outlier degree measurement and let k be 2, then C_4, C_5 can be identified as the top-2 outlier clusters and the detection is more accurate.

From the example in Figure 3.3, we observe that C5 is more outlying than C4, since it is smaller and further from normal data clusters. According to Assumption 2,

outliers are statistically different from normal data, the further from the closest data cluster, the more outlying they are. Thus the outlier degree of an outlier cluster should be determined by the size of its own cluster and the distance between itself and the closest data cluster. For any pair of clusters C_i and C_j, the distance between them can be approximate by the distance of two closest cells that belongs to C_i and C_j respectively.

Figure 3.3 Top-k outlier clusters



Definition 4 For any two clusters C_i and C_j, the **distance (C_i, C_j)** is defined as:

$$\textbf{distance } (C_i, C_j) = \min \{\text{distance}(r_s, r_t)\},$$

where r_s is a cell in C_i and r_t is a cell in C_j.

Based on above observation and definitions, the Outlier Degree of an outlier cluster is defined as follows:

Definition 5 Given the set of clusters $C = \{C_1, C_2, \ldots, C_k\}$ in the ordering of $|C_1| \geq |C_2| \geq \ldots \geq |C_k|$, and α, β, d, DC and OC based on Definition 2. For each cluster $C_i \in OC$, the **Outlier Degree (OD)** of C_i is defined as following:

$$OD(C_i) = \frac{\min\{\text{distance}(C_i, C_j)\}}{|C_i|} \qquad (4), \quad \text{where } C_j \in DC.$$

From Definition 5, the smaller a cluster is, the more outlying it is; the further a cluster is from the closest large cluster, the more outlying it is. We can normalize the distance between any pair of two clusters, then numerator of formula (4) is in [0, 1], and the denominator is in [1, ∞], consequently the OD value is in [0, 1].

Figure 3.4 shows an example when a new data t lies in a large data cluster and its OD value equals to 0. Figure 3.5 presents an example when a new data t is very far from the closest large data cluster and its OD value equals to 1.

Figure 3.4 An example of OD = 0

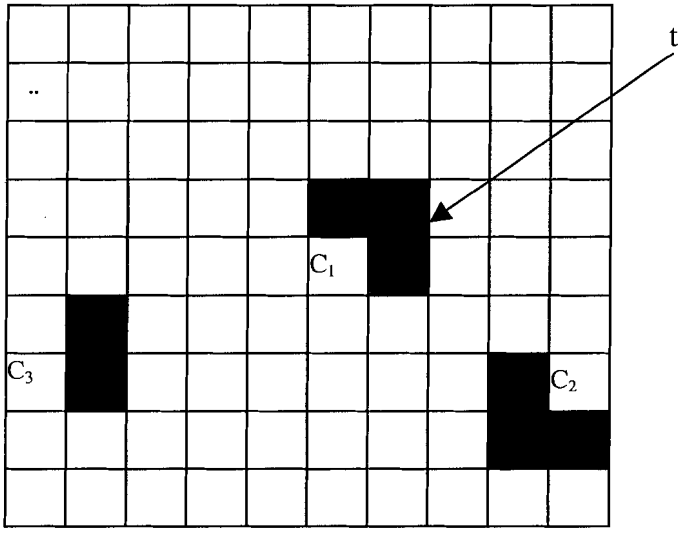
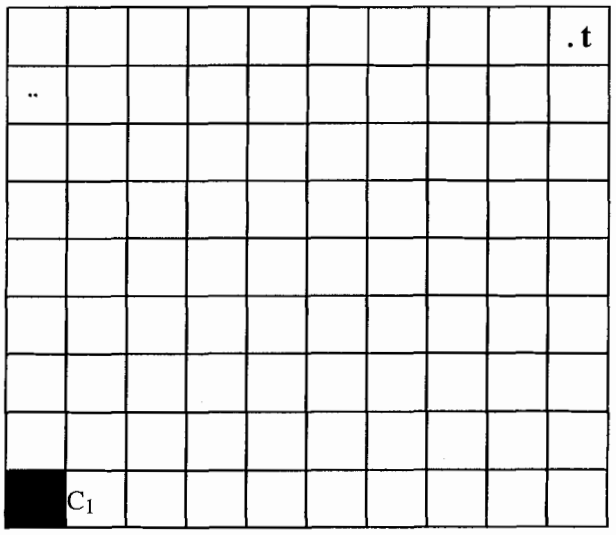


Figure 3.5 An example of OD = 1



3.4 Online outliers detection over data streams

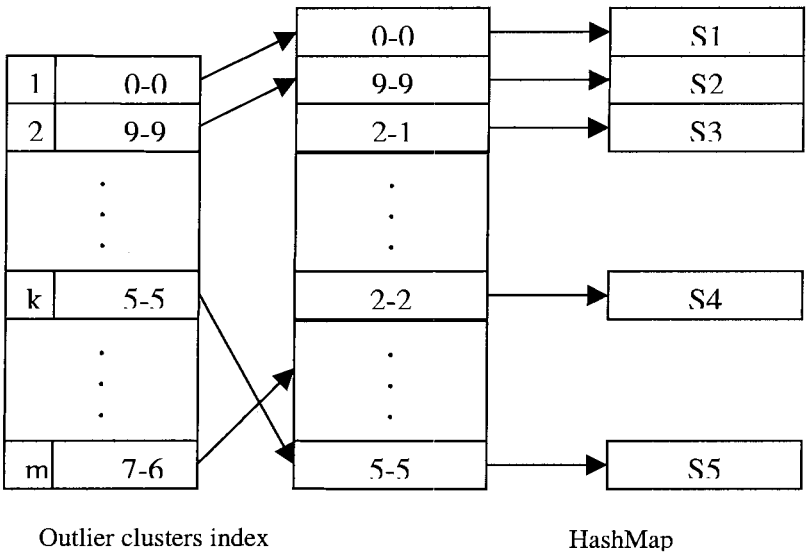
Given the outlier definition and outlier degree measurement, this section presents how to implement an efficient algorithm to online detect outliers in stream environment. Since our approach is cluster-based online outlier detection, we denote it as ***CBOD*** in our later discussion.

The idea of ***CBOD*** is to immediately identify an outlier in the real time by online building summaries of grid-based clusters and check if it is in the top-k outlier clusters.

The challenge is how to control the memory consumption, and what should be done if the memory is full. Because the data stream is infinite, there may be so many clusters that data structure for clusters' summary will eventually overload the memory. When memory is full, we can use the following two strategies:

- First, we can merge adjacent data clusters into one big data clusters. Since out task is to identify outliers that usually lie in outlier clusters, merging adjacent dense data clusters won't affect the detecting accuracy.
- Second, we can prune all the outlier clusters in OC or write them to hard disk for users' offline analysis in the future. The sacrifice of this strategy is that some initial normal data objects in new data clusters probably be regarded as outliers in a short of period after pruning, but it won't miss any real outliers.

Figure 3.6 An example of data structures without merging



To implement the grid, keep the online summary, we used a data structure called HashMap provided by Java Development Kit. HashMap is roughly equivalent to a hash table. It provides random access with a constant time by mapping a key object to a value object. In our algorithm, we use the grid coordinates as the key and the summary S of the

cluster as value object. With this structure, we only keep summaries for non-empty cells or clusters, and for each new coming data, we can directly find the owner cluster's summary or create a new one with the key. To keep track of all the outlier-clusters, we use an index of the keys sorted by their corresponding OD values. Figure 3.6 shows an example of the data structures for a 2-D data set.

To find and merge the adjacent data clusters, we can simply search its neighbour cells in each dimension. In the above example, (2, 2) is a data cluster, we check its 4 neighbours with 1-2, 3-2, 2-1, 2-3 as keys respectively and find that (2, 1) is an adjacent data cluster, then we merge them into one cluster. To prune all the outlier clusters, we can find their keys from the outlier clusters index and delete them in HashMap directly with those keys. The time complexity for merging and pruning is O(n), where n is the number of clusters, which in the worst case is the number of data objects. Figure 3.7 shows the data structures after the merging.

Figure 3.7 An example of data structures after merging

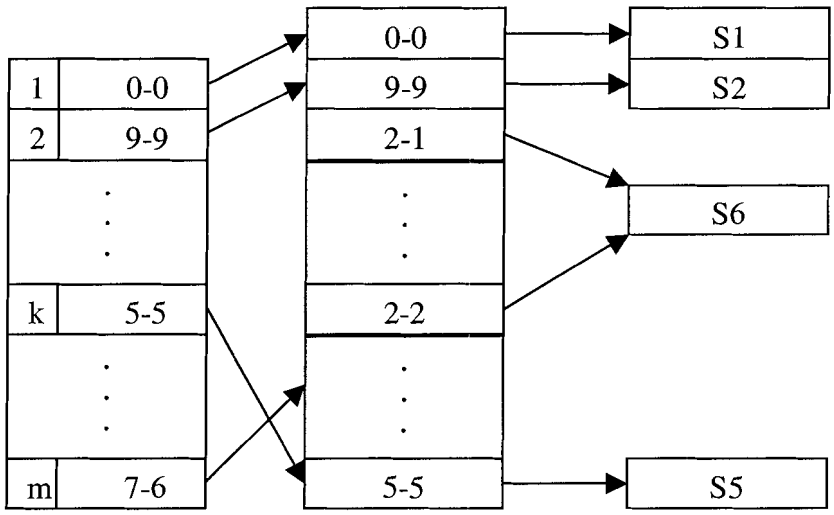


Figure 3.8 gives the pseudo code of CBOD. The input are the data stream D, the minimum percentage (α) of data objects in D that are normal, the minimum ratio (β) of data clusters' average size and outlier clusters' average size, the number (K) of most outlying clusters that users want to check during detection, and the rule to grid the data space, the width (X) of the grid in each dimension when partitioning data space.

Figure 3.8 The CBOD algorithm

Input: data stream D, α, β, K, and X

Methods:

1. Initialize Top-K-Outlier_Clusters[ ] = ∅.
2. **For** each new coming data object *t*,
3. **If** memory full, then merging adjacent data clusters and pruning all outlier clusters.
4. Discretize *t* according to input parameter X.
5. **If** t lies in an existing cluster, then assign *t* to this cluster.
6. **else** create a new cluster of its own.
7. update the summary S = <*R*, *n*, *M*> for that cluster
8. determine *DC* (*D*, α, β) and *OC*(*D*, α, β) according to Definition-3.
9. compute $OD(C_i)$ *for each* $C_i \in OC$ according to Definition-5.
10. update Top-K-Outlier_Clusters[ ].
11. **if**($t \in$ Top-K-Outlier_Clusters[ ]), then predict t as an outlier.
12. **else** predict t as normal data.

The method starts with initialize Top-K-Outlier_Clusters to empty in Line 1. Then it repeats Line 3 to Line 12 for each new coming data object. When a new data

object arrives, if the memory is already full it merges all adjacent dense data clusters and prune all the outlier clusters. First, the new data is discretized according to input parameter X and its owner grid is determined. If it lies in an existing cluster, it is assigned to this cluster in Line 5, otherwise create a new cluster of its own in Line 6. Then update the summary for the cluster and determine DC and OC according to the input parameters (D, α, β) and the outlier definition in Line 7, 8. Continuously calculate the outlier degree OD for each outlier cluster in OC and put the top-k outlier clusters in Top-K-Outlier_Clusters in Line 9, 10. Finally, if the cluster containing the new data object is in Top-K-Outlier_Clusters, it is regarded as an outlier in Line 11, otherwise it is regarded as a normal data in Line 12.

As we discussed above, the following are the time complexity of the main operations.

- Assign a new data to the corresponding cluster or create a new cluster: O(1)
- Merging data clusters and pruning outlier clusters (optional): O(n)
- Sorting clusters by size when determining DC and OC: O(1), because only the rank of this owner cluster of the new data and its neighbours in the sorted cluster list can change.

Thus the total run time complexity of CBOD is O(n), where n is the number of clusters and equals to the number of data objects in the worst case. However, when the worst case occurs, it also means that no outliers exist and most clusters can be merged as data clusters. Thus in average case n is much smaller than the number of data points and is usually at hundred or thousand level. In addition, the merging and pruning are required

only when memory is full, which is very infrequent. Thus, in most cases the run time complexity is basically constant, which can definitely satisfy the online requirement.

CHAPTER 4:EVALUATION

To evaluate the effectiveness of our algorithm, we conduct extensive experiments. In this Chapter, we present data set, experiment design, results and discussion.

4.1 Data Set

The data used for testing is the KDD Cup 1999 data mining competition data set (KDD 1999). It originated from the 1998 DARPA Intrusion Detection Evaluation Program managed by MIT Lincoln Labs. Lincoln Labs simulated a military LAN and peppered it with multiple attacks over a nine-week period.

The raw training data is from the first seven weeks of network traffic, but it contains some noise data and some of them have no or wrong class labels. In addition, since ***CBOD*** is unsupervised and there is no training phase, it is not necessary to use this training data set. The testing data consists of the last two weeks of traffic with around 300,000 connections, and every record has a corrected class label. Each record consisted of 41 features, and a class label that is either "normal" or one of the 37 attack types. These 37 attack types fall into four main categories:

- DOS: denial-of-service
- R2L: unauthorized access from a remote machine
- U2R: unauthorized access to local super-user (root) privileges
- Probing: surveillance and other probing

This data set is popularly used in data stream domain. In our experiment, we did not use all the data from the 1999 KDD Cup testing data set. Since it was generated for a supervised learning competition, the data set contains a high percentage of attack traffic (91%). We need to filter out most of the attacks however to fit Assumption 1. This is achieved by randomly sampling some attacks. As a result, the testing data set only contains a very small number of attacks belonging to 4 main categories respectively. Totally it contains around 61125 records, including 60269 normal connections and 856 attacks. The data distribution is given in Table 4.1.

Table 4.1 Data distribution

Data	Number	Percentage
Normal	60269	98%
Dos	47	2%
Probing	118	
U2R	79	
R2L	612	

The data set has 41 features. In our experiment, we use 4 features:

- connection duration
- number of data bytes from source to destination
- percentage of connections to the same destination that have 'REJ' error
- percentage of connections to the same destination with the same service that have 'REJ' error

Another unsupervised network intrusion detection approach [14] uses the same data set, obtains a similar distribution after filtering many attacks, and also uses 4 features.

4.2 Experiment design

To evaluate our approach, we use detection rate and false positive rate as the performance measurement. They are defined based on a confusion matrix as shown in Table 4.2.

Table 4.2 Confusion Matrix

	Actual positive (outliers)	Actual negative (normal)
Predict as Positive (Outlier)	TP (true positive)	FP (false positive)
Predict as Negative (Normal)	FN (false negative)	TN (true negative)

$$\textbf{Detection Rate} = \frac{TP}{TP + FN}$$

$$\textbf{False Positive Rate} = \frac{FP}{TN + FP}$$

Detection Rate and False Positive Rate are good indicators of performance [14], since they measure what percentage of intrusions the method is able to detect and how many incorrect classifications it makes in the process. We calculate these values over the labelled data to measure the performance.

To determine the parameter values used by ***CBOD***, we conduct extensive experiments based on different values and combinations of α, β, K, X. Figure 4.1 to Figure 4.4 show the impact of these parameters. We choose one group of values having the best result. The parameter values finally used by our algorithm are shown in Table 4.3.

To simulate the memory full situation in our experiments, we assume the memory can only contains summary for 60, 100, 150, 200 and all clusters respectively. Thus we can evaluate the performance of ***CBOD*** in different memory size. If the number of clusters exceeds the limit, the memory is considered as full and merging and pruning strategies will be performed.

Table 4.3 Values of parameters in our algorithm

Parameter	Value
α	98%
β	6
k	60
X	250 - for all integer dimensions 0.1 - for all real number dimensions

Figure 4.1 Comparison of different α (Alpha) values

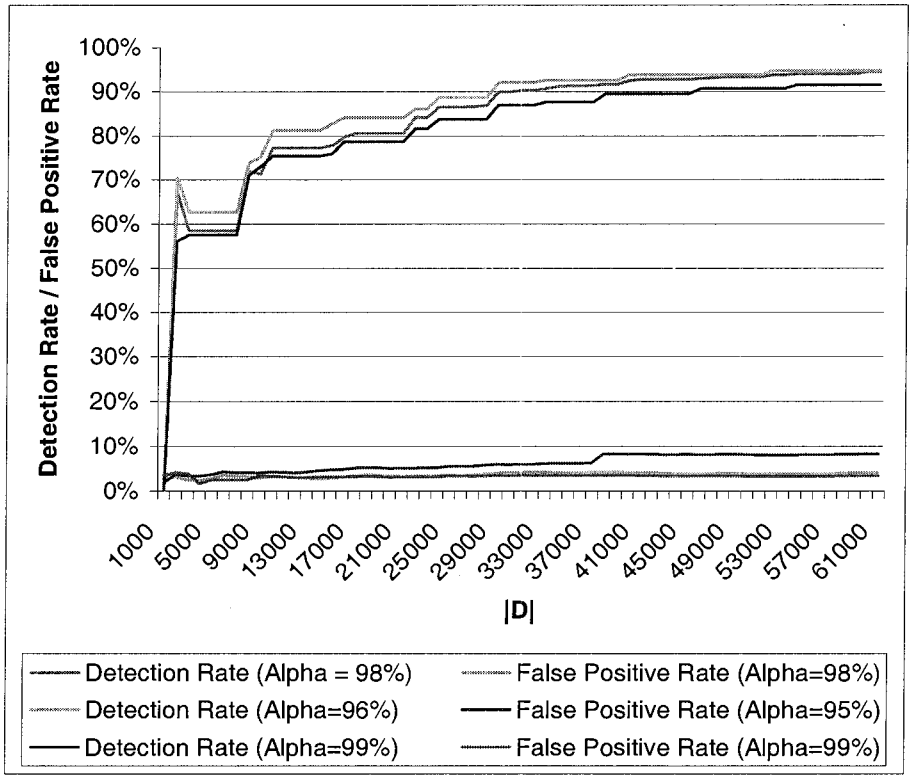


Figure 4.2 Comparison of different β (Beta) values

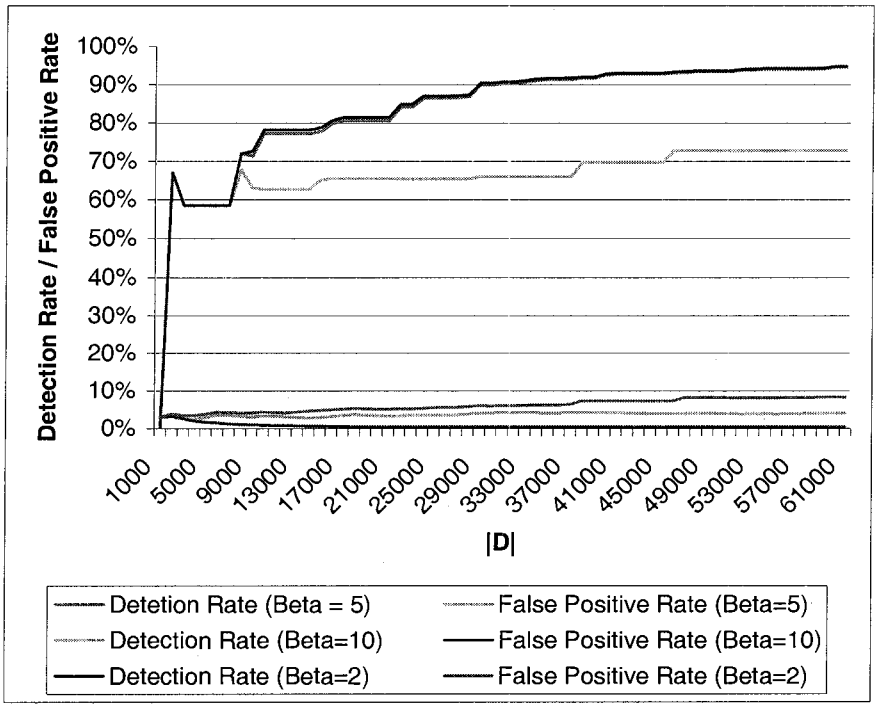


Figure 4.3 Comparison of different K values

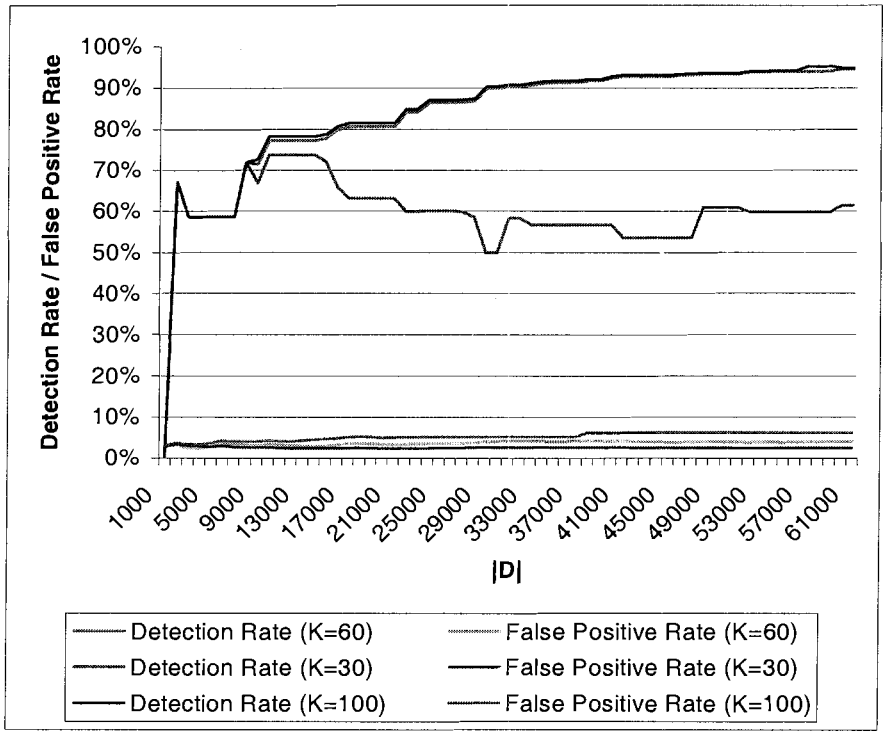
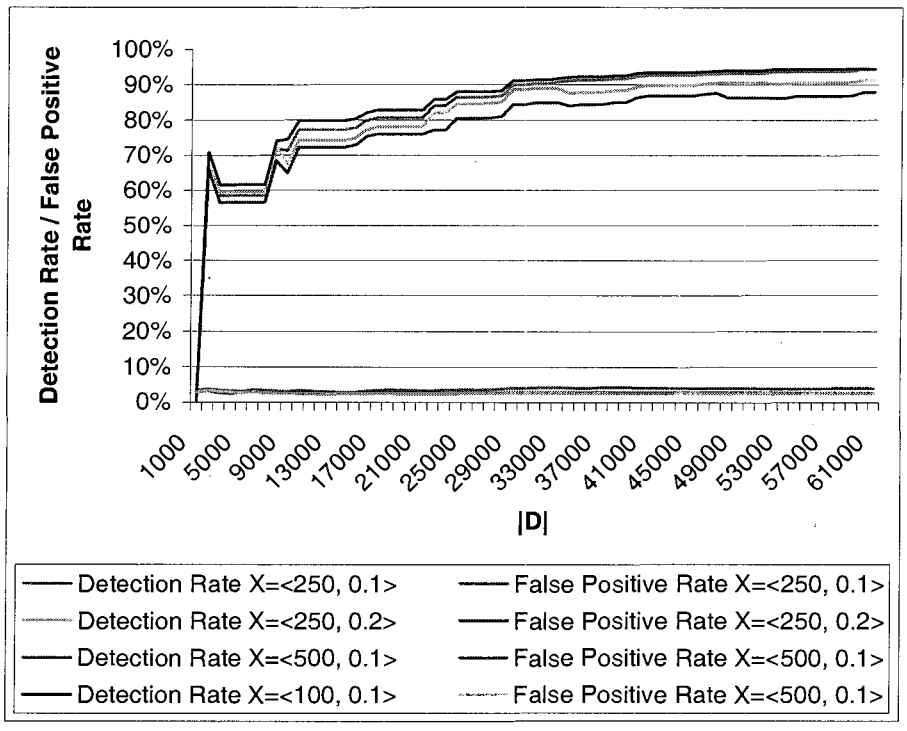


Figure 4.4 Comparision of different X values



To further evaluate the performance of ***CBOD***, we also conduct extensive experiments on two other approaches for comparison.

One comparison partner is Clustream [1], which is a very popular and import clustering algorithm over the data stream. Instead of our own clustering method, we use this algorithm to provides online summary of clusters in data stream, and then still apply our outlier definition and detection algorithm on the top. It is a natural and good way to evaluate the performance of our own clustering algorithm. Clustream needs to perform a k-means clustering for the first *InitNumber* data objects in the initial phase. Since the value of k is determined by the size of main memory and it should be as large as possible, k is naturally set to 60, 100, 150, 200 and 250 respectively. Thus we can compare it with ***CBOD*** in different memory size. We set InitNumber equals to 5000 in our experiments.

The second partner is "Unsupervised Anomaly Detection" [14] for detecting network intrusion. We denote it as UAD in our later discussion. This algorithm has some similarity with CBOD. It is also an unsupervised cluster-based outlier detection method, its experiment uses the same dataset and it is for the same network intrusion detection problem. Thus comparison with this method is another good indicator. UAD need a fixed width w as input parameter. Since we can compute the radius of cells in ***CBOD***, we try some different values around this radius value and use the one having the best result for comparison.

To verify the performance of detecting outliers only based on top-k outlier clusters, for CBOD and Clustream, we also conduct the experiments to detect outliers based on all the outlier clusters as a comparison.

4.3 Results

We calculate and gather Detection Rate and False Positive Rate periodically over the data stream. Figure 4.5 to Figure 4.14 show the experiment results. The content in each figure is shown in Table 4.4

Table 4.4 Result Matrix

	Memory Size (max. number of clusters it can keep)				
	60	**100**	**150**	**200**	**ALL**
Comparison of Detection Rate	Figure 4.5	Figure 4.7	Figure 4.9	Figure 4.11	Figure 4.13
Comparison of False Positive Rate	Figure 4.6	Figure 4.8	Figure 4.10	Figure 4.12	Figure 4.14

Figure 4.5 Comparison of Detection Rate (MM = 60)

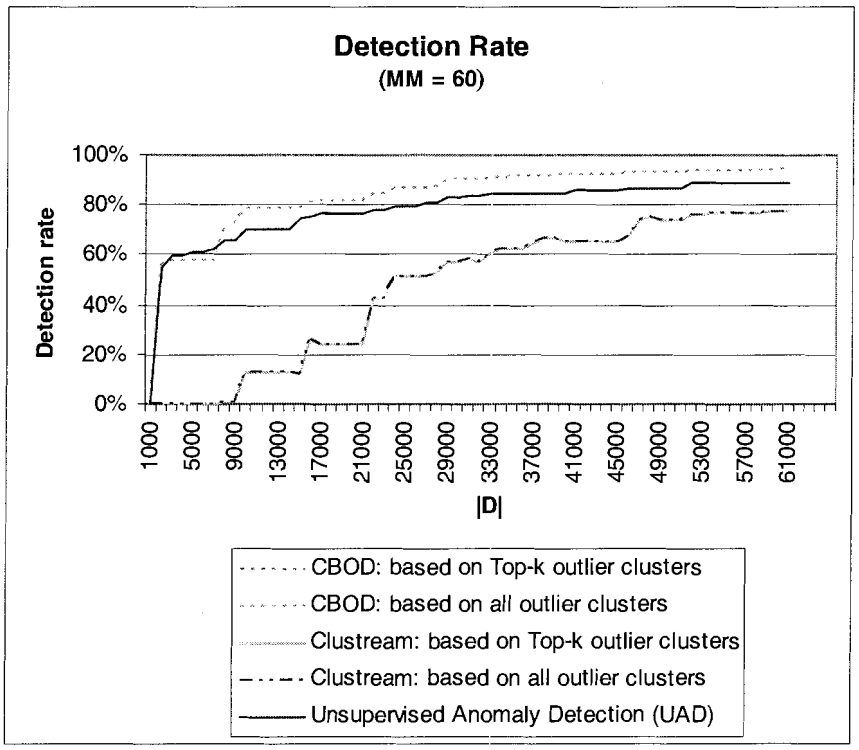


Figure 4.6 Comparison of False Positive Rate (MM = 60)

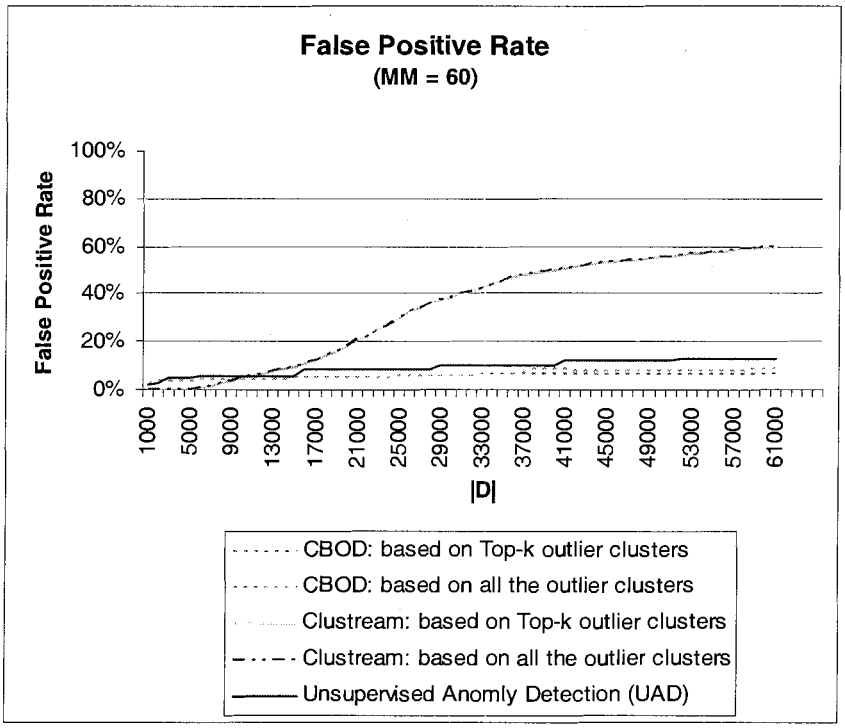


Figure 4.7 Comparison of Detection Rate (MM = 100)

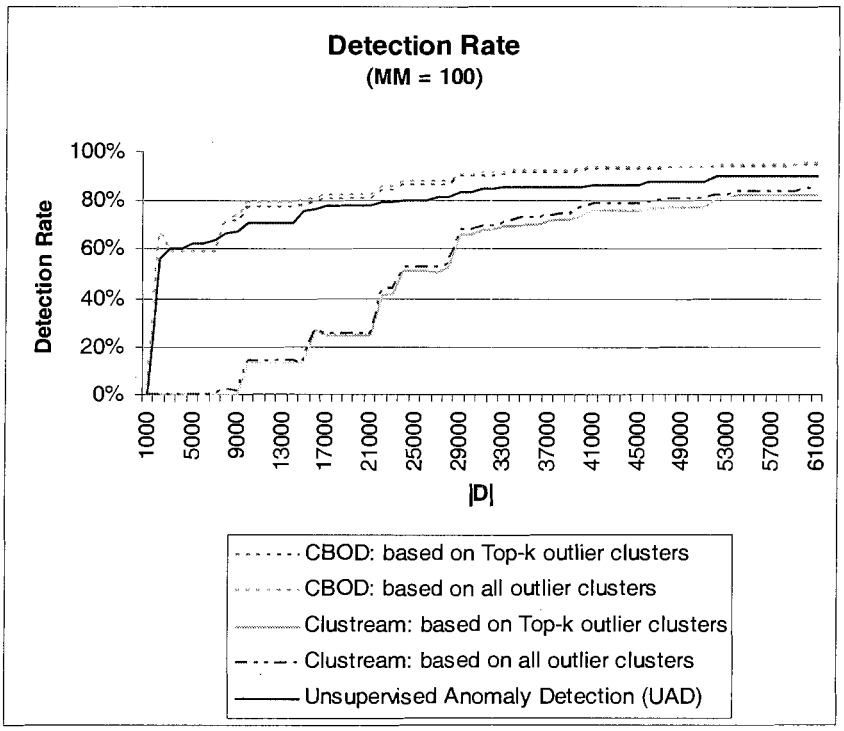


Figure 4.8 Comparison of False Positive Rate (MM= 100)

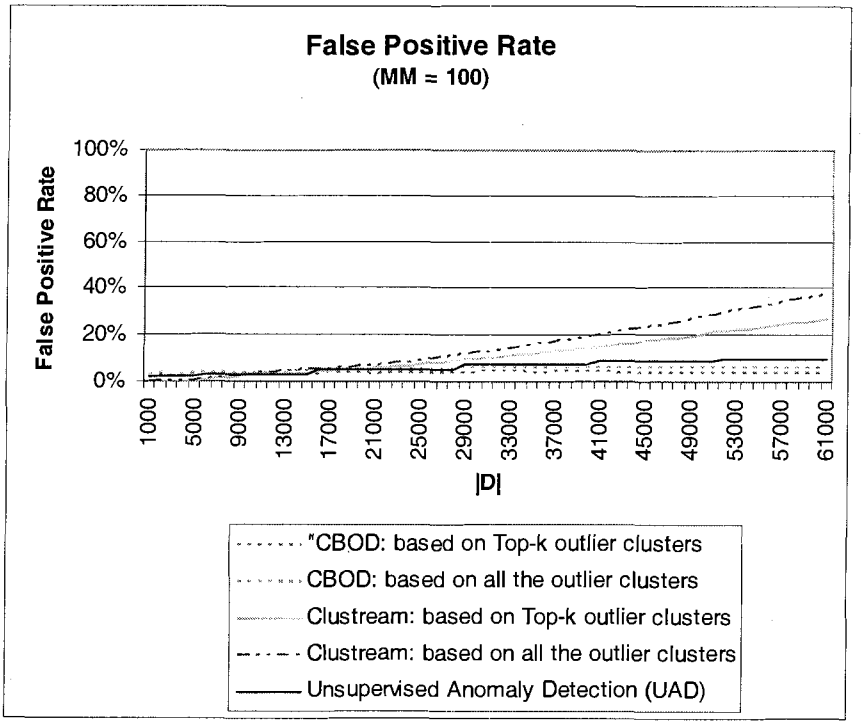


Figure 4.9 Comparison of Detection Rate (MM = 150)

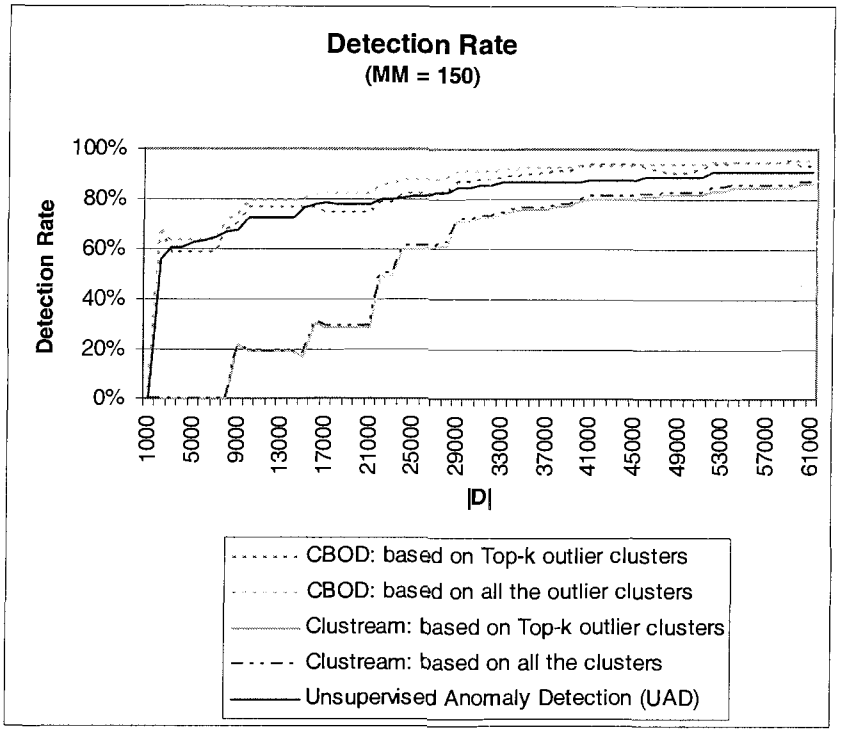


Figure 4.10 Comparison of False Positive Rate (MM = 150)

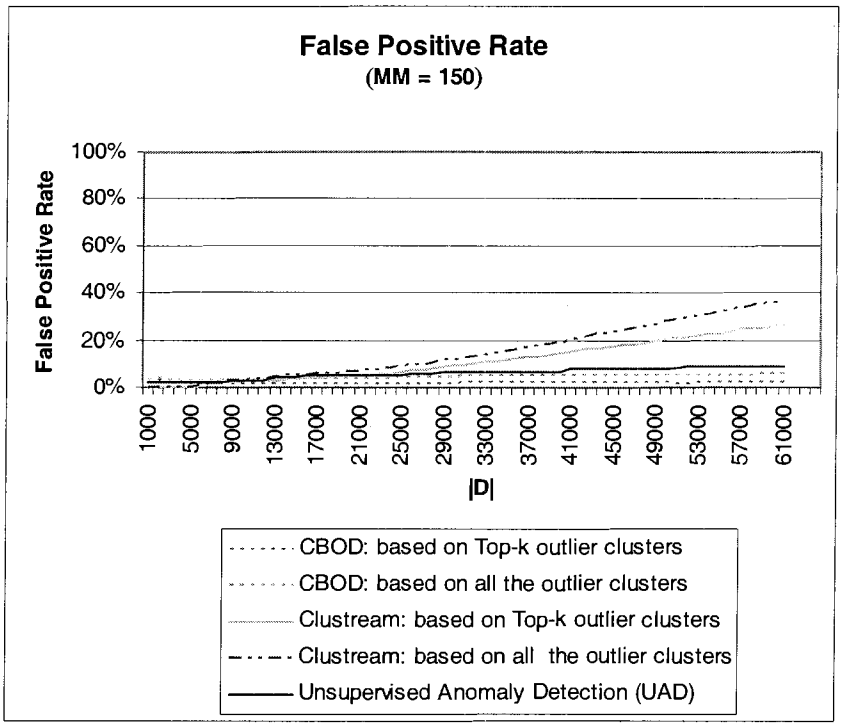


Figure 4.11 Comparison of Detection Rate (MM = 200)

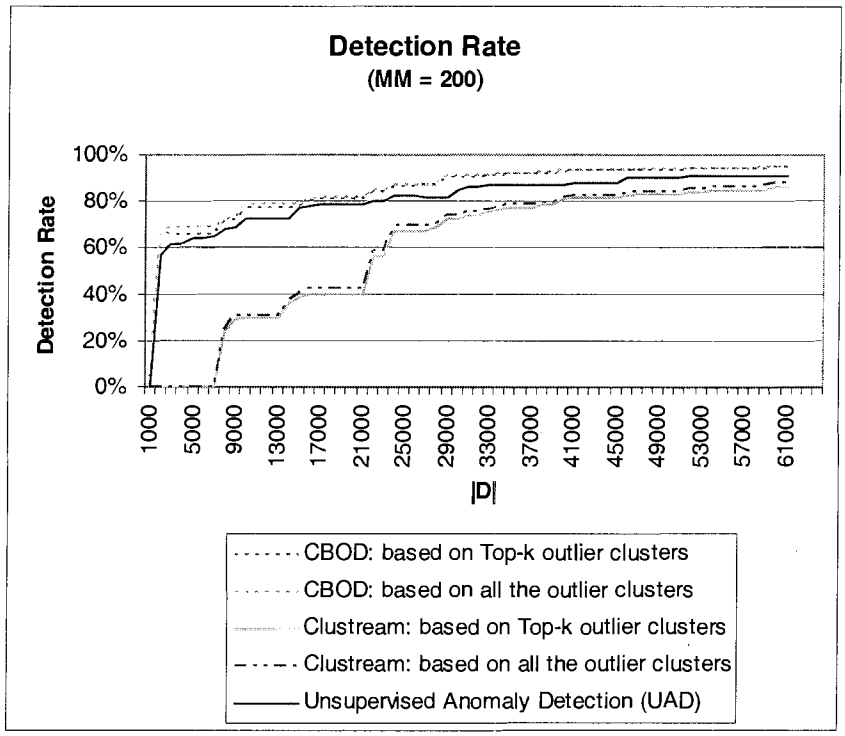


Figure 4.12 Comparison of False Positive Rate (MM = 200)

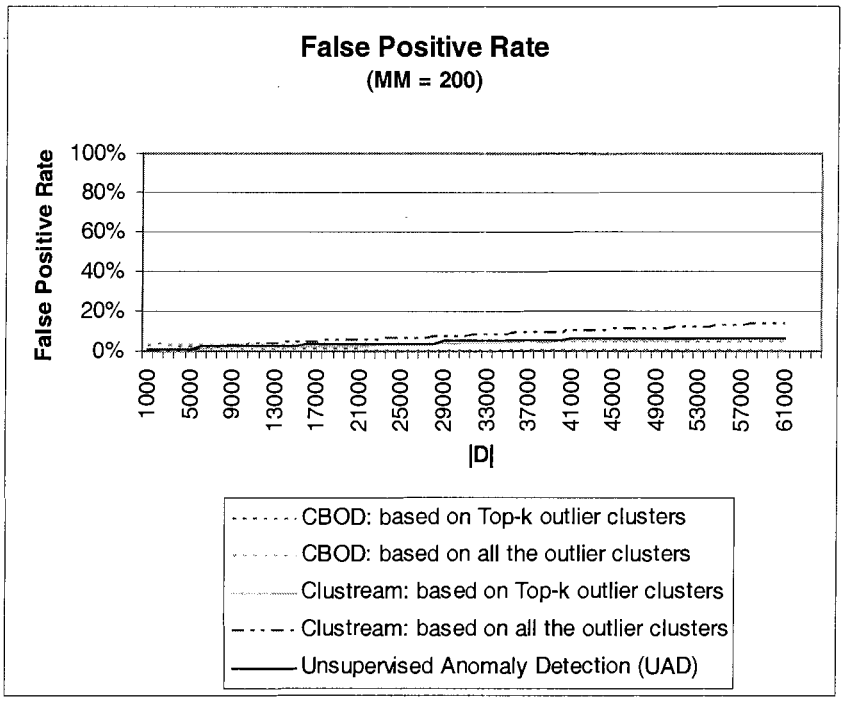


Figure 4.13 Comparison of Detection Rate (MM = all)

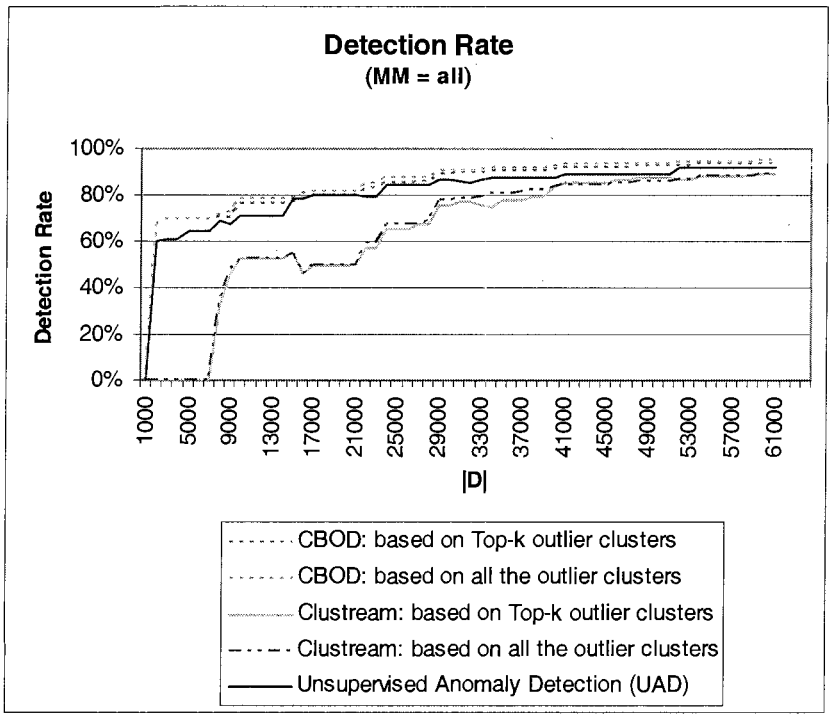
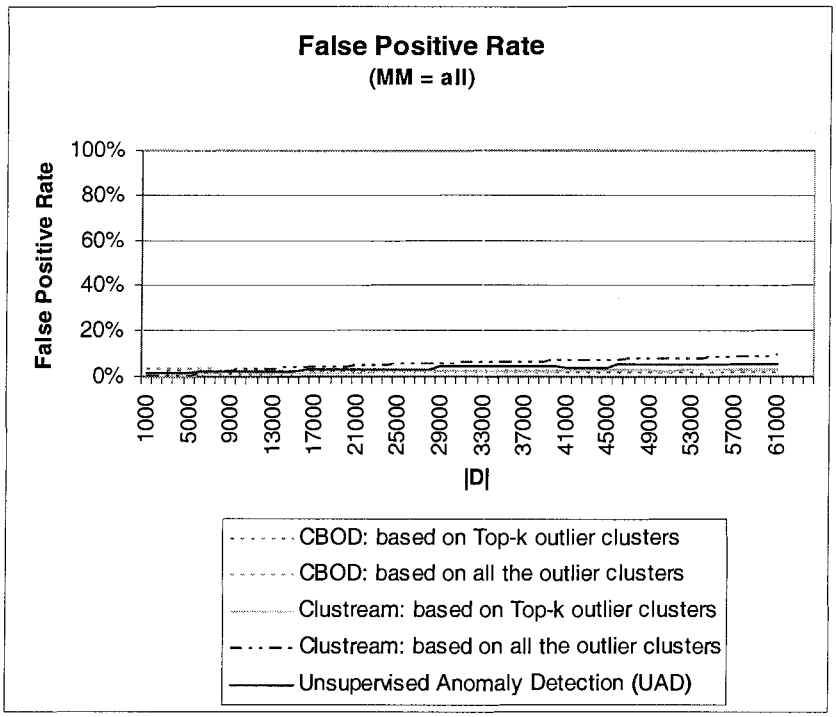


Figure 4.14 Comparison of False Positive Rate (MM = all)



4.4 Discussion

The experiment results show that our algorithm has a better detection rate than Clustream and UAD at different memory sizes. At the beginning the detection rate of ***CBOD*** is zero, since no clusters information exists at the beginning, but it is increased very quickly. At the end of the data stream, detection rate reaches 95%.

The detection rate of UAD is lower than that of our method because it uses a simple clustering algorithm and its outlier degree measurement only consider the size of clusters. Clustream has a worse detection rate, because it focuses on producing meaningful clusters and ignores some small clusters.

At the same time, ***CBOD*** also has a lower false positive rate than Clustream and UAD at different memory sizes. The number of misclassification is only about 4% of the

normal data objects. This is acceptable in the real application. However, Clustream and UAD have a much higher false positive rate, because of the similar reasons as above.

The figures also present that the detection rate based on top-k outlier clusters is very close to the detection rate based on all outlier clusters. However, the false positive rate using top-k outlier clusters is lower than that based on all outlier clusters. This demonstrates the correctness of our novel outlier degree measurement. In data stream environment, it requires a fast processing in the real time. If the algorithm is based on all the clusters, the computation cost is very expensive. Thus the method of detection only based on top-k outlier clusters will be much more efficient than detection using all the outlier clusters, and it also verify the effectiveness of our efficient algorithm.

The experiment results at different memory sizes also demonstrate the effectiveness of the pruning strategies in ***CBOD***. As memory size decreases, the detection rate does not change much. However, as memory size decreases, the false positive rate increases, but is still at an acceptable level. For example, when the memory size is only 60, ***CBOD*** can still achieve 93% detection rate with only 6% false positive rate.

Thus, the experiment results demonstrate that our algorithm works well in the data stream environment and outperforms other approaches.

CHAPTER 5:CONCLUSION AND FUTURE WORK

In this Chapter, we first summarize the thesis, and then discuss some interesting directions for future research.

5.1 Summary of the Thesis

Outlier detection is an important task in data mining research with numerous applications, including credit card fraud detection, discovery of criminal activities in electronic commerce, video surveillance, pharmaceutical research, and weather prediction. Recently, discovering outliers under data stream model have attracted attention for many emerging applications, such as network intrusion detection, sensor networks, earth observation, and stock analysis.

A data stream is an ordered sequence of objects that arrive continuously and must be processed online. And the space available to store information is supposed to be small relatively to the huge size of unbounded streaming data objects. Thus, the data mining algorithms on data streams are restricted to be able to fulfill their works with only one pass over data sets and limited resources. This is very challenging. Most previous studies on outlier detection can not satisfy the requirements in stream environment.

In this thesis, we propose a cluster-based online outlier detection algorithm (***CBOD***) to detect network intrusions. We use a grid-based clustering method to discover clusters and keep online summary over the data stream, and those small clusters far away from large data clusters are regarded as outlier clusters. We also proposed a

novel definition of outlier degree to measure how outlying each cluster is. When a new data arrives, it is considered as an outlier and intrusion if it lies in the top-k outlier clusters.

The following are the contributions of our approach.

- Propose a novel method for online outlier detection over data stream. Our approach can immediately predict if a data object is an outlier, when it arrives.
- Propose a novel definition for outlier*: cluster-based outlier,* which has great new intuition and numerous applications.
- Introduce a quantitative outlier degree measurement.
- Present an efficient and effective grid-based online summarization and clustering algorithm that successfully satisfies the requirement of run time complexity and memory consumption for data stream.
- Besides a binary detecting answer, provide more information or description about outliers, such as the outlier degree, the means of outlier clusters, etc.

The extensive experiments based on the KDD Cup 1999 data mining competition data set demonstrate the effectiveness of our approach.

5.2 Future Work

With the success of this method, it is interesting to re-examine and explore many related problems, extensions and applications. Some of them are listed here:

- *Discover bursts and outliers between bursts over data streams.* This is a different way to define outliers, which includes time dimension and

considers sparse data points between bursts as outliers. Our approach is for detecting network intrusion and does not include time dimension in this application. So discovering bursts and outliers between bursts in other applications worth some research.

- *Outlier detection based on subspace clustering over data stream.* How to work with high dimensionality is very important, because an outlier may be outlying only on some, but not on all, dimensions. This information can describe or explain why the identified outliers are exceptional. This is an interesting and quite challenging topic.

REFERENCE LIST

[1] Charu C. Aggarwal, Jiawei Han and Philip S. Yu, "A Framework for Clustering Evolving Data Streams", VLDB, 2003.

[2] Nam Hun Park and Won Suk Lee, "Statistical Grid-based Clustering over data streams", SIGMOD Record, 2004.

[3] Gurmeet Singh Manku and Rajeev Motwani, "Approximate Frequent Counts over Data Streams", VLDB, 2002.

[4] Jeffrey Xu Yu, Zhihong Cheng, Hongjun Lu and Aoying Zhou, "False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams", VLDB, 2004.

[5] Nitin Thaper and Sudipto Guha, "Dynamic Multidimensional Histograms", SIGMOD 2002.

[6] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets", VLDB, 1998.

[7] S. Ramaswamy, R. Rastogi, and S. Kyuseok, "Efficient Algorithms for Mining Outliers from Large Data Sets", SIGMOD, 2000.

[8] M. M. Breunig, H. P. Kriegel, R. T. Ng and J. Sander, "LOF: identifying density-based local outliers", SIGMOD, 1998.

[9] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces", Proc of PKDD'02, 2002

[10] C. Aggarwal and P. Yu, "Outlier detection for high dimensional data", SIGMOD, 2001.

[11] D. Hawkins, Identification of outliers. Chapman and Hall, Reading, London, 1980.

[12] Portnoy, L., Eskin, E and Stolfo, S., "Intrusion detection with unlabeled data using clustering", DMSA-2001.

[13] Javitz, H. S. and Vadles, A., "The NIDES statistical component: Description and justification", Technical report, SRI International, 1993.

[14] Eleazar Eskin, A., M., L. and S., "A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data", In Applications of Data Mining of Computer Security, 2002.

[15] L. O. Chalaghan, N. Mishra, A. Meyerson and S. Guha. "Streaming-data algorithms for high-quality clustering", In: Proc of ICDE, 2002.

[16] W. Lee, S. J. Stolfo and K. Mok. Data mining in work flow environments: Experiences in intrusion detection. In Proc of KDD, 1999.

[17] Stephanie Forrest, S.A. Hofmeyr, A. Somayaji and T.A. Longstaff. A sense of self for unix processes. In Proc of the IEEE Symposium on Security and Privacy, 1996.

[18] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In Proc of the USENIX Security Symposium, 1998.

[19] Anup Ghosh and Aaron Schwartzbard. A study in using neural networks for anomaly and misuse detection. In Proc of the Eighth USENIX Security Symposium, 1999.

[20] Christa Warrender, Stepanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: alternative data models. In Proc of the IEEE Symposium on Security and Privacy, 1999.