University of Crete
Department of Computer Science

Outlier Detection over Data Streams using Statistical Modeling and Density Neighborhoods

MSc Thesis

Dimitrios Velegrakis

Heraklion

January 2009

Abstract

In the last few years, Outlier detection has become an important problem in many industrial and financial applications. Outliers are usually caused by system faults, a sudden or an unexpected change in the existing behavior and human errors. This problem is further complicated by the fact that in many cases, outliers have to be detected from data streams that arrive at an enormous pace. Despite the enormous amount of data being collected in many scientific and commercial applications, particular events of interests are still quite rare. These rare events, very often called outliers or anomalies, are defined as events that occur very infrequently. Detection of outliers has recently gained a lot of attention in many domains, ranging from video surveillance and intrusion detection to fraudulent transactions, web usage logs and direct marketing. Data mining techniques developed for this problem are based on both supervised and unsupervised learning.

We propose a novel unsupervised, non-parametric and incremental algorithm. This novel algorithm uses the sliding window model, combines two unsupervised techniques, namely, statistical modeling and nearest neighbor searching, achieve high precision and recall and is efficient both in time execution and memory consumption.

In this thesis, we propose a $SNNOS$(**S**tatistical **N**earest **N**eighbor **O**utlier **S**tream) algorithm which uses the sliding-window model and which takes the advantages of each technique, aiming to get results with high accuracy, efficient time execution and low memory consumption. To this direction, we propose a novel framework which is separated into two stages,the statistical and the density stage. In the statistical stage, our aim is to estimate approximately the distribution of data. We use kernel density functions to estimate the probability densities for data points. Then, we employ ten continuous distributions functions and compute the probability density for data points of each function using the technique of Maximum-Likelihood Estimation. We ?nd the distribution which has a probability density very close to the probability density of kernel density function. In the density stage, we propose a scoring function which is based to nearest neighbor algorithm and is called DNO (**D**ensity **N**eighborhood **O**utlierness).

Acknowledgements

First of all I would like to thank my Professor Dimitrio Plexousaki, my supervisor, for his continuous guidance and encouragement during this work.

I sincerely thank my friends from the Information Systems Laboratory at the Institute of Computer Science (ICS) of the Foundation for Research and Technology Hellas (FO.R.T.H.) for their friendship and the pleasant moments we spent. Moreover, I want to thank my friends Christo, Giorgo and Milto from Network System Laboratory for their important and useful advice during my work.

I am sincerely indebted to my parents Giorgo and Maria and my brother Yannis for all the encouragement they have given to me. Their constant support and encouragement have really brought me here.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 3

1.3 Thesis outline 3

2 Data Streams **5**

2.1 Data Stream Model 5

2.2 Time Windows 6

2.2.1 Sliding Windows 6

2.2.2 Landmark Windows 7

2.2.3 Tilted Windows 7

2.3 Comparison of windows 8

3 Outlier Detection **11**

3.1 Characteristics of Outliers 11

3.2 Characteristics of Outlier Detection Approaches 12

3.3 Taxonomy of Outlier Detection Methods 14

3.3.1 Classification-based Method 14

3.3.2 Nearest Neighbor-based Method 19

3.3.3 Cluster-based Method 24

3.3.4 Statistical-based Method 28

3.3.5 Information Theory-based Outlier Method 35

3.3.6 Spectral-based Method 36

3.4 Summary 38

4 Outlier Detection over Data Streams **40**

4.1 Incremental Local Outlier Detection over Data Streams 41

4.1.1 Introduction 41

4.1.2 Methodology 42

4.2 Frequent Pattern Based Outlier Detection over Data Streams 47

4.2.1 Introduction 47

4.2.2 Methodology 49

4.3 Online Cluster Based Outlier Detection over Data Streams 50

4.3.1 Introduction 50

4.3.2 Methodology 50

5 Statistical Nearest Neighbor Outlier Stream Algorithm **54**

5.1 Problem definition and Basic formulation 54

5.2 Statistical modeling 55

5.2.1 Kernel Density Estimation 55

5.2.2 Maximum Likelihood Estimation 56

5.2.3 Our proposed method for combining the KDE and MLE 59

5.3 Nearest Neighbor 60

5.3.1 Density-Based Outlierness 60

5.3.2 Incremental Strategy 61

5.4 Statistical Nearest Neighbor Outlier Stream Algorithm 62

6 Performance Evaluation and Experimental Results **65**

6.1 Description of Datasets 65

6.2 ROC curves 66

6.3 Precision and Recall 67

6.4 Precision, Recall and Execution Time for SNNOS 68

6.4.1 Comparison of SNNOS for various window sizes for sliding window model 69

6.4.2 Incremental vs Non-Incremental Implementation 69

6.4.3 Statistical vs Nearest Neighbor approach 71

6.5 Real Datasets 72
6.5.1 Meteorological Dataset 74
6.5.2 Shuttle Dataset 75
6.5.3 Letter Dataset 75
6.6 Synthetic Dataset 77
6.7 Comparing SNNOS with ILOF 77
6.8 Memory Consumption 83

7 Conclusion and Future work **84**

List of Tables

6.1 Confusion matrix defines four possible scenarios when classifying class C . 67

6.2 Comparison of time execution for the four datasets and for five different sizes of windows, 100, 200, 300, 400 and 500 69

6.3 Comparison of precision for the four datasets and for five different sizes of windows, 100, 200, 300, 400 and 500 70

6.4 Comparison of recall for the four datasets and for five different sizes of windows, 100, 200, 300, 400 and 500 70

6.5 Nearest Neighbor part average time execution of $SNNOS$ and for window sizes 100 and 200 72

6.6 Statistical part time execution of $SNNOS$ and for window sizes 100 and 200 72

6.7 Precision and Recall of $SNNOS$ for window size 200 using only Statistical Part 73

6.8 Precision and Recall of $SNNOS$ for window size 200 using only Nearest Neighbor Part 73

6.9 Relative Decrement in performance of $SNNOS$ for window size 200 . . . 73

6.10 Comparison of time execution for $SNNOS$ and $ILOF$ 79

6.11 Comparison of time execution for $SNNOS$ and $Hybrid - ILOF$ with incremental step for $SNNOS$ equal to 50 and incremental step for $Hybrid - ILOF$ equal to 1 79

6.12 Comparison of time execution for $SNNOS$ and $ILOF$ with the same incremental step for $SNNOS$, $ILOF$ and $Hybrid - ILOF$ equal to 1 79

6.13 Comparison of *Precision* for $SNNOS$, $ILOF$ and $Hybrid - ILOF$ . . . 79

6.14 Comparison of *Recall* for $SNNOS$ and $ILOF$ and $Hybrid - ILOF$ . . . 80

6.15 Comparison of AUC values for $SNNOS$ and $ILOF$ and $Hybrid - ILOF$. 80

List of Figures

2.1 Sliding window 7
2.2 Landmark window 8
2.3 Natural Tilted window 8
2.4 Logarithmic Tilted window 9
2.5 Landmark Window Model 10
2.6 Sliding Window Model 10

3.1 Multi-class Anomaly Detection 15
3.2 One-class Anomaly Detection 16
3.3 Local density based techniques over Global density based techniques. . . . 22

4.1 The general framework for insertion of data record and computing its LOF value in incremental LOF algorithm. 44
4.2 Update of k-nearest neighbor distance upon insertion of a new record . . . 44
4.3 The framework for deletion of data record in incremental LOF method. . . 46
4.4 Update of k-nearest neighbor distance upon deletion of record p_c 46
4.5 The DSFindFPOF algorithm 49
4.6 The CBOD algorithm 53

6.1 The ROC curves for different detection algorithms 68
6.2 Time execution of the incremental and non-incremental implementation, using 1000 data points and windows of sizes 100 and 200. 71
6.3 Outlier Validation for the Meteorological Dataset for the time period $[1, 100]$ 74
6.4 ROC Curve for DNO for the Meteorological Dataset 75

6.5 ROC Curve for DNO for the Shuttle Dataset 76
6.6 ROC Curve for DNO for the Letter Dataset 76
6.7 ROC Curve for DNO for the Synthetic Dataset 77
6.8 Comparison of ROC Curves for ILOF and DNO for Letter Dataset 80
6.9 Comparison of ROC Curves for ILOF and DNO for Letter dataset with different sizes of windows for DNO 81
6.10 Comparison of SNNOS and ILOF on how the value of AUC is related to the speedup of execution time 81
6.11 Comparison of SNNOS and ILOF on how the value of precision is related to the speedup of execution time 82
6.12 Comparison of SNNOS and ILOF on how the value of recall is related to the speedup of execution time 82

Chapter 1

Introduction

1.1 Motivation

In recent years, we have witnessed the widely recognized phenomenon of high speed data streams. A data stream is a massive real-time continuous sequence of data elements. The typical applications include sensor network, stock tickers, network traffic measurement, click streams and telecommunication call records. The main challenge of these applications is that the data element arrives continuously and the volume of the data is so large that they can hardly be stored in the main memory (even on the local disk) for online processing, and sometimes the system has to drop some data elements due to the high arriving speed. The data in the traditional database applications are organized on the hard disk by the *Database Management System*($DBMS$) so the queries from the users can be answered by scanning the indices or the whole data set. Considering of the characteristics of the stream applications, it is not feasible to simply load the arriving data elements onto the $DBMS$ and operate on them because the traditional $DBMS$ are not designed for rapid and continuous loading of individual data element and they do not directly support continuous queries that are typical of data stream applications.

Data mining, as a powerful knowledge discovery tool, aims at modeling relationships and discovering hidden patterns in large databases. Among four typical data mining tasks, outlier detection is the closest to the initial motivation behind data mining than predictive modeling, cluster analysis and association analysis. Outlier detection has been a widely

researched problem in several knowledge disciplines, including statistics, data mining and machine learning. It is also known as anomaly detection [2][21][64] and novelty detection [18][48] in some literature. Being called differently, all these definitions aim at identifying instances of unusual behavior when compared to the majority of observations. Coming across various definitions of an outlier, it seems that no universally accepted definition exists.Two classical definitions of an outlier include Hawkins[29] and Barnett and Lewis[9]. According to the former, "an outlier is an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism", where as the latter defines "an outlier is an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data". The term "outlier" can generally be defined as an observation that is significantly different from the other values in a data set.

Although, data mining is a well-established field in itself, direct application of mining algorithms to data streams is often unsuitable due to the fact that it is impossible to maintain all the stream elements in memory. Moreover, as new data arrives online, the mining algorithms must adapt to the changing trends. Hence, approximation and adaptivity are the key ingredients of any data stream mining algorithm. Data stream mining has received more of the researchers attention in the past few years. Some of the most common stream-mining tasks include:

- Multi-dimensional on-line analysis.
- Mining spatial and temporal correlations in streaming data trends.
- Mining novelty, outliers and anomalous behavior.
- On-line adaptive clustering and classification.
- Frequent pattern matching.

Based on real-life applications, it can clearly be seen that outlier detection is a quite critical part of any data analysis. In the detection of outliers, there is a universally accepted assumption that the number of anomalous data is considerably smaller than normal data in a data set. Thus, a straightforward approach to identify outliers is to construct a profile

of the normal behaviors of the data and then use certain measure methods to calculate the degree to which data deviate from the profile in a data set. Those instances that significantly deviate from the profile are declared as outliers. However, existing methods using pre-labeled data to build a normal model in a training phase before detecting outliers are very challenging since not all possible normal behaviors have been encompassed within the normal model.

1.2 Objectives

In this thesis, we present our research on outlier detection methods, which target on sliding-window data streams. We propose an efficient, novel, nonparametric and unsupervised outlier detection algorithm that require no prior knowledge of data. Our algorithm is called *SNNOS* (**S**tatistical **N**earest **N**eighbor **O**utlier **S**tream) and consists of three phases. In the first phase we store temporarily the data which arrives in a cache memory manager. In the second phase, we use a novel and efficient strategy which combines two methods of statistical learning, the *Kernel Density Estimation*(KDE)method and the *Maximum Likelihood Estimation*(MLE) method. The KDE method uses kernel density functions to estimate probability densities for each data point in the window. In the MLE method, we use ten continuous distributions and for each of them we estimate the probability density for each data point in the window using Maximum Likelihood Estimation. Then, we estimate approximately the distribution of data by comparing the probability densities. Finally, we compute two quantiles:lower-quantile with parameter 0.05 and the upper-quantile with parameter 0.95, which will be used as thresholds for finding outliers. The third phase, uses *nearest-neighbor search* algorithm to compute a density scoring function. The density scoring function is called DNO (**D**ensity **N**eighborhood **O**utlierness). The intuition behind the DNO is that we want to suggest an alternative and efficient density scoring function for the computation of neighborhood's density.

1.3 Thesis outline

The thesis is structured as follows:

- Chapter 2 presents the main concepts of data streams.
- Chapter 3 describes outliers and presents a taxonomy of the most known approaches for outlier detection.
- Chapter 4 discusses the related work to the specific domain of outlier detection over data streams.
- Chapter 5 describes the novel algorithm SNNOS (Statistcal Nearest Neighbor Outlier Stream) which consists of two parts the statistical and the nearest neighbor. Also, we explain the intuition behind the use of each part.
- Chapter 6 presents a set of experiments that we performed in order to evaluate the basic characteristics of our algorithm. We also discuss the results and arrive to important conclusions regarding the performance of the algorithm and their extensibility.
- Chapter 7 summarizes our work and its contributions and presents some extensibility suggestions for our algorithm.

Chapter 2

Data Streams

2.1 Data Stream Model

In the data stream model, some or all of the input data that are to be operated on are not available for random access from disk or memory, but rather arrive as one or more continuous data streams. Data streams differ from the conventional stored relation model in several ways [25]:

- The data elements in the stream arrive online.
- The system has no control over the order in which data elements arrive to be processed, either within a data stream or across data streams.
- Data streams are usually unbounded in size.
- Once an element from a data stream has been processed it is discarded or archived and it cannot be retrieved easily unless it is explicitly stored in memory, which typically is small relative to the size of the data streams.

In the standard stream model, the input elements $a_1, a_2, ..., a_j, ...$ arrive sequentially, item by item and describe an underlying function A. Stream models differ on how a_i describes A [25]. We can distinguish between:

- Insert Only Model: once an element a_i is seen, it cannot be changed.
- Insert-Delete Model: elements a_i can be deleted or updated.
- Accumulative Model: each a_i is an increment to $A[j] = A[j-1] + a_i$.

2.2 Time Windows

Time windows are a commonly used approach to answer queries in open-ended data streams. Instead of computing an answer over the whole data stream, the query (or operator) is computed, eventually several times, over a finite subset of tuples. In this model, a time stamp is associated with each tuple. The time stamp defines when a specific tuple is valid (e.g. inside the window) or not. Queries are evaluated over the tuples inside the window. However, in the case of joining multiple streams the semantics of time stamps is much less clear e.g. the time stamp of an output tuple. Several window models have been used in the literature. The following are the most relevant [25].

2.2.1 Sliding Windows

Usually data streams are of unbounded length, and in many applications old data elements are less important compared with new elements. For instance, in the traffic monitoring system the administrator might be more interested in the patterns discovered among the recent stream data.Most of the time, we are not interested in computing statistics over all the past but only in the recent past. The simplest approach are sliding windows of fixed size. This type of windows is similar to first in, first out data structures. Whenever an element j is observed and inserted in the window, another element j - w, where w represents the window size, is forgotten [25]. An example of sliding window is shown in Figure 2.1. The sliding window is a commonly used model to address this issue. There are two kinds of sliding window models:

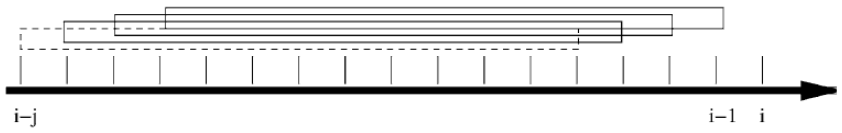

Figure 2.1: Sliding window

- Count-based model: only the most recent N elements are kept in the window, where N is the window size
- Time-based model: only keep elements arrived within a fixed time period.

2.2.2 Landmark Windows

Landmark windows [26] identify relevant points (the landmarks) in the data stream and the aggregate operator uses all records seen so far after the landmark. Successive windows share some initial points and are of growing size. In some applications, the landmarks have a natural semantic. For example, in daily basis aggregates the beginning of the day is a landmark. An example of landmark window is shown in Figure 2.2.

2.2.3 Tilted Windows

In tilted windows, the time scale is compressed. The most recent data are stored inside the window at the finest detail (granularity). Oldest information is stored at a coarser detail, in an aggregated way. The level of granularity depends on the application. This window model is designated a tilted time window. Tilted time windows can be designed in several ways. Han and Kamber [36] presented two possible variants: natural tilted time windows, and logarithm tilted windows. Illustrative examples are presented in Figure 2.3 and Figure 2.4. In the first case, data are stored with granularity according to a natural

Figure 2.2: Landmark window

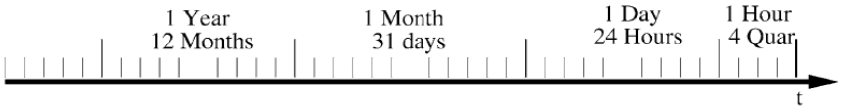

Figure 2.3: Natural Tilted window

time taxonomy: last hour at a granularity of 15 minutes (4 points), last day in hours (24 points), last month in days (32 points) and last year in months (12 points). In the case of logarithmic tilted windows, given a maximum granularity with periods of t , the granularity decreases logarithmically as data are older. As time goes by, the window stores the last time period t , the one before that, and consecutive aggregates of less granularity (two periods, four periods, eight periods etc.).

2.3 Comparison of windows

In the *landmark* and *sliding* windows any past observation either is in the window or it is not inside the window will be forgotten in the next step. In *tilted* windows the data

Figure 2.4: Logarithmic Tilted window

asymmetrically distributed into multiple time slots such that the recent time period is assigned more time slots than the past. The *tilted* window is suitable for people to mine the recent data at a fine granularity while mining the long-term data at a coarse granularity. Moreover, the *landmark* window is not aware of time and therefore cannot distinguish between new data and old ones. To overcome this difficulty, the *sliding* window, a variation of the landmark model, has been proposed. It assigns different weights to transactions such that new ones have higher weights than old ones. These three approaches provide approximate answers for long-term data and adjust their storage requirement based on the available space. Furthermore, the three window models namely, sliding, landmark and tilted satisfy two important requirements: *approximation* and *adjustability*. However, in certain applications, users could only be interested in the data recently arriving within a fixed time period. Obviously, the *tilted* and *landmark* windows models are unable to satisfy this need. In contrast, the sliding-window model achieves this goal. Given a window size W, only the latest W transactions are utilized for mining. As a transaction arrives, the oldest transaction in the sliding window is expired. Furthermore, compared with the *tilted* and *landmark* windows models considering only the insertion of transactions, the sliding-window model further considers the deletion of transactions. Therefore, if a method succeeds in the sliding-window model, it can be easily applied to the *tilted* and *landmark* windows models. Moreover, *tilted* and *landmark* windows models consider a fixed number of transactions as the basic unit for mining, which is not easy for people to specify. By

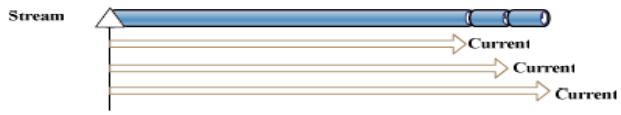

Figure 2.5: Landmark Window Model

Figure 2.6: Sliding Window Model

contrast, it is natural for people to specify a time period as the basic unit. Therefore, in this paper, we use the time-sensitive sliding-window model, which regards a fixed time period as the basic unit for mining.

Definition: Time-sensitive Sliding-window (TS)

Given a time point t and a time period p, the set of all the transactions arriving in $[t-p+1, t]$ will form a basic block. A data stream is decomposed into a sequence of basic blocks, which are assigned with serial numbers starting at 1. Given a window with length $|W|$, we slide it over this sequence to see a set of overlapping sequences, where each sequence is called the time-sensitive sliding-window.

Chapter 3

Outlier Detection

Outlier detection refers to the problem of finding patterns in data that do not conform to expected behavior. These non-conforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities or contaminants in different application domains. Of these, anomalies and outliers are two terms used most commonly in the context of anomaly detection; sometimes interchangeably. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, localization and tracking, environmental monitoring, insurance or health care, intrusion detection, fault detection in safety critical systems, and military surveillance for enemy activities [14],[87].

3.1 Characteristics of Outliers

- Type of Detected Outliers

  Outliers can be identified as either global or local outliers. A global outlier is an anomalous data point with respect to all other points in the whole data set, but possibly not with respect to points in its local neighborhood. A local outlier is a data point that is significantly different with respect to other points in its local neighborhood, but may not be an outlier in a global view of the data set [87].

- Degree of Being an Outlier

  A data point can be considered as an outlier in two manners, scalar (binary) or

outlierness. The scalar fashion is that the point is either an outlier or not. On the other hand, the outlierness fashion provides the degree of which the point is an outlier when compared to other points in a data set. This outlierness is also known as anomaly score or outlier score, which usually can be calculated by using specific measurement methods [71].

- Dimension of Detected Outliers

  Whether a data point is an outlier or not is determined by the values of its attributes. A univariate datum that has a single attribute can be detected as an outlier only based on the fact that a single attribute is anomalous with respect to that of other data. On the other hand, a multivariate datum that has multiple attributes may be identified as an outlier since some of its attributes together have anomalous values, even if none of its attributes individually has an anomalous value. Thus, for detecting multivariate outliers is a more complicated process.

- Number of Detected Outliers at Once

  Outlier detection techniques can be designed to identify different number of outliers at a time. In some techniques, one outlier is identified and removed at a time, then the procedure will be repeated until no outliers are detected. These techniques may be subject to the problem of missing some real outliers during the iteration. On the other hand, for other techniques, they can identify a collection of outliers at once. However, these techniques may cause some normal data to be declared as outliers during iteration.

3.2 Characteristics of Outlier Detection Approaches

Use of Pre-labeled Data

Outlier detection approaches can generally be classified into three basic categories, i.e., supervised, unsupervised and semi-supervised learning approaches. This categorization is based on the degree of using pre-defined labels to classify normal or abnormal data [71].

- Supervised learning

These approaches initially require the learning of a normality and an abnormality models by using pre-labeled data, and then classify a new data point as normal or abnormal depending on which model the data points fits into [39] [76]. These supervised learning approaches usually are applied for many fraud detection and intrusion detection applications. However, they have two major drawbacks, i.e., pre-labeled data is not easy to obtain in many real-life applications, and also new types of rare events may not be included in pre-labeled data.

- Unsupervised learning

  These approaches can identify outliers without the need of pre-labeled data. For example, distributed methods identify outliers based on a standard statistical distribution model [81] [68]. Similarly, distance- based methods identify outliers based on the measure of full dimensional distance between a point and its nearest neighbors [51] [10] [12]. Compared to supervised learning approaches, these unsupervised learning approaches are more general because they do not need pre-labeled data that are not available in many practical applications.

Use of Parameters of Data Distribution

Unsupervised learning approaches can be further grouped into three categories, i.e., parametric, non-parametric and semi-parametric methods, on the basis of the degree of using the parameters of the underlying data distribution.

- Parametric methods

  These methods assume that the whole dataset can be modeled to one standard statistical distribution (e.g., the normal distribution), and then directly calculate the estimated parameters of this distribution based on means and covariance of the original data. A point that deviates significantly from the data model is declared as an outlier. These methods are suitable for situations in which the data distribution model is a priori known and parameter settings have been previously determined. However, in many practical situations, a priori knowledge of the underlying data distribution is not always available and also it may not be a simple task to compute the parameters of the data distribution.

- Non-parametric methods

  These methods make no assumption on the statistical properties of data and instead identify outliers based either on the full dimensional distance measure between points for nearest neighbor approaches or on the probability density for kernel density approaches. In nearest-neighbor approaches outliers are considered as those points that are distant from their own neighbors in the data set. These methods also use some user-defined parameters ranging from the size of local neighborhood to the threshold of distance measure. In kernel density approaches a kernel density function is used for estimating the probability density function of a data point. Compared to parametric methods, these non-parametric methods are more flexible and autonomous due to the fact that they require no data distribution knowledge. However, they may have expensive time complexity, especially for high dimensional data sets. Also, the choice of appropriate values for user-defined parameters is not easy.

3.3 Taxonomy of Outlier Detection Methods

3.3.1 Classification-based Method

Classification [71] is used to learn a model (classifier) from a set of labeled data instances (training) and then, classify a test instance into one of the classes using the learnt model (testing). Classification based anomaly detection techniques operate in a similar two-phase fashion. The training phase learns a classifier using the available labeled training data. Classification based anomaly detection techniques operate under the following general assumption:

A classifier that can distinguish between normal and anomalous classes can be learnt in the given feature space.

Based on the labels available for training phase, classification based anomaly detection techniques can be grouped into two broad categories: multi-class and one-class anomaly detection techniques. Multi-class classification based anomaly detection techniques assume that the training data contain labeled instances belonging to multiple normal classes [8]. Such anomaly detection techniques learn a classifier to distinguish between each normal

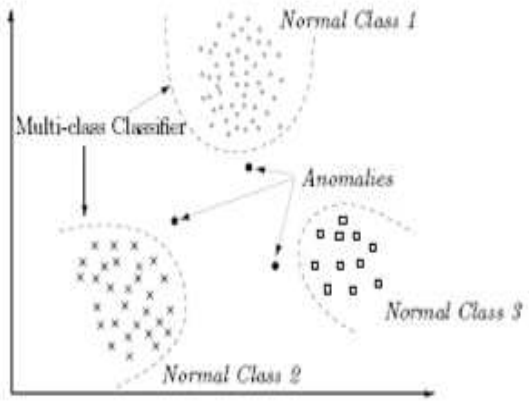

Figure 3.1: Multi-class Anomaly Detection

class against the rest of the classes.

See figure 3.1 for illustration. A test instance is considered anomalous if its not classified as normal by any of the classifiers. Some techniques in this sub-category associate a confidence score with the prediction made by the classifier. If none of the classifiers are confident in classifying the test instance as normal, the instance is declared to be anomalous.

One-class classification based anomaly detection techniques assume that all training instances have only one class label. Such techniques learn a discriminative boundary around the normal instances using a one-class classification algorithm, e.g., one-class SVMs[60], one-class Kernel Fisher Discriminants [58], as shown in figure 3.2. Any test instance that does not fall within the learnt boundary is declared as anomalous.

Neural Networks-based Method

Neural networks have been applied to anomaly detection in multi-class as well as one-class setting. A basic multi-class anomaly detection technique using neural networks operates in two steps. First, a neural network is trained on the normal training data to learn the different normal classes. Second, each test instance is provided as an input to the neural network. If the network accepts the test input, it is normal and if the network rejects a

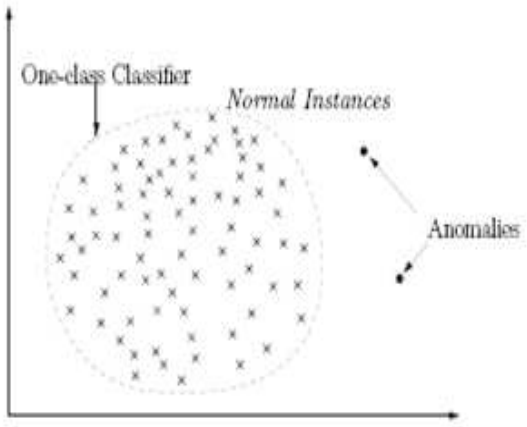

Figure 3.2: One-class Anomaly Detection

test input, it is an anomaly [67]. Replicator Neural Networks have been used for one-class anomaly detection[78]. A multi-layer feed forward neural network is constructed that has the same number of input and output neurons (corresponding to the features in the data). The training involves compressing data into three hidden layers. The testing phase involves reconstructing each data instance x_i using the learnt network to obtain the reconstructed output o_i . The reconstruction error δ_i for the test instance x_i is then computed as:

$$\delta_i = \frac{1}{n}\sum_{j=1}^{n}(x_{ij} - o_{ij})^2 \tag{3.1}$$

where n is the number of features over which the data is defined. The reconstruction error δ_i is directly used as an anomaly score for the test instance.

Bayesian Networks-based Method

Bayesian networks has been used for anomaly detection in the multi-class setting. A basic technique for a univariate categorical data set using a Bayesian network estimates the posterior probability of observing a class label (from a set of normal class labels and the anomaly class label), given a test data instance. The class label with the largest

posterior is chosen as the predicted class for the given test instance. The likelihood of observing the test instance given a class, and the prior on the class probabilities, are estimated from the training data set. The zero probabilities, especially for the anomaly class, are smoothed using Laplace Smoothing. The basic technique can be generalized to multivariate categorical data set by aggregating the per-attribute posterior probabilities for each test instance and using the aggregated value to assign a class label to the test instance. Several variants of the basic technique has been proposed for network intrusion detection [8][62][13], for novelty detection in video surveillance [18], for anomaly detection in text data [6], and for disease outbreak detection [79]. The basic technique described above assumes independence between the different attributes. Several variations of the basic technique have been proposed that capture the conditional dependencies between the different attributes using more complex Bayesian networks [17].

Support Vector Machines-based Method

Support Vector Machines (SVMs) have been applied to anomaly detection in the one-class setting. Such techniques use one class learning techniques for SVM [56] and learn a region that contains the training data instances (a boundary). Kernels, such as radial basis function (RBF) kernel, can be used to learn complex regions. For each test instance, the basic technique determines if the test instance falls within the learnt region. If a test instance falls within the learnt region, it is declared as normal, else it is declared as anomalous. Variants of the basic technique have been proposed for anomaly detection in audio signal data, novelty detection in power generation plants and system call intrusion detection [44]. The basic technique also been extended to detect anomalies in temporal sequences [46]. A variant of the basic technique finds the smallest hyper-sphere in the kernel space, which contains all training instances, and then determines which side of that hyper-sphere does a test instance lie. If a test instance lies outside the hyper-sphere, it is declared to be anomalous. Another approach is to use Robust Support Vector Machines (RSVM) [34] [66] which are robust to the presence of anomalies in the training data . Also, RSVM have been applied to system call intrusion detection.

Rule-based Method

Rule based anomaly detection techniques learn rules that capture the normal behavior of a system. A test instance that is not covered by any such rule is considered as an anomaly. Rule based techniques have been applied in multi-class as well as one-class setting. A basic multi-class rule based technique consists of two steps. First step is to learn rules from the training data using a rule learning algorithm, such as RIPPER, Decision Trees, etc. Each rule has an associated confidence value which is proportional to ratio between the number of training instances correctly classified by the rule and the total number of training instances covered by the rule. Second step is to find, for each test instance, the rule that best captures the test instance. The inverse of the confidence associated with the best rule is the anomaly score of the test instance. Several minor variants of the basic rule based technique have been proposed [59]. Association rule mining has been used for one-class anomaly detection by generating rules from the data in an unsupervised fashion. Association rules are generated from a categorical data set. To ensure that the rules correspond to strong patterns, a support threshold is used to prune out rules with low support. Association rule mining based techniques have been used for network intrusion detection [72] [8], system call intrusion detection, credit card fraud detection and fraud detection in spacecraft house keeping data . Frequent itemsets are generated in the intermediate step of association rule mining algorithms. In [85] the authors propose an anomaly detection algorithm for categorical data sets in which the anomaly score of a test instance is equal to the number of frequent itemsets it occurs in.

Computational Complexity

The computational complexity of classification based techniques depends on the classification algorithm being used. Generally, training decision trees tends to be faster while techniques that involve quadratic optimization, such as SVMs, are more expensive. The testing phase of classification techniques is usually very fast since the testing phase uses a learnt model for classification.

Advantages and Disadvantages of Classification Based Techniques

The advantages of classification based techniques are as follows: (1) Classification based techniques, especially the multi-class techniques, can make use of powerful algorithms that

can distinguish between instances belonging to different classes. (2) The testing phase of classification based techniques is fast since each test instance needs to be compared against the pre-computed model.

The disadvantages of classification based techniques are as follows: (1) Multi-class classification based techniques rely on availability of accurate labels for various normal classes, which is often not possible. (2) Classification based techniques assign a label to each test instance, which can also become a disadvantage when a meaningful anomaly score is desired for the test instances. Some classification techniques that obtain a probabilistic prediction score from the output of a classifier, can be used to address this issue.

3.3.2 Nearest Neighbor-based Method

The concept of nearest neighbor analysis has been used in several anomaly detection techniques. Such techniques are based on the following key assumption:

Normal data instances occur in dense neighborhoods, while anomalies occur far from their closest neighbors.

Nearest neighbor based anomaly detection techniques require a distance or similarity measure defined between two data instances. Distance (or similarity) between two data instances can be computed in different ways. For continuous attributes, Euclidean distance is a popular choice but other measures can be used [71]. For categorical attributes, simple matching coefficient is often used but more complex distance measures can be used [28]. For multivariate data instances, distance or similarity is usually computed for each attribute and then combined. Most of the techniques that will be discussed in this section, as well as the clustering based techniques section 3.3.3 do not require the distance measure to be strictly metric. The measures are typically required to be positive-definite and symmetric, but they are not required to satisfy the triangle inequality. Nearest neighbor based anomaly detection techniques can be broadly grouped into two categories:

- Techniques that use the distance of a data instance to its kth nearest neighbor as the anomaly score.
- Techniques that compute the relative density of each data instance to compute its

anomaly score.

Distance to k Nearest Neighbor

A basic nearest neighbor anomaly detection technique is based on the following definition: *The anomaly score of a data instance is defined as its distance to its k^{th} nearest neighbor in a given data set.*

Usually, a threshold is applied on the anomaly score to determine if a test instance is anomalous or not. [86] compute the anomaly score of a data instance as the sum of its distances from its k nearest neighbors. A different way to compute the anomaly score of a data instance is to count the number of nearest neighbors that are not more than d distance apart from the given data instance [41]. This method can also be viewed as estimating the global density for each data instance since it involves counting the number of neighbors in a hyper-sphere of radius d. While most techniques discussed in this category so far have been proposed to handle continuous attributes, several variants have been proposed to handle other data types. A hyper-graph based technique is proposed by [77] called HOT where the authors model the categorical values using a hyper-graph, and measure distance between two data instances by analyzing the connectivity of the graph. A distance measure for data containing a mix of categorical and continuous attributes has been proposed for anomaly detection [49]. The authors define links between two instances by adding distance for categorical and continuous attributes separately. For categorical attributes, the number of attributes for which the two instances have same values defines the distance between them. For continuous attributes, a covariance matrix is maintained to capture the dependencies between the continuous values. Several variants of the basic technique have been proposed to improve the efficiency. Some techniques prune the search space by either ignoring instances that cannot be anomalous or by focussing on instances that are most likely to be anomalous. [10] show that for a sufficiently randomized data, a simple pruning step could result in the average complexity of the nearest neighbor search to be nearly linear. After calculating the nearest neighbors for a data in- stance, the algorithm sets the anomaly threshold for any data instance to the score of the weakest anomaly found so far. Using this pruning procedure, the technique discards instances that are close, and hence not

interesting. [55] propose a partition based technique, which first clusters the instances and computes lower and upper bounds on distance of a instance from its k^{th} nearest neighbor for instances in each partition. This information is then used to identify the partitions that cannot possibly contain the top k anomalies; such partitions are pruned. Anomalies are then computed from the remaining instances (belonging to unpruned partitions) in a final phase. Similar cluster based pruning has been proposed by [27] [74]. [80] use sampling to improve the efficiency of the nearest neighbor based technique. The authors compute the nearest neighbor of every instance within a smaller sample from the data set. Thus the complexity of the proposed technique is reduced to $O(MN)$ where M is the sample size chosen. To prune the search space for nearest neighbors, several techniques partition the attribute space into a hyper-grid consisting of hypercubes of fixed sizes. The intuition behind such techniques is that if a hypercube contains many instances, such instances are likely to be normal. Moreover, if for a given instance, the hypercube that contains the instance and its adjoining hypercubes contain very few instances, the given instance is likely to be anomalous.

Relative Density Nearest Neighbor

Density based anomaly detection techniques estimate the density of the neighbor- hood of each data instance. An instance that lies in a neighborhood with low density is declared to be anomalous while an instance that lies in a dense neighborhood is declared to be normal. For a given data instance, the distance to its k_{th} nearest neighbor is equivalent to the radius of a hyper-sphere, centered at the given data instance, which contains k other instances. Thus the distance to the k_{th} nearest neighbor for a given data instance can be viewed as an estimate of the inverse of the density of the instance in the data set and the basic nearest neighbor based technique described in the previous subsection can be considered as a density based anomaly detection technique. Density based techniques perform poorly if the data has regions of varying densities. For example, consider a two-dimensional data set shown in figure 3.3. Due to the low density of the cluster C_1 it is apparent that for every instance q inside the cluster C_1, the distance between the instance q and its nearest neighbor is greater than the distance between the instance p_2 and the nearest neighbor

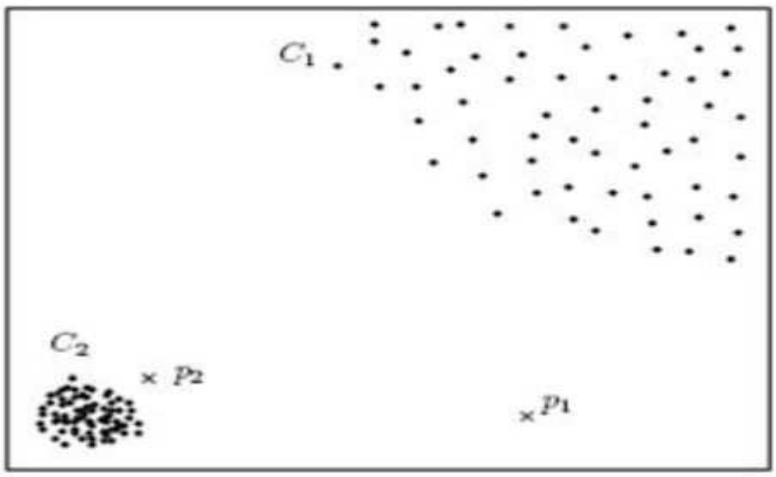

Figure 3.3: Local density based techniques over Global density based techniques.

from the cluster C_2, and the instance p_2 will not be considered as anomaly. Hence, the basic technique will fail to distinguish between p_2 and instances in C_1. However, the instance p_1 may be detected. To handle the issue of varying densities in the data set, a set of techniques have been proposed to compute density of instances relative to the density of their neighbors. [12] assign an anomaly score to a given data instance, known as Local Outlier Factor (LOF). For any given data instance, the LOF score is equal to ratio of average local density of the k nearest neighbors of the instance and the local density of the data instance itself. To find the local density for a data instance, the authors first find the radius of the smallest hyper-sphere centered at the data instance, that contains its k nearest neighbors. The local density is then computed by dividing k by the volume of this hyper-sphere. For a normal instance lying in a dense region, its local density will be similar to that of its neighbors, while for an anomalous instance, its local density will be lower than that of its nearest neighbors. Hence the anomalous instance will get a higher LOF score. In the example shown in figure 3.3, LOF will be able to capture both anomalies (p1 and p2) due to the fact that it considers the density around the data instances. Several researchers have proposed variants of LOF technique. Some of these variants estimate the local density of an instance in a different way. Some variants have adapted the original technique to more complex data types. Since the original LOF technique is $O(N^2)$ (N is the data size), several techniques have been proposed that improve the efficiency of LOF.

[73] discuss a variation of the LOF, which they call Connectivity-based Outlier Factor

(COF). The difference between LOF and COF is the manner in which the k neighborhood for an instance is computed. In COF, the neighborhood for an instance is computed in an incremental mode. To start, the closest instance to the given instance is added to the neighborhood set. The next instance added to the neighborhood set is such that its distance to the existing neighborhood set is minimum among all remaining data instances. The distance between an instance and a set of instances is defined as the minimum distance between the given instance and any instance belonging to the given set. The neighborhood is grown in this manner until it reaches size k. Once the neighborhood is computed, the anomaly score (COF) is computed in the same manner as LOF. [51] propose a measure called Multi-granularity Deviation Factor (MDEF) which is a variation of LOF. MDEF for a given data instance is equal to the standard deviation of the local densities of the nearest neighbors of the given data instance (including the data instance itself). The inverse of the standard deviation is the anomaly score for the data instance. The anomaly detection technique presented in the paper is called LOCI. Several variants of LOF have been proposed to handle different data types. [83] use a similarity measure instead of distance to handle categorical attributes. [54]extend LOF to work in an incremental fashion to detect anomalies in video sensor data. Some variants of the LOF technique have been proposed to improve its efficiency. [38] propose a variant, in which only the top n anomalies are found instead of finding LOF score for every data instance. The technique includes finding micro-clusters in the data and then finding upper and lower bound on LOF for each of the micro-clusters.

Computational Complexity

A drawback of the basic nearest neighbor based technique and the LOF technique, is the $O(N^2)$ complexity required. Since these techniques involve finding nearest neighbors for each instance efficient data structures such as k-d trees and R-trees can be used. But such techniques do not scale well as the number of attributes increases. Several techniques have directly optimized the anomaly detection technique under the assumption that only top few anomalies are interesting. If an anomaly score is required for every test instance, such techniques are not applicable. Techniques that partition the attribute space into a hyper-grid, are linear in data size but are exponential in the number of attributes, and hence

are not well suited for large number of attributes. Sampling techniques try to address the $O(N^2)$ complexity issue by determining the nearest neighbors within a small sample of the data set. But sampling might result in incorrect anomaly scores if the size of the sample is very small.

Advantages and Disadvantages of Nearest Neighbor Based Techniques

The advantages of nearest neighbor based techniques are as follows: (1) A key advantage of nearest neighbor based techniques is that they are unsupervised in nature and do not make any assumptions regarding the generative distribution for the data. Instead, they are purely data driven. (3) Adapting nearest neighbor based techniques to a different data type is straight- forward, and primarily requires defining an appropriate distance measure for the given data.

The disadvantages of nearest neighbor based techniques are as follows: (1) For unsupervised techniques, if the data has normal instances that do not have enough close neighbors or if the data has anomalies that have enough close neighbors, the technique fails to label them correctly, resulting in missed anomalies. (2) The computational complexity of the testing phase is also a significant challenge since it involves computing the distance of each test instance with all instances belonging to either the test data itself, or to the training data, to compute the nearest neighbors. (3) Performance of a nearest neighbor based technique greatly relies on a distance measure, defined between a pair of data instances, that can effectively distinguish between normal and anomalous instances. Defining distance measures between instances can be challenging when the data is complex, e.g. graphs, sequences, etc.

3.3.3 Cluster-based Method

Clustering [71] is used to group similar data instances into clusters and is primarily an unsupervised technique. Even though clustering and anomaly detection appear to be fundamentally different from each other, several clustering based anomaly detection techniques have been developed. Clustering based anomaly detection techniques can be grouped into three categories. First category of clustering based techniques rely on the following assumption:

Normal data instances belong to a cluster in the data, while anomalies either do not belong to any cluster.

Techniques based on the above assumption apply a known clustering based algorithm to the data set and declare any data instance that does not belong to any cluster as anomalous. Several clustering algorithms that do not force every data instance to belong to a cluster, such as DBSCAN [23] , ROCK [28], and SNN clustering [19] can be used. The FindOut algorithm [82] is an extension of the WaveCluster algorithm [63] in which the detected clusters are removed from the data and the residual instances are declared as anomalies. A disadvantage of such techniques is that they are not optimized to find anomalies, since the main aim of the underlying clustering algorithm is to find clusters. Second category of clustering based techniques rely on the following assumption:

Normal data instances lie close to their closest cluster centroid, while anomalies are far away from their closest cluster centroid.

Techniques based on the above assumption consist of two steps. In the first step, the data is clustered using a clustering algorithm. In the second step, for each data instance, its distance to its closest cluster centroid is calculated as its anomaly score. A number of anomaly detection techniques that follow this two step approach have been proposed using different clustering algorithms. Smith et al. [2002] studied Self-Organizing Maps (SOM), K-means Clustering, and Expectation Maximization (EM) to cluster training data and then use the clusters to classify test data. In particular, SOM [42] has been widely used to detect anomalies in a semi-supervised mode in several applications such as intrusion detection, fault detection, and fraud detection.[7] propose a technique is robust to anomalies in the training data. The authors first separate normal instances from anomalies in the training data, using frequent item-set mining, and then use the clustering based technique to detect anomalies. Techniques based on the second assumption can also operate in semi-supervised mode, in which the training data is clustered and instances belonging to the test data are compared against the clusters to obtain an anomaly score for the test data instance. If the training data has instances belonging to multiple classes, semi-supervised clustering can be applied to improve the clusters.[30] incorporate the knowledge of labels to improve on their unsupervised clustering based anomaly detection technique[32] by calculating a

measure called semantic anomaly factor which is high if the class label of an object in a cluster is different from the majority of the class labels in that cluster. Note that if the anomalies in the data form clusters by themselves, the above discussed techniques will not be able to detect such anomalies. To address this issue a third category of clustering based techniques have been proposed that rely on the following assumption:

Normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse clusters.

Techniques based on the above assumption declare instances belonging to clusters whose size and/or density is below a threshold as anomalous. Several variations of the third category of techniques have been proposed [50] [20] [32]. The technique proposed by [32], called FindCBLOF, assigns an anomaly score known as Cluster-Based Local Outlier Factor (CBLOF) for each data instance. The CBLOF score captures the size of the cluster to which the data instance belongs, as well as the distance of the data instance to its cluster centroid. Several clustering based techniques have been proposed to improve the efficiency of the existing techniques discussed above. Fixed width clustering is a linear time (O(Nd)) approximation algorithm used by various anomaly detection techniques [32] [20]. An instance is assigned to a cluster whose center is within a pre-specified distance to the given instance. If no such cluster exists then a new cluster with the instance as the center is created. Then they determine which clusters are anomalies based on their density and distance from other clusters. The width can either be a user-specified parameter [20] or can be derived from the data. [15] propose an anomaly detection technique using $k-d$ trees which provide a partitioning of the data in linear time. They apply their technique to detect anomalies in astronomical data sets where computational efficiency is an important requirement. Another technique which addresses this issue is proposed by [69]. The authors propose an indexing technique called CD-trees to efficiently partition data into clusters. The data instances which belong to sparse clusters are declared as anomalies. Several clustering based techniques require distance computation between a pair of instances. Thus, in that respect, they are similar to nearest neighbor based techniques. The choice of the distance measure is critical to the performance of the technique; hence the discussion in the previous section regarding the distance measures hold for clustering

based techniques also. The key difference between the two techniques, however, is that clustering based techniques evaluate each instance with respect to the cluster it belongs to, while nearest neighbor based techniques analyze each instance with respect to its local neighborhood.

Computational Complexity

The computational complexity of training a clustering based anomaly detection technique depends on the clustering algorithm used to generate clusters from the data. Thus such techniques can have quadratic complexity if the clustering technique requires computation of pairwise distances for all data instances, or linear when using heuristic based techniques such as k-means or approximate clustering techniques [20]. The test phase of clustering based techniques is fast, since it involves comparing a test instance with a small number of clusters.

Advantages and Disadvantages of Clustering Based Techniques

The advantages of clustering based techniques are as follows: (1) Clustering based techniques can operate in an unsupervised mode. (2) Such techniques can often be adapted to other complex data types by simply plugging in a clustering algorithm that can handle the particular data type. (3) The testing phase for clustering based techniques is fast since the number of clusters against which every test instance needs to be compared is a small constant.

The disadvantages of clustering based techniques are as follows: (1) Performance of clustering based techniques is highly dependent on the effectiveness of clustering algorithm in capturing the cluster structure of normal instances. (2) Many techniques detect anomalies as a by-product of clustering, and hence are not optimized for anomaly detection. (3) Several clustering algorithms force every instance to be assigned to some cluster. This might result in anomalies getting assigned to a large cluster, thereby being considered as normal instances by techniques that operate under the assumption that anomalies do not belong to any cluster. (4) Several clustering based techniques are effective only when the anomalies do not form significant clusters among themselves. (5) The computational complexity for clustering the data is often a bottleneck, especially if $O(N^2d)$ clustering

algorithms are used.

3.3.4 Statistical-based Method

The underlying principle of any statistical anomaly detection technique is:
An anomaly is an observation which is suspected of being partially or wholly irrelevant because it is not generated by the stochastic model assumed.
Statistical anomaly detection techniques are based on the following key assumption:
Normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model.
Statistical techniques fit a statistical model (usually for normal behavior) to the given data and then apply a statistical inference test to determine if an unseen instance belongs to this model or not. Instances that have a low probability to be generated from the learnt model, based on the applied test statistic, are declared as anomalies. Both parametric as well as non-parametric techniques have been applied to fit a statistical model. While parametric techniques assume the knowledge of underlying distribution and estimate the parameters from the given data [21] non-parametric techniques do not generally assume knowledge of underlying distribution [48]. In the next two subsection we will discuss parametric and non-parametric anomaly detection techniques.

Parametric Techniques

As mentioned before, parametric techniques assume that the normal data is generated by a parametric distribution with parameters Θ and probability density function $f(x, \Theta)$, where x is an observation. The anomaly score of a test instance (or observation) x is the inverse of the probability density function $f(x, \Theta)$. The parameters Θ are estimated from the given data.

Alternatively, a statistical hypothesis test (also referred to as discordancy test in statistical outlier detection literature [9] maybe used. The null hypothesis (H_0) for such tests is that the data instance x has been generated using the estimated distribution (with parameters Θ). If the statistical test rejects H_0, x is declared to be anomaly. A statistical hypothesis test is associated with a test statistic, which can be used to obtain a proba-

bilistic anomaly score for the data instance x. Based on the type of distribution assumed, parametric techniques can be further categorized as follows:

Gaussian Model

Such techniques assume that the data is generated from a Gaussian distribution. The parameters are estimated using Maximum Likelihood Estimates (MLE). The distance of a data instance to the estimated mean is the anomaly score for that instance. A threshold is applied to the anomaly scores to determine the anomalies. Different techniques in this category calculate the distance to the mean and the threshold in different ways. A simple outlier detection technique, often used in process quality control domain, is to declare all data instances that are more than 3σ distance away from the distribution mean μ, where σ is the standard deviation for the distribution. The $\mu \pm \sigma$region contains 99.7% of the data instances. More sophisticated statistical tests have also been used to detect anomalies, as discussed in [9]. The box plot rule is the simplest statistical technique that has been applied to detect univariate and multivariate anomalies. A box-plot graphically depicts the data using summary attributes such as smallest non-anomaly observation (min), lower quartile (Q_1), median, upper quartile (Q_3), and largest non-anomaly observation (max). The quantity $Q_3 - Q_1$ is called the Inter Quartile Range (IQR). The box plots also indicates the limits beyond which any observation will be treated as an anomaly. A data instance that lies more than $1.5 * IQR$ lower than Q_1 or $1.5 * IQR$ higher than Q_3 is declared as an anomaly. The region between $Q_1 - 1.5IQR$ and $Q_3 + 1.5IQR$ contains 99.3% of observations, and hence the choice of $1.5IQR$ boundary makes the box plot rule equivalent to the 3σ technique for Gaussian data. Grubb's test (also known as the maximum normed residual test) is used to detect anomalies in a univariate data set under the assumption that the data is generated by a Gaussian distribution. For each test instance x, its z score is computed as follows:

$$z = \frac{|x - \bar{x}|}{s} \tag{3.2}$$

where $\bar{x}$ and s are the mean and standard deviation of the data sample, respectively.

A test instance is declared to be anomalous if:

$$z > \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\alpha/(2N),N-2}}{N-2+t^2_{\alpha/(2N),N-2}}} \tag{3.3}$$

where N is the data size and $t_{\alpha/(2N),N-2}$ is a threshold used to declare an instance to be anomalous or normal. A variant of the Grubb's test for multivariate data was proposed by [37], which uses the Mahalanobis distance of a test instance x to the sample mean $\bar{x}$, to reduce multivariate observations to univariate scalars.

$$y^2 = (x-\bar{x})' \, S^{-1} \, (x-\bar{x}) \tag{3.4}$$

where S is the sample covariance matrix. The univariate Grubb's test is applied to y to determine if the instance x is anomalous or not. Several other variants of Grubb's test have been proposed to handle multivariate data sets graph structured data and Online Analytical Processing (OLAP) data cubes[4] . The student's t-test has also been applied for anomaly detection in to detect damages in structural beams. A normal sample, N_1 is compared with a test sample, N_2 using the t-test. If the test shows significant difference between them, it signifies the presence of an anomaly in N_2. [81] use a χ^2 statistic to determine anomalies in operating system call data. The training phase assumes that the normal data has a multivariate normal distribution. The value of the χ^2 statistic is determined as:

$$\chi^2 = \sum_{i=1}^{n} \frac{(X_i - E_i)^2}{E_i} \tag{3.5}$$

where X_i is the observed value of the i_{th} variable, E_i is the expected value of the i_{th} variable (obtained from the training data) and n is the number of variables. A large value of X^2 denotes that the observed sample contains anomalies.

Regression Model The basic regression model based anomaly detection technique

consists of two steps. In the first step, a regression model is fitted on the data. In the second step, for each test instance, the residual for the test instance is used to determine the anomaly score. The residual is the part of the instance which is not explained by the regression model. The magnitude of the residual can be used as the anomaly score for the test instance, though statistical tests have been proposed to determine anomalies with certain confidence [75]. Presence of anomalies in the training data can influence the regression parameters and hence the regression model might not produce accurate results. A popular technique to handle such anomalies while fitting regression models is called robust regression (estimation of regression parameters while accommodating anomalies). Robust regression techniques not only hide the anomalies, but can also detect the anomalies, because the anomalies tend to have larger residuals from the robust fit. A variant that detect anomalies in multivariate time-series data generated by an Autoregressive Moving Average (ARMA) model, was proposed by [24]. In this technique the authors transform the multivariate time-series to univariate time-series by linearly combining the components of the multivariate time-series. The interesting linear combinations (projections in 1-d space) are obtained using a projection pursuit technique that maximizes the Kurtosis coefficient (a measure for the degree of peakedness/flatness in the variable distribution) of the time-series data. The anomaly detection in each projection is done by using univariate test statistics.

Mixture of Parametric Distributions Such techniques use a mixture of parametric statistical distributions to model the data. Techniques in this category can be grouped into two sub-categories. The first sub-category of techniques model the normal instances and anomalies as separate parametric distributions, while the second sub-category of techniques model only the normal instances as a mixture of parametric distributions. For the first sub-category of techniques, the testing phase involves determining which distribution normal or anomalous the test instance belongs to. A test instance is tested using the Grubb's test on both distributions, and accordingly labeled as normal or anomalous. Similar techniques have been proposed in[21] and [3]. [21] use Expectation Maximization (EM) algorithm to develop a mixture of models for the two classes, assuming that each data point is an anomaly with apriori probability λ, and normal with apriori probability $1 - \lambda$. Thus, if $\mathbf{D}$

represents the actual probability distribution of the entire data, and **M** and **A** represent the distributions of the normal and anomalous data respectively, then

$$\mathbf{D} = \lambda\mathbf{A} - (1 - \lambda)\mathbf{M} \tag{3.6}$$

M is learnt using any distribution estimation technique, while **A** is assumed to be uniform. Initially all points are considered to be in **M**. The anomaly score is assigned to a point based on how much the distributions change if that point is removed from **M** and added to **A**. The second sub-category of techniques model the normal instances as a mixture of parametric distributions. A test instance which does not belong to any of the learnt models is declared to be anomaly. Gaussian mixture models have been mostly used for such techniques [2], and have been used to detect strains in airframe data, to detect anomalies in mammographic image analysis and for network intrusion detection.

Non-Parametric

The anomaly detection techniques in this category use non-parametric statistical models, such that the model structure is not defined a priori, but is instead determined from given data. Such techniques typically make fewer assumptions regarding the data, such as smoothness of density, when compared to parametric techniques.

Histogram The simplest non-parametric statistical technique is to use histograms to maintain a profile of the normal data. Such techniques are also referred to as frequency based or counting based. Histogram based techniques are particularly popular in intrusion detection community and fraud detection, since the behavior of the data is governed by certain profiles (user or software or system) that can be efficiently captured using the histogram model[21][22]. A basic histogram based anomaly detection technique for univariate data consists of two steps. The first step involves building a histogram based on the different values taken by that feature in the training data. In the second step, the technique checks if a test instance falls in any one of the bins of the histogram. If it does, the test instance is normal, otherwise it is anomalous. A variant of the basic histogram based technique is to assign an anomaly score to each test instance based on the height (frequency)

of the bin in which it falls. The size of the bin used when building the histogram is key for anomaly detection. If the bins are small, many normal test instances will fall in empty or rare bins, resulting in a high false alarm rate. If the bins are large, many anomalous test instances will fall in frequent bins, resulting in a high false negative rate. Thus a key challenge for histogram based techniques is to determine an optimal size of the bins to construct the histogram which maintains low false alarm rate and low false negative rate. Histogram based techniques require normal data to build the histograms. For multivariate data, a basic technique is to construct attribute-wise histograms. During testing, for each test instance, the anomaly score for each attribute value of the test instance is calculated as the height of the bin that contains the at- tribute value. The per-attribute anomaly scores are aggregated to obtain an overall anomaly score for the test instance. The basic histogram based technique for multivariate data has been applied to system call intrusion detection, network intrusion detection, fraud detection, damage detection in structures, detecting web-based attacks and anomalous topic detection in text data. The SRI International's real-time Network Intrusion Detection System (NIDES) [53] has a sub-system that maintains long-term statistical profiles to capture the normal behavior of a computer system. The authors propose a Q statistic to compare a long-term profile with a short term profile (observation). The statistic is used to determine another measure called S statistic which reflects the extent to which the behavior in a particular feature is anomaly with respect to the historical profile. The feature-wise S statistics are combined to get a single value called IS statistic which determines if a test instance is anomalous or not.

Kernel Function A non-parametric technique for probability density estimation is parzen windows estimation . This involves using kernel functions to approximate the actual density. Anomaly detection techniques based on kernel functions are similar to parametric methods described earlier. The only difference is the density estimation technique used. To detect anomalies which uses kernel functions, the probability distribution function (pdf) for the normal instances have to be estimated. A new instance which lies in the low probability area of this pdf is declared to be anomalous.

Computational Complexity

The computational complexity of statistical anomaly detection techniques depends on the

nature of statistical model that is required to be fitted on the data. Fitting single parametric distributions from the exponential family, e.g., Gaussian, Poisson, Multinomial, etc., is typically linear in data size as well as number of attributes. Fitting complex distributions (such as mixture models, Hidden Markov Models(HMM) , etc.) using iterative estimation techniques such as Expectation Maximization (EM), are also typically linear per iteration, though they might be slow in converging depending on the problem and/or convergence criterion. Kernel based techniques can potentially have quadratic time complexity in terms of the data size.

Advantages and Disadvantages of Statistical Techniques

The advantages of statistical techniques are: (1) If the assumptions regarding the underlying data distribution hold true, statistical techniques provide a statistically justifiable solution for anomaly detection. (2) The anomaly score provided by a statistical technique is associated with a confidence interval, which can be used as additional information while making a decision regarding any test instance. (3) If the distribution estimation step is robust to anomalies in data, statistical techniques can operate in a unsupervised setting without any need for labeled training data.

The disadvantages of statistical techniques are: (1) The key disadvantage of statistical techniques is that they rely on the assumption that the data is generated from a particular distribution. This assumption often does not hold true, especially for high dimensional real data sets. (2) Even when the statistical assumption can be reasonably justified, there are several hypothesis test statistics that can be applied to detect anomalies; choosing the best statistic is often not a straightforward task. In particular, constructing hypothesis tests for complex distributions that are required to fit high dimensional data sets is nontrivial. (3) Histogram based techniques are relatively simple to implement, but a key shortcoming of such techniques for multivariate data is that they are not able to capture the interactions between different attributes. An anomaly might have attribute values that are individually very frequent, but their combination is very rare, but an attribute-wise histogram based technique would not be able to detect such anomalies.

3.3.5 Information Theory-based Outlier Method

Information theoretic techniques analyze the information content of a data set using different information theoretic measures such as Kolomogorov Complexity, entropy, relative entropy, etc. Such techniques are based on the following key assumption:
Anomalies in data induce irregularities in the information content of the data set.
Let $C(D)$ denote the complexity of a given data set, D. A basic information theoretic technique can be described as follows. Given a data set D, find the minimal subset of instances, I, such that $C(D) - C(D - I)$ is maximum. All instances in the subset thus obtained, are deemed as anomalous. The problem addressed by this basic technique is to find a pareto-optimal solution, which does not have a single optima, since there are two different objectives that need to be optimized. In the above described technique, the complexity of a data set (C) can be measured in different ways. Kolomogorov complexity [Li and Vitanyi 1993] has been used by several techniques Keogh et al. 2004]. Arning et al. [1996] use the size of the regular expression to measure the Kolomogorov Complexity of data (represented as a string) for anomaly detection. [40] use the size of the compressed data file (using any standard compression algorithm), as a measure of the data set's Kolomogorov Complexity. Other information theoretic measures such as entropy, relative uncertainty, etc., have also been used to measure the complexity of a categorical data set [31]. The basic technique described above, involves dual optimization to minimize the subset size while maximizing the reduction in the complexity of the data set. Thus an exhaustive approach in which every possible subset of the data set is considered would run in exponential time. Several techniques have been proposed that perform approximate search for the most anomalous subset. [31] [33] use an approximate algorithm called Local Search Algorithm (LSA) to approximately determine such a subset in a linear fashion, using entropy as the complexity measure. Information theoretic techniques have also been used in data sets in which data instances are naturally ordered, e.g., sequential data, spatial data. In such cases, the data is broken into substructures (segments for sequences, subgraphs for graphs, etc.), and the anomaly detection technique finds the substructure, I, such that $C(D) - C(D - I)$ is maximum. This technique has been applied to sequences, graph data and spatial data. A key challenge of such techniques is to find the optimal size of the substructure which

would result in detecting anomalies.

Computational Complexity

As mentioned earlier, the basic information theoretic anomaly detection technique has exponential time complexity, though approximate techniques have been pro- posed that have linear time complexity.

Advantages and Disadvantages of Information Theoretic Techniques

The advantages of information theoretic techniques are as follows: (1) They can operate in an unsupervised setting. (2) They do not make any assumptions about the underlying statistical distribution for the data.

The disadvantages of information theoretic techniques are as follows: (1) The performance of such techniques is highly dependent on the choice of the information theoretic measure. Often, such measures can detect the presence of anomalies only when there are significantly large number of anomalies present in the data. (2) Information theoretic techniques applied to sequences and spatial data sets rely on the size of the substructure, which is often nontrivial to obtain. (3) It is difficult to associate an anomaly score with a test instance using an information theoretic technique.

3.3.6 Spectral-based Method

Spectral techniques try to find an approximation of the data using a combination of attributes that capture the bulk of variability in the data. Such techniques are based on the following key assumption: Assumption: Data can be embedded into a lower dimensional subspace in which normal instances and anomalies appear significantly different. Thus the general approach adopted by spectral anomaly detection techniques is to determine such subspaces (embeddings, projections, etc.) in which the anomalous instances can be easily identified [5]. Such techniques can work in an unsupervised as well as semi-supervised setting. Several techniques use Principal Component Analysis (PCA) for projecting data into a lower dimensional space. One such technique [52] analyzes the projection of each data instance along the principal components with low variance. A normal instance that satisfies the correlation structure of the data will have a low value for such projections while an anomalous instances that deviates from the correlation structure will have a large

value. [35] propose a spectral technique to detect anomalies in a time series of graphs. Each graph is represented as an adjacency matrix for a given time. At every time instance, the principle component of the matrix is chosen as the activity vector for the given graph. The time-series of the activity vectors is considered as a matrix and the principal left singular vector is obtained to capture the normal dependencies over time in the data. For a new (test) graph, then angle between its activity vector and the principal left singular vector obtained from the previous graphs is computed and used to determine the anomaly score of the test graph. In a similar approach, [70] propose an anomaly detection technique on a sequence of graphs by performing Compact Matrix Decomposition (CMD) on the adjacency matrix for each graph and thus obtaining an approximation of the original matrix. For each graph in the sequence, the authors perform CMD and compute the approximation error between the original adjacency matrix and the approximate matrix. The authors construct a time series of the approximation errors and detect anomalies in the time series of errors; the graph corresponding to anomalous approximation error is declared to be anomalous. [64] present an anomaly detection technique where the authors perform robust PCA to estimate the principal components from the covariance matrix of the normal training data. The testing phase involves comparing each point with the components and assigning an anomaly score based on the point's distance from the principal components. Thus if the projection of x on the principal components are $y_1, y_2, ..., y_p$ and the corresponding eigenvalues are $\lambda_1, \lambda_2, ..., \lambda_p$

$$\sum_{i=1}^{q} \frac{y_i^2}{\lambda_i} = \frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} + \ldots + \frac{y_q^2}{\lambda_q}, \;\; q \leq p \tag{3.7}$$

has a chi-square distribution. Using this result, the authors propose that, for a given significance level , observation x is an anomaly if

$$\sum_{i=1}^{q} \frac{y_i^2}{\lambda_i} > \chi_q^2(\alpha) \tag{3.8}$$

It can be shown that the quantity calculated in Equation 3.7 is equal to the Maha-

lanobis distance of the instance x from the sample mean (See Equation 3.4) when $q = p$. Thus the robust PCA based technique is same as a statistical technique discussed in Section in a smaller subspace. The robust PCA based technique has been applied to the network intrusion detection domain and for detecting anomalies in space craft components [43]. Computational Complexity Standard PCA based techniques are typically linear in data size but often quadratic in the number of dimensions. Non linear techniques can improve the time complexity to be linear in the number of dimensions but polynomial in the number of principal components. Techniques that perform SVD on the data typically quadratic in data size.

Advantages and Disadvantages of Spectral Techniques

The advantages of spectral anomaly detection techniques are as follows: (1) Spectral techniques automatically perform dimensionality reduction and hence are suitable for handling high dimensional data sets. Moreover, they can also be used as a pre-processing step followed by application of any existing anomaly detection technique in the transformed space. (2) Spectral techniques can be used in an unsupervised setting.

The disadvantages of spectral anomaly detection techniques are as follows: (1) Spectral techniques are useful only if the normal and anomalous instances are separable in the lower dimensional embedding of the data. (2) Spectral techniques typically have high computational complexity.

3.4 Summary

As we describe all the above techniques has advantages and disadvantages. An ideal algorithm will be this which will keep advantages and eliminate disadvantages. The algorithms from the same domain is difficult to put across this goal. Therefore, there is a thought of permuting techniques from different domains in order to keep the advantages of each approach and sweep away its disadvantages. We select to combine the statistical area with nearest neighbor area. Our decision is based on the following strategy. Firstly, we separate data in sliding windows and for each attribute we estimate approximately the distribution of data in the window. Moreover, we compute the bandwidth of kernel density estimation

for each window. For the nearest neighbor approach, we aim to eliminate the problems such as when normal instances do not have enough close neighbors or anomalies have enough close neighbors by computing a density scoring function for those data points which lies in the low probability area of the estimating distribution's pdf. These data points are computed in the statistical part. To conclude, we apply an incremental strategy on the computation of density scoring function by way of decreasing the execution time of the nearest neighbor approach.

Chapter 4

Outlier Detection over Data Streams

In recent years, data in many rare events applications (e.g. stock market, network traffic monitoring, video surveillance, web usage logs) arrives continuously at an enormous pace thus posing a significant challenge to analyze it. In such cases, it is important to make decisions quickly and accurately. If there is a sudden or unexpected change in the existing behavior, it is essential to detect this change as soon as possible. Assume, for example, there is a computer in the local area network that uses only limited number of services (e.g., Web traffic, telnet, ftp) through corresponding ports. All these services correspond to certain types of behavior in network traffic data. If the computer suddenly starts to utilize a new service (e.g., sash), this will certainly look like a new type of behavior in network traffic data. Hence, it will be desirable to detect such behavior as soon as it appears especially since it may very often correspond to illegal or intrusive events. Even in the case when this specific change in behavior is not necessary intrusive or suspicious, it is very important for a security analyst to understand the network traffic and to update the notion of the normal behavior.Automated identification of suspicious behavior and objects based on information extracted from video streams is currently an active research area. Other potential applications include traffic control and surveillance of commercial and residential buildings. These tasks are characterized by the need for real-time processing and by dynamic, non-stationary and often noisy environment. Hence, there is necessity for incremental outlier detection that can adapt to novel behavior and provide timely identification of unusual events.

4.1 Incremental Local Outlier Detection over Data Streams

4.1.1 Introduction

In this study [54], they propose an incremental outlier detection algorithm based on computing the densities of local neighborhoods. The main idea of the *LOF* algorithm [12] is to assign to each data record a degree of being outlier. This degree is called the *local outlier factor* (LOF) of a data record. Data points with high LOF have local densities smaller than their neighborhood and typically represent stronger outliers, unlike data points belonging to uniform clusters that usually tend to have lower LOF values. The algorithm for computing the LOF for all data records has the following steps:

1. For each data record q compute *k_distance(q)* as the distance to the k_{th} nearest neighbor of q

2. Compute reachability distance for each data record q with respect to data record p as follows:

$$reachdist_k(q,p) = max(d(q,p), k\_distance(p)) \tag{4.1}$$

   where $d(q,p)$ is Euclidean distance from q to p.

3. Compute local reachability density (lrd) of data record q as the inverse of the average reachability distance based on the k nearest neighbors of the data record q (In the original LOF publication [12], parameter k was named $MinPts$).

$$lrd(q) = \frac{1}{\sum_{p \in kNN(q)} reachdist_k(q,p)/k} \tag{4.2}$$

4. Compute LOF of data record q as the ratio of average local reachability density of qs k nearest neighbors and local reachability density of the data record q.

$$LOF(q) = \frac{\frac{1}{k} \sum_{p \in kNN(q)} lrd(p)}{lrd(q)} \tag{4.3}$$

The main advantages of LOF approach over other outlier detection techniques include:

- It detects outliers with respect to density of their neighboring data records; not to the global model.
- It is able to detect outliers regardless the data distribution of normal behavior, since it does not make any assumptions about the distributions of data records.

In order to fully justify the need for incremental outlier detection techniques, it is important to understand that applying standard LOF outlier detection algorithms to data streams would be extremely computationally inefficient and very often may lead to incorrect results.

4.1.2 Methodology

Their proposed incremental LOF algorithm is designed to provably provide the same prediction performance (detection rate and false alarm rate) as the standard LOF. It is achieved by consistently maintaining for all existing points in the database the same LOF values as the standard LOF algorithm. Their proposed incremental LOF algorithm has time complexity $O(NlogN)$ thus clearly outperforming the standard LOF approach which has $O(N^2logN)$. After all N data records are inserted into the data set, the final result of the incremental LOF algorithm on N data points is independent of the order of insertion. To design an incremental LOF algorithm, they have been motivated by two goals. Firstly, the result of the incremental algorithm must be equivalent to the result of the standard algorithm every time t a new point is inserted into a data set. Second, asymptotic time complexity of incremental LOF algorithm has to be comparable to the standard LOF algorithm. In order to have feasible incremental algorithm, it is essential that, at any time moment t, insertion/deletion of the data record results in limited number of updates of algorithm parameters. Specifically, the number of updates per each insertion/deletion must not be dependent on the current number of records in the dataset; otherwise, the

performance of the incremental LOF algorithm would be $\Omega(N^2)$ where N is the final size of the dataset. The proposed incremental LOF algorithm computes LOF value for each data record inserted into the data set and instantly determines whether inserted data record is outlier. In addition, LOF values for existing data records are updated if needed.

Insertion of Data Points

In the insertion part, the algorithm performs two steps:

- Insertion of new record, when it computes reach-dist, lrd and LOF values of a new point
- Maintenance, when it updates k_distances, reach-dist, lrd and LOF values for affected existing points.

Therefore, let illustrate these steps through the example of inserting a new data point n into a data set. Insertion of the point n may decrease the k_distance of certain neighboring points, and it can happen only to those points that have the new point n in their k-neighborhood. Hence, we need to determine all such affected points.

According to 4.1, when k_distance(p) changes for a point p, reach-dist(q,p) will be affected only for points q that are in k-neighborhood of the point p. According to 4.2, lrd value of a point q is affected if: a) the k-neighborhood of the point q changes or b) reach-dist from point q to one of its k-neighbors changes. The 2-neighborhood of a point will change only if the new point n becomes one of its 2-neighbors. Hence, we need to update lrd on all points to which the point n is now one of their 2-neighbors and on all points q where reach-dist(q,p) is updated and p is among 2-nearest neighbors of q . According to 4.3, LOF values of an existing point q should be updated if lrd(q) is updated or lrd(p) of one of its 2-neighbors p changes.

The general framework for the incremental LOF method is shown in 4.1. As in the standard LOF algorithm [12], we define k_{th} nearest neighbor of a record p as a record q from the dataset S such that for at least k records $o' \in S \backslash p$ it holds that $d(p, o') \leq d(p, q)$, and for at most k-1 records $o' \in S \setminus p$ holds that $d(p, o') \leq d(p, q)$, where d(p,q) denotes Euclidean distance between data records p and q. We refer d(p,q) as k_distance(p). k nearest

*Incremental LOF*_insertion(*Dataset S*)
- Given: Set S $\{p_1, \ldots, p_N\}$ $p_i \in R^D$, where D corresponds to the dimensionality of data records.
- For each data point p_c in data set S
  - insert(p_c)
  - *Compute* kNN(p_c)
  - ($\forall p_j \in$ kNN(p_c))
    compute *reach-dist*$_k(p_c, p_j)$ using Eq. (1);

  //Update_neighbors of p_c
  - $S_{update\_k\_distance}$ =kRNN(p_c);
  - ($\forall p_j \in S_{update\_k\_distance}$)
    update *k-distance*(p_j) using Eq.(5);
  - $S_{update\_lrd} = S_{update\_k\_distance}$;
  - ($\forall p_j \in S_{update\_k\_distance}$), ($\forall p_i \in$kNN($p_j$)\{$p_c$})
    reach-dist$_k(p_i, p_j)$ =k-distance(p_j);
    if $p_j \in$ kNN(p_i)
    $S_{update\_lrd} = S_{update\_lrd} \cup \{p_i\}$;
  - $S_{update\_LOF} = S_{update\_lrd}$;
  - ($\forall p_m \in S_{update\_lrd}$)
    update *lrd*(p_m) using Eq. (2);
    $S_{update\_LOF} = S_{update\_LOF} \cup$ kRNN(p_m);
  - ($\forall p_1 \in S_{update\_LOF}$)
    update *LOF*(p_1) using Eq.(3);
  - compute lrd(p_c) using Eq.(2);
  - compute LOF(p_c) using Eq.(3);
- End //for

Figure 4.1: The general framework for insertion of data record and computing its LOF value in incremental LOF algorithm.

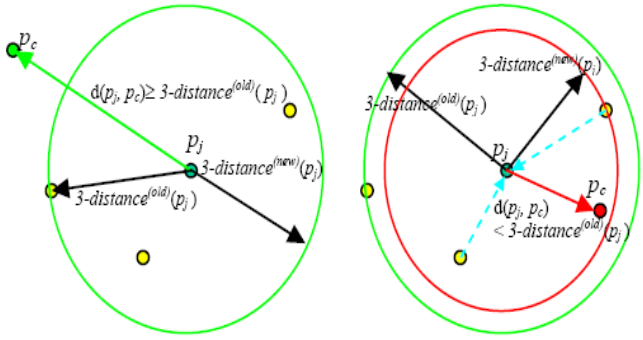

Figure 4.2: Update of k-nearest neighbor distance upon insertion of a new record

neighbors (referred to as kNN(p)) include all points $r \in S \setminus p$ such that $d(p,r) \leq d(p,q)$. We also define k reverse nearest neighbors of p (referred to as kRNN (p)) as all points q for which p is among their k nearest neighbors. For a given data record p, kNN(p) and kRNN(p) can be respectively retrieved by executing nearest-neighbor and reverse nearest neighbor queries on a dataset S.

The insertion of point p_c affects the k_distance at points p_j that have point p_c in their k-nearest neighborhood. New k_distances of the affected points p_j are updated as follows:

$$kdistance^{(new)}(p_j) = \begin{cases} d(p_j, p_c), & p_c \text{ is the } k^{th} \text{ nearest neighbor of } p_j \\ (k-1)distance^{(old)}(p_j), & \text{otherwise} \end{cases} \tag{4.4}$$

Deletion of Data Points

In data stream applications it is sometimes necessary to delete certain data records. Very often, not only a single data record is deleted from the data set, but the entire data block that may correspond to particular outdated behavior. Similarly like in an insertion, upon deleting the block of data records S_{delete}, there is a need to update parameters of the incremental LOF algorithm. The general framework for deleting the block of data records S_{delete} from the dataset S is given in 4.3. The deletion of each record $p_c \in S_{delete}$ from dataset S influences the kdistances of its kRNN. k-neighborhood increases for each data record pj that is in reverse k-nearest neighborhood of pc. For such records, k_distance(p_j) becomes equal to the distance from pj to its new k_{th} nearest neighbor. The reachability distances from $p'_j s$ (k-1) nearest neighborsp_i to p_j need to be updated.

Observe that the reachability distance from the k_{th} neighbor of p_j to record p_j is already equal to their Euclidean distance $d(p_i, p_j)$ and does not need to be updated 4.3. Analog to insertion, lrd value needs to be updated for all points p_j where k_distance is updated. In addition, lrd value needs to be updated for points p_i such that p_i is in kNN of p_j and p_j is in kNN of p_i. Finally, LOF value is updated for all points p_m on which lrd value is updated as well as on their kRNN.

Incremental LOF_deletion(Dataset S, S_{delete})

- $S_{update\_k\_distance}=\emptyset$;
- $(\forall p_c \in S_{delete})$
  $S_{update\_k\_distance} = S_{update\_k\_distance} \cup kRNN(p_c)$;
  delete(p_c); //we can delete p_c after finding
  // all reverse neighbors
- $S_{update\_k\_distance} = S_{update\_k\_distance} \setminus S_{delete}$; //points from S_{delete} may still be present when computing reverse k-nearest neighbors
- $(\forall p_j \in S_{update\_k\_distance})$
  update k-distance(p_j);
- $S_{update\_lrd} = S_{update\_k\_distance}$;
- $(\forall p_j \in S_{update\_k\_distance})$ $(\forall p_i \in (k\text{-}1)NN(p_j))$
  $reach\text{-}dist_k(p_i,p_j)$=k-distance($p_j$);
  if $p_j \in kNN(p_i)$
  $S_{update\_lrd} = S_{update\_lrd} \cup \{p_i\}$;
- $S_{update\_LOF} = S_{update\_lrd}$;
- $(\forall p_m \in S_{update\_lrd})$
  update $lrd(p_m)$ using Eq. (2);
  $S_{update\_LOF} = S_{update\_LOF} \cup kRNN(p_m)$;
- $(\forall p_l \in S_{update\_LOF})$
  update $LOF(p_l)$ using Eq.(3);

return

Figure 4.3: The framework for deletion of data record in incremental LOF method.

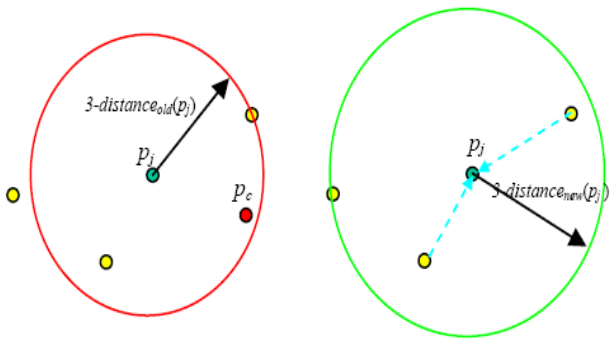

Figure 4.4: Update of k-nearest neighbor distance upon deletion of record p_c

4.2 Frequent Pattern Based Outlier Detection over Data Streams

4.2.1 Introduction

Let $I = \{i_1, i_2, ..., i_m\}$ be a set of m literals called *items*. Let the database $D = \{t_1, t_2, ..., t_n\}$ be a set of n transactions each consisting of a set of items from I. An itemset X is a non-empty subset of I. The length of itemset X is the number of items contained in X, and X is called a *k-itemset* if its length is k. A transaction $t \in D$ is said to contain an itemset X if $X \subseteq t$. The support of an itemset X is the percentage of transactions in D containing X:

$$support(X) = \|\{t \in D \,|\, x \subseteq t\}\| \;/\; \|\{t \in D\}\| \tag{4.5}$$

The problem of finding all *frequent itemsets* in D is then defined as follows. Given user defined threshold for the permissible minimal support, find all itemsets with support greater or equal to *minisupport*. Frequent itemsets are also called *frequent patterns*. From the viewpoint of knowledge discovery, frequent patterns reflect the "common patterns" that apply to many objects, or to large percentage of objects, in the dataset. In contrast, outlier detection focuses on a very small percentage of data objects.

- *FPOF-Frequent Pattern Outlier Factor*

  Let the database $D = \{t_1, t_2, ..., t_n\}$ be a set of n transactions with *items I*. Given threshold *minisupport*, the set of all frequent patterns is donated as: *FPS (D, minisupport)*. For each transaction t, the *Frequent Pattern Outlier Factor of t* is defined as:

$$FPOF(t) = \frac{\sum_{X} support(X)}{\|FPS(D,\, minisupport)\|} \;,\; \text{where}\, X \subseteq t \,\text{and}\, X \in FPS(D,\, minisupport) \tag{4.6}$$

  The interpretation of 4.6 is: If a data object contains more frequent patterns, its $FPOF$ value will be larger, which indicates that it is unlikely to be an outlier. In

contrast, objects with smaller $FPOF$ values will have greater outlyingnesses. In addition, the $FPOF$ value is between 0 and 1.

- For each transaction t, the itemset X is said to be contradict to t if $X \not\subset t$. The contradict-ness of X to t is defined as:

$$Contradictness(X,t) = (\|X\| - \|t \cap X\|) * support(X) \tag{4.7}$$

In their approach [84], the frequent pattern outlier factor given in 4.6 is used as the basic measure for identifying outliers. To describe the reasons why identified outliers are abnormal, the itemsets those are not contained in the transaction (it is said that the itemset is contradict to the transaction) are good candidates. The consideration behind 4.7 is as follows. First, the greater the support of the itemset X, the greater the value of *contradict-ness of* X to t since larger support value of X suggests a more strong deviation. Second, longer itemsets will give a better description than that of shorter ones. With 4.7, it is possible to identify the contribution of each *itemset* to the outlyingness of specified transaction. However, it is not feasible to list all the *contradict itemset*, and it will be preferable to present only the top k contradict frequent pattern to the end user.

- *TKCFP-Top K Contradict Frequent Pattern*

The meanings of D, I, *minisupport* and *FPS (D, minisupport)* are the same as given above. For each transaction t, the *itemset* X is said to be a *top k contradict frequent pattern* if there exist no more than *(k-1) itemsets* whose contradictness is higher than that of X, where $X \in$ *FPS(D, minisupport)*. Their task is to mine top-n outliers with regard to the value of frequent pattern outlier factor. For each identified outlier, its *top k contradict frequent patterns* will also be discovered for the purpose of description.

Algorithm *DSFind**FPOF***

Input: *DS* //the transactional data stream

minisupport //threshold for the permissible minimal support

top-n // threshold value for *top n fp-outliers*

top-k //threshold value for *top k contradict frequent patterns*

$\mathcal{E}$ //error parameter for frequent pattern

Output: *The top-n FP-outliers*

with their corresponding TKCFPs so far

```
01 begin
02      foreach transaction t in DS do begin
03          update entries of the form (e, f, maximal error )
04          compute the value of outlier factor of t
                    using current frequent patterns
05          if t is in the top n fp-outliers then
06              finds its top-k contradict frequent patterns
07              output t and its top-k contradict frequent patterns
08          end
09      end
10 end
```

Figure 4.5: The DSFindFPOF algorithm

4.2.2 Methodology

The key aspect for detecting FP-Outliers is to get the frequent item-sets. However, existing methods for frequent pattern mining require multiple passes over the datasets, which is not allowed in the data stream model. Thus, instead of finding the exact frequent patterns with multiple passes, we get the estimated frequent patterns by exploring approximation counts technique over data streams developed by Manku and Motwani [47]. The algorithm in [84] DSFindFPOF for detecting outliers from data stream is listed in 4.5. For each transaction in the stream, the value of FPOF is computed (Step 2-4). Then, updating the top-n FP-outliers with their corresponding top-k contradict frequent patterns (Step 5-8).

4.3 Online Cluster Based Outlier Detection over Data Streams

4.3.1 Introduction

Given a data space in multiple dimensions $A_1, A_2, ..., A_m$ with domains $D_1, D_2, ..., D_m$ respectively, let the data stream D be a sequence of data objects, where each data object $t \in D_1 \times ... \times D_m$. Their task is to online detect if a new coming data is an outlier. The basic idea in their approach is as follows. Firstly, it 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. They propose that "the smaller a cluster is, 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. 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

4.3.2 Methodology

In their approach [16], 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 cannot be dynamically changed. Whenever a new data stream 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 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. Thus to save memory, a group of adjacent large data clusters can be merged into one big cluster. For each cluster, they maintain a statistic summary S, which is defined as follows:

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

- 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 stream 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. With this grid-based clustering algorithm, they can easily find and merge the adjacent dense clusters by simply searching its neighbor 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 dimension increase, the number of cells increases dramatically and the number of cluster may also increase. Moreover, they propose a new cluster-based definition for outliers and a novel measure for outlier degree.

Definition 3: *The data objects which do not lie in large clusters are outliers*

Definition 4: Let $C = \{C_1, C_2, ..., C_h\}$ is the set of clusters in the ordering of $\mid C_1 \mid \geq \mid C_2 \mid \geq ... \geq \mid C_h \mid$. Given two pre-specified parameters α and β, they find the minimal d as the boundary such that it can first satisfy condition 1 and then also satisfy condition 2.

Condition 1: $\mid C_1 \mid + \mid C_2 \mid + ... + \mid C_d \mid \geq \alpha \mid D \mid$

Condition 2: $\frac{(|C_1|+|C_2|+...+|C_d|)/d}{(|C_{d+1}|+|C_{d+2}|+...+|C_h|)/(h-d)} \geq \beta$

Then the set of large *Data Cluster(DC)* = $\{C_1, C_2, ..., C_d\}$ and the set of small *Outlier Cluster(OC)* = $\{C_{d+1}, ..., C_h\}$.Therefore, all the data objects in OC are outliers. The above definition consifers two heuristics:

- Outliers are just a small number of data objects and most data objects are not outliers
- The average size of clusters in DC should also be much bigger than that of clusters in OC.

To describe an outlier cluster, someone must 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 labeled as outliers. To avoid this, they predict those data objects in the top-k outlier cluster as outliers. Therefore, there is need for an outlier degree.

Definition 5: For any two clusters C_i and C_j, the *distance(C_i, C_j)* is defined as:

$$distance(C_i, C_j) = min\{distance(r_s, r_t)\} \tag{4.8}$$

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

Definition 6: Based on the above definitions, for each cluster $C_i \in OC$, the Outlier Degree(OD) of C_i is defined as following:

$$OD(C_i) = \frac{min\{distance(C_i\, C_j)\}}{\|C_i\|}\ ,\ \text{where}\, C_j \in DC \tag{4.9}$$

From the above definition, the smaller a cluster is, the more outlying it is and the further a cluster is from the closest larger cluster, the more outlying it is.

Since their approach is cluster-based online outlier detection , their algorithm is named as **CBOD**. 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

Input: data stream D, α, β, K, and X

Methods:

1. Initialize Top-K-Outlier_Clusters[ ] = $\emptyset$.
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.

Figure 4.6: The CBOD algorithm

clusters that data structure for clusters' summary will eventually overload the memory. When memory is full, they follow two strategies.

- Firstly, they merge adjacent data clusters into one big data cluster. Since the task is to identify outliers that usually lie in outlier clusters, merging adjacent dense data clusters will not affect the detecting accuracy.

- Secondly, either they prune all the outlier clusters in OC or write them to hard disk for users' offline analysis in the future.

The figure 4.6 describes the details of the **CBOD** algorithm. The total runtime of **CBOD** is $O(n)$, where n is the number of clusters and equals to the number of data objects in the worst case.

Chapter 5

Statistical Nearest Neighbor Outlier Stream Algorithm

5.1 Problem definition and Basic formulation

Given a data space in m dimensions $A_1, A_2, ..., A_m$ with supports $D_1, D_2, ..., D_m$ respectively, let the data stream D be a sequence of data points, where each data point $t \in D_1 \times ... \times D_m$. Our aim is to present a general framework for online outlier detection over data streams which contain numerical values. The basic component of this framework is an incremental, unsupervised and nonparametric algorithm, which detects online if there are data points in the current sliding window that must be treated as outliers. The basic idea in our approach is as follows. Firstly, we perform an online sampling over the data stream in the current sliding window in order to estimate the parameters of each distribution by using the method of Maximum Likelihood Estimation. Then, we compute probability density function of data points in the current window. Moreover, we compute the probability density function of data points in the current window by using Kernel Density Estimation. We compare the probability density functions of two methods(MLE and KDE) for three data points in the current window and the distribution whose probability density function is more close with the value of pdf from KDE is chosen as the distribution of data. Finally, we compute the lower-quantile and upper-quantile of the estimated distribution. Secondly, we propose two scoring functions which are based on the nearest neighbor algorithm. For

the first scoring function, we use the distances between the nearest neighbors for each data point and for the second we compute the densities of neighborhoods for each data point. Finally, a data point whose values exceed the thresholds of quantiles and its value from the scoring functions are far away from one for density scoring function , will be considered as an outlier.

5.2 Statistical modeling

5.2.1 Kernel Density Estimation

Kernel density estimation is a non parametric method to reveal the unknown density of a distribution by selecting a suitable density estimator[65]. Theoretically, the kernel density function $\tilde{f}_h(x)$ is guaranteed to converge to the real density $f(x)$ for an arbitrary distribution. More formally, assume that we have a data set D, containing n points whose values are $\overline{X_1}, \overline{X_2}, ..., \overline{X_n}$. We can approximate the underlying density $f(x)$ using the following kernel function (The kernel function$K(x)$, is a function of random variable X. The kernel width h, determines the smoothing level of kernel function):

$$\tilde{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} \mathcal{K}\Big(\frac{x - \overline{X_i}}{h}\Big)\,, \quad x, \overline{X_i} \in \mathbb{R}\,, \quad \int_{x \in \mathbb{R}} \mathcal{K}(x)dx = 1 \tag{5.1}$$

Commonly used kernel functions are the Epanechnikov, Gaussian and Quartic kernel. Since the Gaussian kernel is unbounded, it stiffens the cost of computing the integral. So, we choose as default kernel the Epanechnikov kernel which is a square function, easily integrated and bounded, which is given by :

$$\mathcal{K}\Big(\frac{x - \overline{X_i}}{h}\Big) = \begin{cases} \frac{3}{4} \cdot \frac{1}{h}\left(1 - \left(\frac{x-\overline{X_i}}{h}\right)^2\right), & \left|\frac{x-\overline{X_i}}{h}\right| < 1 \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

The kernel density estimation indicates that the kernel width significantly affects the shape of a kernel function. A widely used rule for approximating the kernel width is the Scotts rule [61] given by Eq.(5.3). However, these strategies depend on the complete

sample, which is impracticable in a data stream scenario. To overcome this problem, we adopt an approximate solution, by considering only the data in the sliding window. The number of sliding window (N) is larger, the approximate kernel width $\tilde{h}$ is closer to h. σ is the standard deviation of the whole data stream and $\tilde{\sigma}$ is the sample standard deviation of sliding window.

$$h \approx 2.345\sigma n^{-1.5} \approx \tilde{h} \triangleq 2.345\tilde{\sigma}(N)^{-1.5} \tag{5.3}$$

In our implementation, we have implemented four kernel density functions namely the Epanechnikov, Gaussian, Triangle, Quartic. The default function is the Epanechinkov kernel function which we use mostly in our experiments.

5.2.2 Maximum Likelihood Estimation

One approach to the problem of distribution parameter estimation is to use the samples to estimate the unknown probabilities and probability densities, and to use the resulting estimates as if they were the true values. In typical supervised pattern classification problems, the estimation of the prior probabilities presents no serious difficulties[57]. However, estimation of the class-conditional densities is another matter. The number of available samples always seems too small, and serious problems arise when the dimensionality of the feature vector x is large. If we know the number of parameters in advance and our general knowledge about the problem permits us to parameterize the conditional densities, then the severity of these problems can be reduced significantly. Suppose, for example, that we can reasonably assume that $p(x|\omega_i)$ is a normal density with mean μ_i and covariance matrix Σ_i, although we do not know the exact values of these quantities. This knowledge simplifies the problem from one of estimating an unknown function $p(x|\omega_i)$ to one of estimating the parameters μ_i and Σ_i. Maximum likelihood and several other methods view the parameters as quantities whose values are fixed but unknown. The best estimate of their value is defined to be the one that maximizes the probability of obtaining the samples actually observed. Maximum likelihood estimation methods have a number of attractive attributes. First, they almost always have good convergence properties as the number of training samples increases. Further, maximum likelihood estimation often can be simpler

than alternate methods such as Bayesian learning.

Suppose that we separate a collection of samples according to a set of classes, so that we have c sets, $D_1, ..., D_c$, with the samples in D_j drawn independently and identically according to the probability law $p(x|\omega_j)$. We say such samples are i.i.d. - independent identically distributed random variables. We assume that $p(x|\omega_j)$ has a known parametric form, and is therefore determined uniquely by the value of a parameter vector θ_j. For example, we might have $p(x|\omega_j) \sim N(\mu_j, \Sigma_j)$, where θ_j consists of the components of μ_j and Σ_j. To show the dependence of $p(x|\omega_j)$ on θ_j explicitly, we write $p(x|\omega_j)$ as $p(x|\omega_j, \theta_j)$. Our problem is to use the information provided by the training samples to obtain good estimates for the unknown parameter vectors $\theta_1, ..., \theta_c$ associated with each class. To simplify the treatment of this problem, we assume that samples in D_i give no information about θ_j if $i \neq j$ — that is, we assume that the parameters for the different classes are functionally independent. This permits us to work with each class separately, and simplify our notation by deleting indications of class distinctions. With this assumption we have c separate problems of the following form: Use a set D of training samples drawn independently from the probability density $p(x, \theta)$ to estimate the unknown parameter vector θ. Suppose that D contains n samples, $x_1, ..., x_n$. Then, since the samples were drawn independently, we have

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{k=1}^{n} p(\mathbf{x}_k|\boldsymbol{\theta}) \tag{5.4}$$

$p(D|\theta)$ is called the likelihood of θ with respect to the set of samples. The maximum likelihood estimate of θ is, by definition, the value $\hat{\theta}$ that maximizes $p(D|\theta)$. Intuitively, this estimate corresponds to the value of θ that in some sense best agrees with or supports the actually observed training samples. For analytical purposes, it is usually easier to work with the logarithm of the likelihood than with the likelihood itself. Since the logarithm is monotonically increasing, the $\hat{\theta}$ that maximizes the log-likelihood also maximizes the likelihood. If $p(D|\theta)$ is a well behaved, differentiable function of θ, $\hat{\theta}$ can be found by the standard methods of differential calculus. If the number of parameters to be set is p, then

we let θ denote the p-component vector $\theta = (\theta_1, ...\theta_p)^T$, and $\nabla\theta$ be the gradient operator

$$\nabla_{\boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial}{\partial\theta_1} \\ \vdots \\ \frac{\partial}{\partial\theta_p} \end{bmatrix} \tag{5.5}$$

We define $l(\theta)$ as the log-likelihood function

$$l(\boldsymbol{\theta}) \equiv \ln p(\mathcal{D}|\boldsymbol{\theta}) \tag{5.6}$$

We can then write our solution formally as the argument that maximizes the log-likelihood, i.e.,

$$\widehat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) \tag{5.7}$$

where the dependence on the data set D is implicit. Thus we have from Eq.(5.4)

$$l(\boldsymbol{\theta}) = \sum_{k=1}^{n} \ln p(\mathbf{x}_k|\boldsymbol{\theta}) \tag{5.8}$$

and

$$\nabla_{\boldsymbol{\theta}} l = \sum_{k=1}^{n} \nabla_{\boldsymbol{\theta}} \ln p(\mathbf{x}_k|\boldsymbol{\theta}) \tag{5.9}$$

Thus, a set of necessary conditions for the maximum likelihood estimate for θ can be obtained from the set of p equations

$$\nabla_{\boldsymbol{\theta}} l = 0 \tag{5.10}$$

A solution $\hat{\theta}$ to Eq.(5.10) could represent a true global maximum, a local maximum or minimum, or (rarely) an inflection point of $l(\theta)$. One must be careful, too, to check if the extremum occurs at a boundary of the parameter space, which might not be apparent from the solution to Eq.(5.10). If all solutions are found, we are guaranteed that one represents the true maximum, though we might have to check each solution individually (or calculate second derivatives) to identify which is the global optimum. Of course, we must notice that $\hat{\theta}$ is an estimate; it is only in the limit of an infinitely large number of

training points that we can expect that our estimate will be equal to the true value of the generating function.

5.2.3 Our proposed method for combining the KDE and MLE

In our algorithm, we employ ten commonly used continuous distribution functions in order to compare the probability density of each of them with the probability density of the kernel density estimation method. The intuition behind the proposed method is:
The distribution function whose probability density is closer to the probability density of the kernel density estimation will be considered as the approximated distribution
Because of the fact that a strategy which will make a comparison for each data point in the sliding window in order to find approximately the distribution will be consuming, we decided to compare only the probability densities of three data points which placed at the beginning , middle and end of data stream in the window. The ten distributions which we use to our system are the following: ChiSquare, Exponential, ExtremeValue, Gamma, Logistic, LogNormal, Normal(Gaussian), Pareto, Power, Weibull . In our implementation, for the computation of the probability density and quantiles for each distribution, we make use of a Java library which is called **SSJ** and especially the package *prodist* of this library. **SSJ** [45] is a Java library for stochastic simulation, developed in the Département d'Informatique et de Recherche Opérationnelle (DIRO), at the Université de Montréal. It provides facilities for generating uniform and nonuniform random variants, computing different measures related to probability distributions, performing goodness-of-fit tests, applying quasi- Monte Carlo methods, collecting statistics (elementary), and programming discrete-event simulations with both events and processes. Additional Java packages are also developed on top of **SSJ** for simulation applications in finance, call centers management, communication networks, etc. The package of (*SSJ*) **probdist** contains a set of Java classes providing methods to compute mass, density, distribution, complementary distribution, and inverse distribution functions for many continuous probability distributions, as well as estimating the parameters of these distributions.

5.3 Nearest Neighbor

The outlierness of an object typically appears to be more outstanding with respect to its local neighborhood. In view of this, recent work on outlier detection has been focused on finding local outliers, which are essentially objects that have significantly lower density than their local neighborhood [12]. As an objective measure, the degree of outlierness of an object p is defined to be the ratio of its density over the average density of its neighboring objects. To quantify the p's neighboring objects, users must specify a value k, and the neighboring objects are defined as the objects which are not further from p than p's k_{th} nearest objects. The existing outlierness measures are not easily applicable to a complex situation in which the dataset contains multiple neighborhoods with very different densities. The reason for the above problem lies in the inaccurate estimation for the density of an object's neighborhood.

5.3.1 Density-Based Outlierness

The major drawback of LOF algorithm [12] lies in computing reachability distances defined as $reachdist_k(q, o) = max\{kdistance(o), distance(q, o))\}$. Computing reachability distance of q involves computing distances of all data points within $q's$ neighborhood, and each compared with the k-distance of that neighborhood which is very expensive when $MinPts$ is large. Secondly, LOF has to be computed for every object before the few outliers are detected. This is not desirable since outliers constitute only a small fraction of the entire dataset.

Density Neighborhood Outlierness (DNO)

In our work, we propose a novel and efficient density scoring function which is based on the idea of LOF. This density scoring function aims to decrease the computation time for outliers and has approximately the same accuracy with the LOF algorithm. The density function we consider for the neighborhood of a data point q is defined as the inverse of the average distances between the data point q and its nearest neighbors

$$DensityNeighborhood(q) = \frac{1}{\frac{1}{|NN_k(q)|}\sum_{i \in NN_k(q)} d(q, NN_k(i))} \tag{5.11}$$

The distance in the Eq.(5.11) $d(q, NN_k(i)$ is the Euclidean distance between the data point q and each data point from its neighborhood. The $DensityNeighborhood(q)$ measures the concentration of data points around a data point q. The data points with low densities have high potential of being outliers and vise versa.

To detect the outlier we assign an outlying scoring function DNO (Density Neighborhood Outlierness) for each data point. The DNO for a each data point q in the window is given by the following formula.

$$DNO(q) = \sum_{p \in NN_k(q)} \frac{DensityNeighborhood(p)}{DensityNeighborhood(q)} \tag{5.12}$$

In DNO, the numerator represents the local density of the neighborhood of the data points p which belong in the neighborhood of q, while the denominator represents the local density of the neighborhood of the data point q which was computed by Eq.(5.11) in the sliding window. A low local DNO indicates that the neighborhood around a data point is crowded and hence a lower potential of being an outlier whereas a high local DNO indicates a not crowded neighborhood and hence a lower outlying potential.

5.3.2 Incremental Strategy

The intuition behind our incremental strategy is to identify the change of behavior of our data points and reduce the computation time. By this, we mean that only the data points for which their neighborhood changes by moving from one window to the next sliding window. To design our incremental $SNNOS$ algorithm, we have been motivated by two aims. Firstly, the result of our incremental algorithm must be approximately equivalent to the result of $ILOF$ every time new data points are inserted into the sliding window. Second, the time complexity of the incremental approach has to be comparable to $ILOF$. In order to have a feasible incremental algorithm, it is essential that, at any time instant, insertion/deletion of the data points results in a limited number of updates

of the algorithm parameters. Specifically, the number of updates per insertion/deletion must not be dependent on the current number of data points in the window; otherwise, the performance of the incremental approach would be $\Omega(N^2)$ where N is the size of the window. The incremental strategy is separated into two parts, namely, the insertion part and the deletion part. In the next subsections, we perform the analysis of incremental update when data points either inserted into or removed from the sliding window.

Update on Insertion of new data points

In the insertion part, the algorithm proceeds in two steps:

- Insertion of new data points x, when it computes $DNO(x)$
- Maintenance, when it updates $DNO(x)$ for affected existing points.

Insertion of new data points may either increase or decrease the $DensityNeighborhood(x)$ of a data point x, and it can happen only to those points that have one or more new points in their neighborhood. Hence, we need to detect all such affected points and update their scoring function $DNO(x)$.

Update on Deletion of new data points

In the deletion part, the algorithm performs only one step:

- Maintenance, when it updates $DNO(x)$ for affected existing points.

Deletion of data points from the current sliding window may either increase or decrease the $DensityNeighborhood(x)$ of a data point x, and it can happen only to those points that have one or more points to be removed from their neighborhood. Hence, we need to detect all such affected points and update their scoring function $DNO(x)$.

5.4 Statistical Nearest Neighbor Outlier Stream Algorithm

In this section we present the overall framework of our proposed method. The inputs of our algorithm are the Data Stream DS,the number of neighbors k and a user-defined threshold

value for the scoring function *denthr*.

Complexity analysis of the SNNOS Algorithm

The complexity of our algorithm $SNNOS$ is computed as following: Let denote N the number of data points in the sliding window and d the dimension for each data point. Moreover, let denote K the number of sliding windows used for processing data. The Statistical modeling part requires $O(K \cdot N \cdot d)$ operations. The Nearest Neighbor part requires $O(K \cdot N \cdot logN)$ operations, from which $O(logN)$ are required for the computation of nearest neighbors.The computation of nearest neighbors requires $O(logN)$ operations because we use an efficient nearest neighbor search algorithm with R*trees [11] as index. For the implementation of R*trees and the nearest neighbor search algorithm we use source code from the implementation of *ELKI* [1]. The insertions and deletions using as index R*trees require $O(logN)$ operations for the incremental update. In details, let T_{NN}, T_{ins} and T_{del} correspond to time needed for the nearest neighbor searching, insertion and deletion of a data point into/from the sliding window, including index updating. Due to the fact that we use an efficient indexing structure(R*trees) for inserting/deleting the data point the time complexities are as follows: $T_{NN} = T_{ins} = T_{del} = O(logN)$. Consequently, the time complexity of nearest neighbor part of our algorithm ($SNNOS$) is logarithmic in the current size N of the time-sliding window. The overall time complexity of the second part after all updates to the sliding window of size N are applied is $O(K \cdot NlogN)$. Therefore, the overall complexity of our algorithm is $O(K \cdot N(d + logN))$.

Algorithm 1: Statistical Nearest Neighbor Outlier Stream (SNNOS)

input : **DS** Data Stream , **k** nearest neighbors, **denthr** threshold for density scoring function
output: A list with Outlier Data Points

foreach *(dp (data point) ∈ DS)* **do**
First Part: Statistical Modelling
foreach *(dimension i of dp)* **do**
estimate $pdf_{kde}(dp)$(probability density function) using Kernel Density Estimation ;
foreach *(knowndistribution)* **do**
estimate $pdf_{mle}(dp)$(probability density function) using Maximum Likelihood Estimation ;
estimate the distribution for which its $pdf_{mle}(dp)$ is closer to $pdf_{kde}(dp)$;
estimate $quantile(0.95)$ an upper-threshold for the estimated distribution ;
estimate $quantile(0.05)$ a lower-threshold for the estimated distribution ;
if *(dp(i) is either greater than the upper-threshold or less than the lower-threshold)* **then**
store this *dp* as a candidate outlier in $CandidateOutlierList$;
end
end
end
Second Part: Nearest Neighbor Scoring Function
if *(dp is an existing point in the current sliding window)* **then**
if *(at least a data point is either inserted into its Neighborhood or deleted from its Neighborhood)* **then**
update $DNO_k(dp)$;
end
end
if *(dp is a new point in the current sliding window)* **then**
compute $DSNO_k(dp)$;
end
if *(dp ∈ CandidateOutlierList) AND* $(DNO_{dp} \geq denthr)$ **then**
store *dp* in the $OutlierList$;
end
end

Chapter 6

Performance Evaluation and Experimental Results

In this section we present and discuss our results from applying our method to both synthetic and real datasets. We also evaluate the performance of our algorithm *SNNOS* and discuss actual performance measurements. The following experiments were conducted on an Pentium Intel Core 2 Duo 2.0 GHz with 2048 MB main memory running Windows Vista. All algorithms were implemented in Java and executed on Eclipse Project , while our experiments were conducted using the MATLAB environment. Moreover, we want to thank Natasa Reljin because she provide us the source code of $ILOF$.

6.1 Description of Datasets

In the following, we describe in detail the datasets which were employed in our experimental evaluation.

- **Meteorological Dataset**

  This is a real dataset which contains 60000 data points with 4 attributes , namely the air temperature, the rainfall, the humidity and the atmospheric pressure.

- **Shuttle Dataset**

  The Shuttle dataset contains 10 attributes all of which are numerical. The last column is the class, which is coded as follows :(1)Rad Flow (2)Fpv Close (3)Fpv

Open (4)High (5)Bypass (6)Bpv Close (7)Bpv Open. Approximately 80% of the data belongs to class 1.

- **Letter Dataset**

  This is a real dataset which contains letter recognition data. The number of instances is 20000, while the number of attributes is 16. The attributes , all of which take integer values, are the following: horizontal position of box, vertical position of box, width of box, height of box, total number of pixels, mean x of on pixels in box, mean y of on pixels in box, mean x variance, mean y variance, mean xy correlation, mean of $x \times x \times y$, mean of $x \times y \times y$, mean edge count left to right, correlation of x-edge with y, mean edge count bottom to top and correlation of y-edge with x.

- **Synthetic Dataset**

  The Synthetic dataset contains 10 attributes all of which are numerical. The number of instances is equal to 1000. We create synthetic data using multivariate normal distributions. The covariance matrices need to be symmetric and positive definite, which is ensured by constructing them to be diagonally dominant, with positive diagonal elements only. The off-diagonal entries of the covariance matrix, generated as a random number in the range $[-1, 1]$ with a distribution following $\pm y, y = x^2$ where x is a uniformly random deviate in $[0, 1]$ and the sign of y is determined randomly. The diagonal entries of the covariance matrix, generated as the sum of all off-diagonal entries plus a random number in the range $0.2\sqrt{D}$ with a distribution following $\pm y, y = x^2$, where x is a uniformly random deviate in $[0, 1]$ he sign of y is determined randomly, and D is the dimensionality of the dataset.

6.2 ROC curves

Outlier detection algorithms are typically evaluated using the true detection rate, the false detection rate, and the *ROC curves*. In order to define these metrics, let us look at a confusion matrix, shown in Table 6.1. In the outlier detection problem, assuming class C as the outlier or the rare class of interest, and NC as a normal (majority) class, there are four possible outcomes when detecting outliers (class C)-namely true positives (TP),

	Predicted Outliers Class C	Predicted Normal Class NC
Actual Outliers - Class C	True Positives (TP)	False Negatives (FN)
Actual Normal - Class NC	False Positives (FP)	True Negatives (TN)

Table 6.1: Confusion matrix defines four possible scenarios when classifying class C

false negatives (FN), false positives (FP) and true negatives (TN). From Table 6.1 , true detection rate and false detection rate may be defined as follows:

$$DetectionRate = \frac{TP}{TP + FN} \tag{6.1}$$

$$FalseAlarmRate = \frac{FP}{FP + TN} \tag{6.2}$$

Detection rate gives information about the relative number of correctly identified outliers, while the false alarm rate reports the number of normal data misclassified as outliers (class NC). The ROC curve represents the trade-off between the detection rate and the false alarm rate and is typically shown on a $two - dimensional(2D)$ (Figure 6.1), where false alarm rate is plotted on x-axis, and detection rate is plotted on y-axis. The ideal ROC curve has 0% false alarm rate, while having 100% detection rate (Figure 6.1). However, the ideal ROC curve is hardly achieved in practice. The ROC curve can be plotted by estimating the detection rate for different false alarm rates (Figure 6.1). The quality of a specific outlier detection algorithm can be measured by computing the area under the curve (AUC) defined as the surface area under its ROC curve. The AUC for the ideal ROC curve is equal to 1, while $AUCs$ of less than perfect outlier detection algorithms are less than 1. In Figure 6.1, the shaded area corresponds to the AUC for the lowest ROC curve.

6.3 Precision and Recall

In a statistical classification task, the *Precision* for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the class) divided by the total number of elements labeled as belonging to the class (i.e. the sum of true positives and

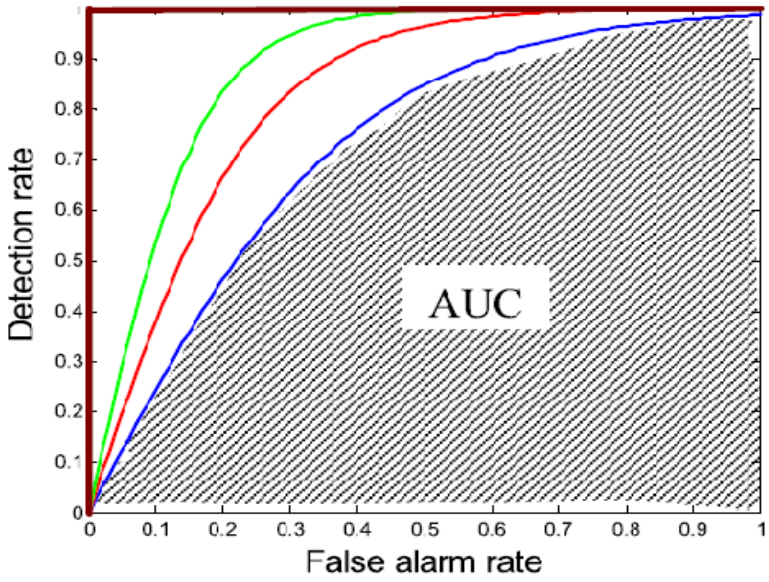

Figure 6.1: The ROC curves for different detection algorithms

false positives, which are items incorrectly labeled as belonging to the class). The *Recall* in this context is defined as the number of true positives divided by the total number of elements that actually belong to the class (i.e. the sum of true positives and false negatives, which are items which were not labeled as belonging to that class but should have been). The formulas for both *precision* and *recall* are as follows.

$$Precision = \frac{TP}{TP + FP} \tag{6.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.4}$$

6.4 Precision, Recall and Execution Time for SNNOS

In this section, we present the total execution time of our algorithm for the four datasets and for a number equal to 1000 data points which is big enough in order to take representable results . In addition, we compare our algorithm in various window sizes for both

	100	200	300	400	500
Meteorological Dataset	2 sec	4 sec	7.1 sec	13.1 sec	19.2 sec
Shuttle Dataset	4 sec	8 sec	15 sec	27.6 sec	39.2 sec
Letter Dataset	6 sec	10 sec	24.1 sec	36 sec	44.8 sec
Synthetic Dataset	3.8 sec	8.1 sec	14.4 sec	27.4 sec	38.9 sec

Table 6.2: Comparison of time execution for the four datasets and for five different sizes of windows, 100, 200, 300, 400 and 500

accuracy and execution time. Moreover, we show how our incremental algorithm outperforms a non-incremental algorithm for different sizes of sliding windows and we show how each part(Statistical and Nearest Neighbor) affects both execution time and accuracy of our algorithm.

6.4.1 Comparison of SNNOS for various window sizes for sliding window model

We compare the performance of our algorithm *SNNOS* for different sizes of sliding window model. We choose to compare five different sizes of sliding window, 100, 200, 300, 400 and 500 with the same incremental step, equal to 50 . The results of our experiments for time execution, precision and recall are presented in the following tables, Table 6.2, Table 6.3 and Table 6.4. From the results, it is obvious that the windows with size equal to 200 and 300 is very good choices for running the algorithm because this size succeeds high accuracy and is efficient in time execution. When the window size is 100 the algorithm is very fast but we the level of accuracy is lower. Moreover, for sizes of windows larger than 200 the accuracy is the same but the overhead in time execution is big enough to prevent us from select these sizes for running our algorithm.

6.4.2 Incremental vs Non-Incremental Implementation

We present the execution time of our algorithm for two different sizes of sliding windows and for each strategy, namely, the non-incremental and the incremental . We expect the

	100	200	300	400	500
Meteorological Dataset	1.0	1.0	1.0	1.0	1.0
Shuttle Dataset	1.0	1.0	1.0	1.0	1.0
Letter Dataset	0.8	0.9	0.9	0.9	1.0
Synthetic Dataset	1.0	1.0	1.0	1.0	1.0

Table 6.3: Comparison of precision for the four datasets and for five different sizes of windows, 100, 200, 300, 400 and 500

	100	200	300	400	500
Meteorological Dataset	1.0	1.0	1.0	1.0	1.0
Shuttle Dataset	1.0	1.0	1.0	1.0	1.0
Letter Dataset	0.8	0.9	0.9	0.9	1.0
Synthetic Dataset	1.0	1.0	1.0	1.0	1.0

Table 6.4: Comparison of recall for the four datasets and for five different sizes of windows, 100, 200, 300, 400 and 500

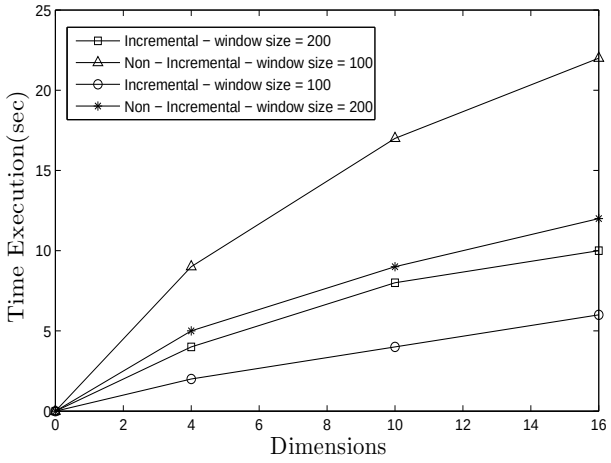

Figure 6.2: Time execution of the incremental and non-incremental implementation, using 1000 data points and windows of sizes 100 and 200.

results shown in Figure 6.2 because with the incremental approach we do not compute the k-nearest neighbor for each data point as we explain in Section 5.3.2, but we update the scores for these data points for which there are changes(insertions or deletions) in their neighborhood. Figure 6.2 shows the differences between the non-incremental and the incremental implementation in time execution.

6.4.3 Statistical vs Nearest Neighbor approach

An interesting experiment is to examine how each part of our algorithm SNNOS(Statistical and Nearest Neighbor) affects both execution time and accuracy of our algorithm. We manage to keep the execution time for nearest neighbor searching in low levels because we compute the k-nearest neighbors for each data point on one scan over the data stream and also we use R*tree indexing for searching which is an efficient way of indexing for nearest neighbor searching. Table 6.5 shows the average time execution for the nearest neighbor part and for window sizes equal to 100 and 200. Table 6.6 shows the average time execution for the statistical part and for window sizes equal to 100 and 200. Moreover, we notice that the total execution time is dominated by the Statistical part. Tables 6.7 and 6.7 show how each part of our algorithm *SNNOS* (Statistical and Nearest Neighbor) affects the accuracy. From the results, it is obvious that a high value of the accuracy of

	Window Size 100	Window Size 200
Meteorological Dataset	0.018 ± 0.002 sec	0.06 ± 0.02 sec
Shuttle Dataset	0.037 ± 0.002 sec	0.14 ± 0.02 sec
Letter Dataset	0.044 ± 0.002 sec	0.18 ± 0.02 sec
Synthetic Dataset	0.035 ± 0.002 sec	0.13 ± 0.02 sec

Table 6.5: Nearest Neighbor part average time execution of $SNNOS$ and for window sizes 100 and 200

	Window Size 100	Window Size 200
Meteorological Dataset	0.1 ± 0.1 sec	0.3 ± 0.1 sec
Shuttle Dataset	0.3 ± 0.1 sec	0.7 ± 0.1 sec
Letter Dataset	0.5 ± 0.1 sec	0.9 ± 0.1 sec
Synthetic Dataset	0.29 ± 0.1 sec	0.68 ± 0.1 sec

Table 6.6: Statistical part time execution of $SNNOS$ and for window sizes 100 and 200

the algorithm is primarily due to the Nearest Neighbor part. In addition, we can see that when the Nearest Neighbor part is not used the decreasing performance is more prominent than when the Statistical part is not employed. For example, if we consider as A_{total} the precision/recall of our algorithm when both parts are used, as A_s the precision/recall when only statistical part is used, and A_{nn} the precision/recall when only nearest neighbor part is used, we can compute the relative decrement of performance as follows.

$$RelativeDecrement = \frac{A_{total} - A_s}{A_{total}} \tag{6.5}$$

$$RelativeDecrement = \frac{A_{total} - A_{nn}}{A_{total}} \tag{6.6}$$

Table 6.9 shows the relative decrement of the performance of our algorithm SNNOS either only the statistical part is used or only the nearest neighbor part is used.

6.5 Real Datasets

For our experiments we use 1000 data points and the window sizes for each time-sliding window is equal to 100 and 200 data points, which are representative numbers for the quality of our experiments. Moreover, for the nearest neighbor part of our algorithm

	Precision	Recall
Meteorological Dataset	0.62	0.62
Shuttle Dataset	0.6	0.6
Letter Dataset	0.6	0.6
Synthetic Dataset	0.63	0.63

Table 6.7: Precision and Recall of $SNNOS$ for window size 200 using only Statistical Part

	Precision	Recall
Meteorological Dataset	0.87	0.87
Shuttle Dataset	0.8	0.8
Letter Dataset	0.8	0.8
Synthetic Dataset	0.81	0.81

Table 6.8: Precision and Recall of $SNNOS$ for window size 200 using only Nearest Neighbor Part

	Using Statistical	Using Nearest Neighbor
Meteorological Dataset	0.38	0.13
Shuttle Dataset	0.4	0.2
Letter Dataset	0.33	0.11
Synthetic Dataset	0.37	0.19

Table 6.9: Relative Decrement in performance of $SNNOS$ for window size 200

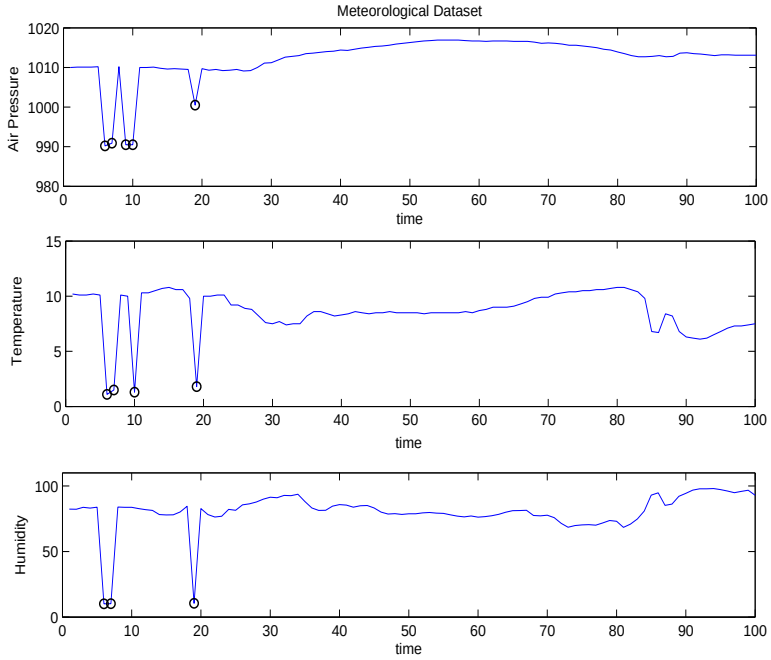

Figure 6.3: Outlier Validation for the Meteorological Dataset for the time period $[1, 100]$

we consider the size of the neighborhood to be equal to $k = 5$. For the validation of our results we combine the plots of data points and the results from the state-of-the-art algorithm LOF. In addition, we plot ROC curves and compute the AUC in order to show how correctly our scoring function DNO detected outliers. In the following sections, we present our results for the datasets.

6.5.1 Meteorological Dataset

The Meteorological Dataset as described above is a real dataset. In the following figure we plot data for one window and the data points which have a circle on them are the outlier data points we take as results from our algorithm. We can see, from Figure 6.3 we are able to to validate our results and consequently to estimate the true detection rate and false alarm rate. The value of AUC in Figure 6.4 is equal to 1 (100%).

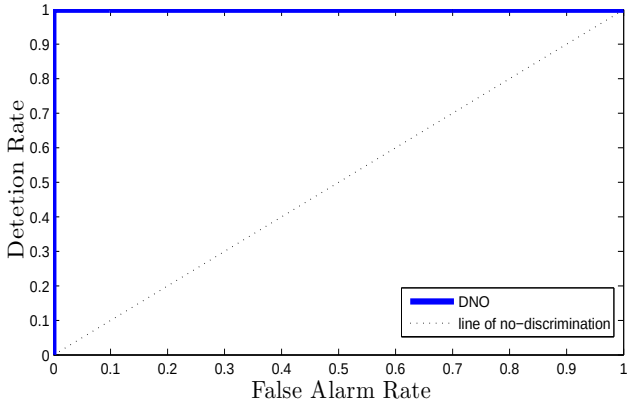

Figure 6.4: ROC Curve for DNO for the Meteorological Dataset

6.5.2 Shuttle Dataset

As we explain, we use the plots of data points to estimate the real outliers. In the Shuttle dataset, we compute the ROC curve for the DNO scoring function and estimate the AUC getting a value greater than 0.9. The value of AUC in the Figure 6.5 is 1 (100%) which means that our scoring function has an excellent percentage of discrimination.

6.5.3 Letter Dataset

The selection of Letter dataset has two goals. Firstly, we want a dataset with many attributes(number of attributes to be greater than ten) and secondly we want to evaluate our algorithm with a dense dataset. The accuracy of many algorithms, especially algorithms which use nearest neighbor searching, decreases if someone evaluates them with dense datasets. This happens because in dense datasets it is very difficult to discriminate the neighborhoods for data points and there are many overlapping areas. Therefore,the accuracy of scoring functions which are based on nearest neighborhoods is decreased. So, the goal for this dataset is to get a value for the AUC greater than 0.9. The value of AUC in Figure 6.6 is 0.987 , so the detection rate for DNO in this dense dataset is 98.7%.

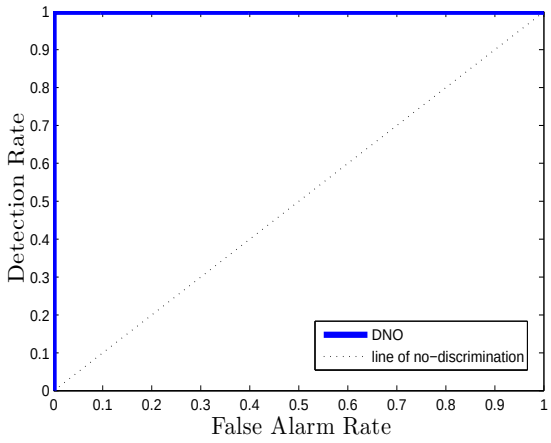

Figure 6.5: ROC Curve for DNO for the Shuttle Dataset

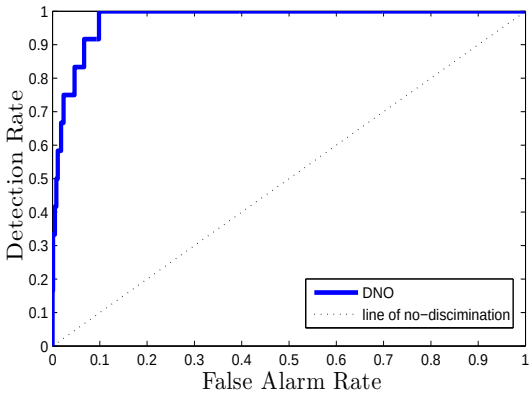

Figure 6.6: ROC Curve for DNO for the Letter Dataset

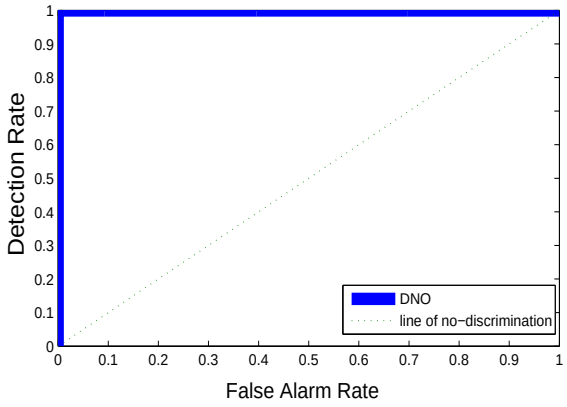

Figure 6.7: ROC Curve for DNO for the Synthetic Dataset

6.6 Synthetic Dataset

This synthetic dataset creates 4 normal clusters and we add outliers in the dataset in specific timestamps. In the Synthetic dataset, we compute the ROC curve for the DNO scoring function and estimate the AUC getting a value greater than 0.9. The value of AUC in the Figure 6.5 is 1 (100%) which means that our scoring function has an excellent percentage of discrimination.

6.7 Comparing SNNOS with ILOF

In this section, we compare the performance of our algorithm $SNNOS$ with the performance of $ILOF$, see Section 4.1. We consider that the $ILOF$ algorithm can find all the outliers in the dataset.

Our algorithm $SNNOS$ outperforms $ILOF$ in time execution and actually we manage to decrease the time execution approximately 62% *when the window size is equal to* 200 *and* 118% *when the window size is equal to* 100.

Table 6.10 shows the execution times of $SNNOS$(window size 100 and 200) and $ILOF$. The fact that our algorithm is faster is something that we expected because we use as the incremental step to our sliding window 50 data points , while the $ILOF$ algorithm updates incrementally only by one point and must compute reachability distances for each

data point. Another reason is the fact that $ILOF$ not only does it compute the k-nearest neighbors but also it computes the k-reverse nearest neighbors.

Also, we modify the $ILOF$ algorithm to use a time-sliding window model in order to make computations over the data. Let denote *Hybrid-ILOF* the modified $ILOF$ with the time-sliding window. Table 6.11 shows the execution times of $SNNOS$(window size 100 and 200), $ILOF$ and $Hybrid - ILOF$. Table 6.15 shows the AUC values for our algorithm $SNNOS$ for window sizes 100 and 200 and $ILOF$ for the three datasets. As we can see in Table 6.15 our algorithm has the same AUC value with $ILOF$ except for the *Letter dataset*. This is due to the fact that *Letter dataset* is very *dense dataset* and there are many overlapping neighborhoods. Moreover, the $SNNOS$ algorithm's AUC value is lower when the sliding window size is 100 because when the window size is small it is more difficult for the scoring function to discriminate the outliers from normal data points. In addition, in Figures 6.8 and 6.9, we can see the comparison between the $SNNOS$ and $ILOF$ with respect to the discrimination level they succeed in the three datasets. Furthermore, we compute the *precision* and *recall* of each algorithm. To compute *precision* and *recall* we have to determine how many outliers identified using $ILOF$ are correctly identified by $SNNOS$. In particular, we were interested in the number of $Top - 10$ data points determined as outliers. Tables 6.13 and 6.14, show the precision and recall of our algorithm for window sizes 100 and 200 and $ILOF$ for the three datasets. A good algorithm must balance both precision and recall, therefore high values for both precision and recall is favored. The results show that our method for the $Meteorological$, $Shuttle$ and $Synthetic$ dataset has the same precision and recall. In the *Letter* dataset the precision and recall of our algorithm is very close to those of $ILOF$. The difference in the *Letter* dataset is due to the fact that the *Letter* dataset is very dense and there are many overlapping neighborhoods for each data point. Moreover, we compare our algorithm $SNNOS$ with $ILOF$ on how the value of AUC, precision and recall is related to the speedup of the execution time. This comparison is illustrated in the Figure 6.10, Figure 6.11 and Figure 6.12. We achieve to speedup the execution time 62% when the size of sliding window is equal to 200 and 118% when the the size of sliding window is equal to 100.

	$SNNOS$(size 100)	$SNNOS$(size 200)	$ILOF$
Meteorological Dataset	2 sec	4 sec	247 sec
Shuttle Dataset	4 sec	8 sec	503 sec
Letter Dataset	6 sec	10 sec	628 sec
Synthetic Dataset	3.8 sec	8.1 sec	510 sec

Table 6.10: Comparison of time execution for $SNNOS$ and $ILOF$

	$SNNOS$(size 100)	$SNNOS$(size 200)	$Hybrid - ILOF$
Meteorological Dataset	2 sec	4 sec	213 sec
Shuttle Dataset	4 sec	8 sec	487 sec
Letter Dataset	6 sec	10 sec	601 sec
Synthetic Dataset	3.8 sec	8.1 sec	495 sec

Table 6.11: Comparison of time execution for $SNNOS$ and $Hybrid - ILOF$ with incremental step for $SNNOS$ equal to 50 and incremental step for $Hybrid - ILOF$ equal to 1

	$SNNOS$(size 100)	$SNNOS$(size 200)	$Hybrid - ILOF$
Meteorological Dataset	5 sec	11 sec	213 sec
Shuttle Dataset	10 sec	18 sec	487 sec
Letter Dataset	13 sec	19 sec	601 sec
Synthetic Dataset	11 sec	17 sec	495 sec

Table 6.12: Comparison of time execution for $SNNOS$ and $ILOF$ with the same incremental step for $SNNOS$, $ILOF$ and $Hybrid - ILOF$ equal to 1

	$SNNOS$(size 100)	$SNNOS$(size 200)	$ILOF$	$Hybrid - ILOF$
Meteorological Dataset	1.0	1.0	1.0	1.0
Shuttle Dataset	1.0	1.0	1.0	1.0
Letter Dataset	0.80	0.90	1.0	1.0
Synthetic Dataset	1.0	1.0	1.0	1.0

Table 6.13: Comparison of *Precision* for $SNNOS$, $ILOF$ and $Hybrid - ILOF$

	$SNNOS$(size 100)	$SNNOS$(size 200)	$ILOF$	$Hybrid - ILOF$
Meteorological Dataset	1.0	1.0	1.0	1.0
Shuttle Dataset	1.0	1.0	1.0	1.0
Letter Dataset	0.8	0.9	1.0	1.0
Synthetic Dataset	1.0	1.0	1.0	1.0

Table 6.14: Comparison of *Recall* for $SNNOS$ and $ILOF$ and $Hybrid - ILOF$

	$SNNOS$(size 100)	$SNNOS$(size 200)	$ILOF$	$Hybrid - ILOF$
Meteorological Dataset	1.0	1.0	1.0	1.0
Shuttle Dataset	1.0	1.0	1.0	1.0
Letter Dataset	0.902	0.987	1.0	1.0
Synthetic Dataset	1.0	1.0	1.0	1.0

Table 6.15: Comparison of AUC values for $SNNOS$ and $ILOF$ and $Hybrid - ILOF$

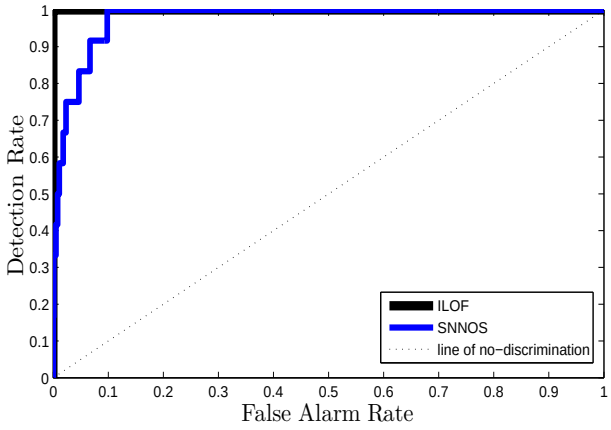

Figure 6.8: Comparison of ROC Curves for ILOF and DNO for Letter Dataset

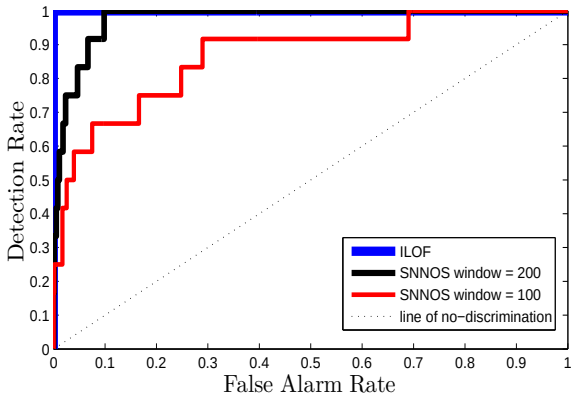

Figure 6.9: Comparison of ROC Curves for ILOF and DNO for Letter dataset with different sizes of windows for DNO

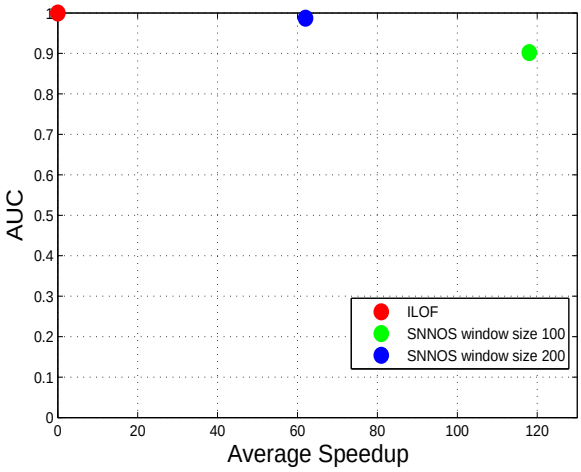

Figure 6.10: Comparison of SNNOS and ILOF on how the value of AUC is related to the speedup of execution time

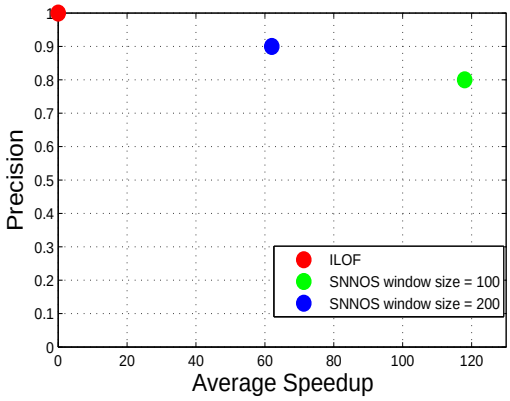

Figure 6.11: Comparison of SNNOS and ILOF on how the value of precision is related to the speedup of execution time

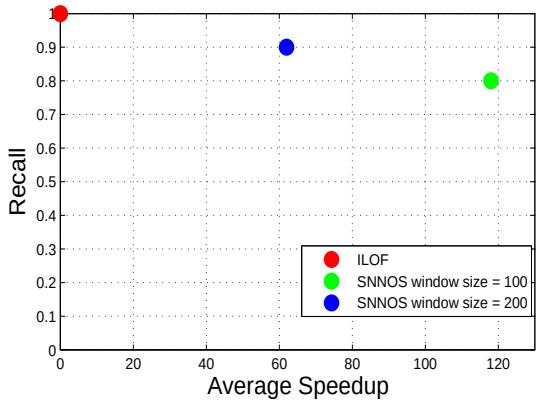

Figure 6.12: Comparison of SNNOS and ILOF on how the value of recall is related to the speedup of execution time

6.8 Memory Consumption

In order to verify the efficiency of our algorithm, we measure the amount of memory required by the algorithm. There are two groups of components of our algorithm that affect the memory consumption: The first group is the sample maintenance in a sliding window and the distribution estimation. For the sample maintenance, we use a treemap while for the distribution estimation we keep additionally in memory a hasmap with the density estimations using the kernel density estimation method and maximum likelihood estimation. For the second group, similarly as for the first group, we use the treemap for sample maintenance in a window, constructing the R*tree index for efficient nearest neighbor searching and we employ a list in which we store the *outlier data points*. Consequently, the memory consumption of our algorithm is low. This is due to our strategy to use a cache manager where we keep the incoming data streams. Then we retrieve the data which belong to a specific sliding window. The $ILOF$ algorithm keeps all the data points in main memory and when a new data point inserted into the database incrementally updates the LOF values for all data points. This strategy has high memory consumption and when the dataset is very large could be inefficient. Our approach to keep only the data points in the current sliding window in main memory and the other in a cache manager is efficient. Therefore, *$SNNOS$ outperforms $ILOF$ in memory consumption.*

Chapter 7

Conclusion and Future work

In the present work, we described an algorithm for detecting outliers in data streams. The proposed algorithm is non-parametric, unsupervised and requires no prior knowledge of data.

The proposed algorithm($SNNOS$) is divided into the statistical and the nearest neighbor stage. The statistical part consists of two methods for statistical learning, namely, the *Kernel Density Estimation* (KDE) method and the *Maximum Likelihood Estimation*(MLE) method. The KDE method uses kernel density functions to estimate probability densities for each data point in the window. The MLE method, is carried out by employing ten continuous distributions for each one of them we estimate the probability density for each data point in the window using *Maximum Likelihood Estimation*. The nearest neighbor part consists of a density scoring function DNO(density-based). In Chapter 6, we presented the experimental results using the proposed algorithm, and we compared its performance with the performance of a state-of-the-art algorithms, namely, the $ILOF$. The *ROC* curves for the three different datasets along with the computation of *precision* and *recall* shows that the proposed $SNNOS$ algorithm is very efficient. Moreover, the execution time with the proposed incremental strategy decreases and outperforms the time of $ILOF$.

As a future research direction, one could improve the system in order to process high dimensional datasets, where new methods for estimating data densities are worth considering. Such methods could be based on algorithms which perform dimensionality reduction

by selecting similar attributes and group the several attributes. Moreover, a modification of the DNO scoring function is necessary to succeed better accuracy percentage in very dense datasets. Finally, it would be very interesting to use our algorithm and examine its performance in a real-time data stream mining system.

Bibliography

[1] Elke Achtert, Hans-Peter Kriegel, and Arthur Zimek. Elki: A software system for evaluation of subspace clustering algorithms. In Bertram Ludscher and Nikos Mamoulis, editors, *SSDBM*, volume 5069 of *Lecture Notes in Computer Science*, pages 580–585. Springer, 2008.

[2] Deepak Agarwal. Detecting anomalies in cross-classified streams: a bayesian approach. *Knowl. Inf. Syst.*, 11(1):29–44, 2006.

[3] C. Aggarwal and S. Yu. An effective and efficient algorithm for high-dimensional outlier detection. *The VLDB Journal*, 14(2):211–221, 2005.

[4] Charu C. Aggarwal and Philip S. Yu. Outlier detection with uncertain data. In *SDM*, pages 483–493. SIAM, 2008.

[5] Amrudin Agovic and Arindam Banerjee. Anomaly detection in transportation corridors using manifold embedding abstract, 2007.

[6] L. Douglas Baker, Thomas Hofmann, Andrew K. Mccallum, and Yiming Yang. A hierarchical probabilistic model for novelty detection in text, 1999.

[7] Daniel Barbar, Yi Li, Jia-Ling Lin, Sushil Jajodia, and Julia Couto. Bootstrapping a data mining intrusion detection system. In *SAC*, pages 421–425. ACM, 2003.

[8] Daniel Barbará, Ningning Wu, and Sushil Jajodia. Detecting novel network intrusions using bayes estimators. In *Proceedings of the First SIAM Conference on Data Mining*, April 2001.

[9] V. Barnett and T. Lewis. Outliers in statistical data. *International Journal of Forecasting*, 12(1):175–176, March 1996.

[10] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. pages 29–38, New York, NY, USA, 2003. ACM.

[11] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.

[12] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. pages 93–104, 2000.

[13] Re Bronstein, Joydip Das, Marsha Duro, Rich Friedrich, Gary Kleyner, Martin Mueller, Sharad Singhal, Ira Cohen, G. Kleyner, M. Mueller, S. Singhal, and I. Cohen. Self-aware services: Using bayesian networks for detecting anomalies in internet-based services. In *Northwestern University and Stanford University. Gary (Igor*, pages 623–638. Publishing, 2001.

[14] Varun Ch, Arindam Banerjee, Vipin Kumar, and Varun Chandola. Outlier detection: A survey, 2007.

[15] Amitabh Chaudhary, Alexander S. Szalay, and Andrew W. Moore. Very fast outlier detection in large multidimensional data sets. In *DMKD*, 2002.

[16] Hongyin Cui. Online outlier detection over data streams. Master's thesis, 2005.

[17] Kaustav Das and Jeff Schneider. Detecting anomalous records in categorical datasets. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 220–229, New York, NY, USA, 2007. ACM.

[18] Christopher P. Diehl and John B. Hampshire Ii. Real-time object classification and novelty detection for collaborative video surveillance. In *In Proceedings of the International Joint Conference on Neural Networks*, pages 2620–2625, 2002.

[19] Levent Ertoz, Michael Steinbach, and Vipin Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. In *Workshop on Text Mining, held in conjunction with the First SIAM International Conference on Data Mining (SDM 2001)*. Society for Industrial and Applied Mathematics, 2003.

[20] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. *A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data*. Kluwer, 2002.

[21] Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. pages 255–262. Morgan Kaufmann, 2000.

[22] Eleazar Eskin, Wenke Lee, and Salvatore J. Stolfo. Modeling system calls for intrusion detection with dynamic window sizes. In *In Proceedings of DARPA Information Survivabilty Conference and Exposition II (DISCEX*, 2001.

[23] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231, 1996.

[24] Pedro Galeano, Daniel Pea, and Ruey S. Tsay. Outlier detection in multivariate time series via projection pursuit. Statistics and Econometrics Working Papers ws044211, September 2004.

[25] J. Gama and M. Gaber (Eds). *Learning from Data Streams*. Springer, 2007.

[26] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 13–24, New York, NY, USA, 2001. ACM.

[27] Amol Ghoting, Srinivasan Parthasarathy, and Matthew Eric Otey. Fast mining of distance-based outliers in high-dimensional datasets. *Data Min. Knowl. Discov.*, 16(3):349–364, 2008.

[28] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Inf. Syst.*, 25(5):345–366, 2000.

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

[30] Zengyou He, Shengchun Deng, and Xiaofei Xu. Outlier detection integrating semantic knowledge. In Xiaofeng Meng, Jianwen Su, and Yujun Wang, editors, *WAIM*, volume 2419 of *Lecture Notes in Computer Science*, pages 126–131. Springer, 2002.

[31] Zengyou He, Shengchun Deng, Xiaofei Xu, and Joshua Zhexue Huang. A fast greedy algorithm for outlier mining. pages 567–576, 2006.

[32] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[33] Zengyou He, Xiaofei Xu, and Shengchun Deng. An optimization model for outlier detection in categorical data, 2005.

[34] Wenjie Hu, Yihua Liao, and V. Rao Vemuri. Robust anomaly detection using support vector machines. In *In Proceedings of the International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc, 2002.

[35] Tsuyoshi IDÉ and Hisashi KASHIMA. Eigenspace-based anomaly detection in computer systems. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 440–449, New York, NY, USA, 2004. ACM.

[36] M. Kamber J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.

[37] M. Juhola J. Laurikkala and E. Kentala. Informal identification of outliers in medical data. In *In Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology.*, pages 20–24, 2000.

[38] Wen Jin, Anthony K. H. Tung, and Jiawei Han. Mining top-n local outliers in large databases. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 293–298, New York, NY, USA, 2001. ACM.

[39] Mahesh V. Joshi, Ramesh C. Agarwal, and Vipin Kumar. Predicting rare classes: can boosting make any weak learner strong? In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 297–306, New York, NY, USA, 2002. ACM.

[40] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215, New York, NY, USA, 2004. ACM.

[41] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3-4):237–253, 2000.

[42] Teuvo Kohonen. *Self-organizing maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[43] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228, 2005.

[44] Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[45] Pierre L'Ecuyer and Eric Buist. Simulation in java with ssj. In *WSC '05: Proceedings of the 37th conference on Winter simulation*, pages 611–620. Winter Simulation Conference, 2005.

[46] Junshui Ma and Simon Perkins. Online novelty detection on temporal sequences. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 613–618, New York, NY, USA, 2003. ACM.

[47] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.

[48] Markos Markou and Sameer Singh. Novelty detection: A review - part 1: Statistical approaches. *Signal Processing*, 83:2003, 2003.

[49] Matthew Eric Otey, Amol Ghoting, and Srinivasan Parthasarathy. Fast distributed outlier detection in mixed-attribute data sets. *Data Min. Knowl. Discov.*, 12(2-3):203–228, 2006.

[50] Matthew Eric Otey, Srinivasan Parthasarathy, Amol Ghoting, G. Li, Sundeep Narravula, and Dhabaleswar K. Panda. Towards nic-based intrusion detection. In Lise Getoor, Ted E. Senator, Pedro Domingos, and Christos Faloutsos, editors, *KDD*, pages 723–728. ACM, 2003.

[51] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. pages 315–, 2003.

[52] Lucas Parra, Gustavo Deco, and Stefan Miesbach. Statistical independence and novelty detection with information preserving nonlinear maps. *Neural Comput.*, 8(2):260–269, 1996.

[53] Peter G. Neumann Phillip A. Porras. Emerald: Event monitoring enabling responses to anomalous live disturbances. Technical report, SRI International, Menlo Park, CA 94025.

[54] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. Incremental local outlier detection for data streams. pages 504–515, 2007.

[55] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. pages 427–438, 2000.

[56] Gunnar Rätsch, Sebastian Mika, Bernhard Schökopf, and Klaus-Robert Müller. Constructing boosting algorithms from svms: an application to one-class classification, 2002.

[57] David G. Stork Richard O. Duda, Peter E. Hart. *Pattern Classification*. Wiley-Interscience, 2nd Edition, 2000.

[58] Volker Roth. Kernel fisher discriminants for outlier detection. *Neural Computing*, 18(4):942–960, 2006.

[59] Stan Salvador and Philip Chan. Learning states and rules for detecting anomalies in time series. *Applied Intelligence*, 23(3):241–255, 2005.

[60] Bernhard Schlkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[61] David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley-Interscience, September 1992.

[62] Abdallah Abbey Sebyala, Temitope Olukemi, and Dr. Lionel Sacks. Active platform security through intrusion detection using naive bayesian network for anomaly detection. In *In London Communications Symposium*, 2002.

[63] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB*, pages 428–439. Morgan Kaufmann, 1998.

[64] Meiling Shyu, Shuching Chen, Kanoksri Sarinnapakorn, and Liwu Chang. A novel anomaly detection scheme based on principal component classifier. In *In IEEE Foundations and New Directions of Data Mining Workshop, in conjunction with ICDM03*, pages 172–179, 2003.

[65] B. W. Silverman. *Density estimation: for statistics and data analysis*. London, 1986.

[66] Xiuyao Song, Mingxi Wu, and Christopher Jermaine. Conditional anomaly detection. *IEEE Trans. on Knowl. and Data Eng.*, 19(5):631–645, 2007. Fellow-Sanjay Ranka.

[67] Claudio De Stefano, Carlo Sansone, and Mario Vento. To reject or not to reject: that is the question-an answer in case of neural classifiers. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 30(1):84–94, 2000.

[68] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. Online outlier detection in sensor data using non-parametric models. pages 187–198, 2006.

[69] Huanliang Sun, Yubin Bao, Faxin Zhao, Ge Yu, and Daling Wang. Cd-trees: An efficient index structure for outlier detection. In *WAIM*, pages 600–609, 2004.

[70] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Stat. Anal. Data Min.*, 1(1):6–22, 2008.

[71] Pang-Ning Tan. *Introduction to Data Mining*. ADDISON WESLEY PUBLICATIONS, June 2006.

[72] Gaurav Tandon and Philip K. Chan. Weighting versus pruning in rule validation for detecting network and host anomalies. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 697–706, New York, NY, USA, 2007. ACM.

[73] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David Wai-Lok Cheung. Enhancing effectiveness of outlier detections for low density patterns. In *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 535–548, London, UK, 2002. Springer-Verlag.

[74] Yufei Tao, Xiaokui Xiao, and Shuigeng Zhou. Mining distance-based outliers from large databases in any metric space. pages 394–403, New York, NY, USA, 2006. ACM.

[75] P H S Torr and D W Murray. Outlier detection and motion segmentation. pages 432–443, 1993.

[76] Ricardo Vilalta and Sheng Ma. Predicting rare events in temporal domains. In *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, page 474, Washington, DC, USA, 2002. IEEE Computer Society.

[77] Li Wei, Weining Qian, Aoying Zhou, Wen Jin, and Jeffrey Xu Yu. Hot: Hypergraph-based outlier test for categorical data. In *PAKDD*, pages 399–410, 2003.

[78] Graham Williams, Rohan Baxter, Hongxing He, Simon Hawkins, and Lifang Gu. comparative study of rnn for outlier detection in data mining. In *in ICDM*, page 709, 2002.

[79] Weng Keen Wong, Andrew Moore, Gregory Cooper, and Michael Wagner. Bayesian network anomaly pattern detection for disease outbreaks. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*, 2003.

[80] Mingxi Wu and Christopher Jermaine. Outlier detection by sampling with accuracy guarantees. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 767–772, New York, NY, USA, 2006. ACM.

[81] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. quality and reliability engineering. *International*, 17:105–112, 2001.

[82] Dantong Yu, Gholamhosein Sheikholeslami, and Aidong Zhang. Findout: Finding outliers in very large datasets. *Knowl. Inf. Syst.*, 4(4):387–412, 2002.

[83] Jeffrey Xu Yu, Weining Qian, Hongjun Lu, and Aoying Zhou. Finding centric local outliers in categorical/numerical spaces. *Knowl. Inf. Syst.*, 9(3):309–338, 2006.

[84] Xiaofei Xu Zengyou HE and Shengchun Deng. Outlier detection over data streams. In *International Conference for Young Computer Scientists (ICYCS03)*, 2003.

[85] Zhexue Huang Joshua Deng Shengchun Zengyou he, Xiaofei xu. A frequent pattern discovery method for outlier detection, 2004.

[86] Ji Zhang and Hai Wang. Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. *Knowl. Inf. Syst.*, 10(3):333–355, 2006.

[87] Y. Zhang, N. Meratnia, and P. J. M. Havinga. A taxonomy framework for unsupervised outlier detection techniques for multi-type data sets. Technical Report TR-CTIT-07-79, Enschede, November 2007.