TOWARD ACCURATE AND EFFICIENT OUTLIER DETECTION IN HIGH DIMENSIONAL AND LARGE DATA SETS

A Thesis
Presented to
The Academic Faculty

by

Minh Quoc Nguyen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science

Georgia Institute of Technology
August 2010

TOWARD ACCURATE AND EFFICIENT OUTLIER DETECTION IN HIGH DIMENSIONAL AND LARGE DATA SETS

Approved by:

Professor Leo Mark, Advisor
College of Computing
Georgia Institute of Technology

Professor Edward Omiecinski, Advisor
College of Computing
Georgia Institute of Technology

Professor Sham Navathe
College of Computing
Georgia Institute of Technology

Professor Ling Liu
College of Computing
Georgia Institute of Technology

Professor Jinho Kim
Department of Computer Science
Kangwon National University

Date Approved: 5 April 2010

To my parents,

my wife,

and my daughter.

ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge my advisors, Professors Leo Mark and Edward Omiecinski for their priceless effort in guiding me throughout my thesis. I would like to attribute my thesis to my advisors for their support and encouragement. I am also thankful for their patience in helping me prepare my resume for my job search.

I would like to thank my committee members: Professors Leo Mark, Edward Omiecinski, Sham Navathe, Ling Liu, and Jinho Kim. I am thankful for Navathe's advice on writing my thesis proposal. Mark, Omiecinski, and Navathe were helpful with my teaching assistantship. In my first year, Liu and Navathe made me feel comfortable to be at Georgia Tech with their warm welcome.

I also acknowledge the Vietnam Education Foundation (VEF) for providing funding during my first two years at Georgia Tech and for providing invaluable networking to support my career.

I will never forget my labmates: Rocky, Saurav, Weiyun, Sangeetha, and Wei. Your friendships made the lab my home. You have been very supportive.

I am thankful for my parents who always believe in me and encourage me to pursue the path that I have chosen. And lastly, I owe my wife a special thank you for her encouragement and endless support during my Ph.D. study. Her patience, endurance, and love give me courage and concentration to reach my goal.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF TABLES ix

LIST OF FIGURES x

SUMMARY xii

I INTRODUCTION 1

1.1 Outlier Detection in Large and High Dimensional Data 3

1.2 Detecting Patterns of Anomalous Activities 4

1.3 Organization of the Thesis 5

II RELATED WORK 6

2.1 Tabular data 8

2.1.1 Statistical Methods 8

2.1.2 Supervised Methods 9

2.1.3 Semi-supervised Methods 11

2.1.4 Unsupervised Methods 12

2.1.5 Distance-based and Density-based 14

2.2 Anomaly in Sequential Data 14

2.2.1 Detecting Unusual Sequences 15

2.2.2 Detecting Unusual Subsequences in a Sequence 17

2.2.3 Detecting Unusual Values in a Sequence 18

2.3 Graph-based Anomaly 19

2.4 Robust Outlier Methods 20

2.5 Summary 21

III EFFICIENT LOCAL DENSITY-BASED OUTLIER DETECTION USING RANDOMIZATION 22

3.1 Motivation 22

3.2 Related Work 24
3.3 Local Outlier Factor 25
3.4 Generalized Local Density-based Outlier 26
3.5 Algorithm 29
3.5.1 Query Time of New Point 31
3.5.2 Time Complexity Analysis 33
3.6 Experiments 35
3.6.1 2D Example 35
3.6.2 Real Datasets 36
3.6.3 Dimensionality 39
3.6.4 Speed Comparison 40
3.6.5 Performance 41
3.6.6 Convergence Rate 41
3.6.7 Other Parameters 42
3.7 Conclusion 43
IV ANOMALOUS PATTERN DETECTION USING ADAPTIVE NEAREST NEIGHBOR APPROACH 44
4.1 Motivation 44
4.2 Adaptive Nearest Neighbors 44
4.3 Experiments 52
4.3.1 LOF, KMEAN, SNN 52
4.3.2 Outcast 55
4.4 Conclusion 57
V ACCURATELY RANKING OUTLIERS WITH DELTA DEVIATION ACCUMULATION AND MIXTURE OF VARIANCES 59
5.1 Motivation 59
5.1.1 Mixture of Variances 60
5.1.2 Accumulated Sub-dimensional Variations 62
5.2 Related Work 64

5.3 Concept of Outlier in High Dimensions 66
5.3.1 Filtering and Normalization 68
5.4 Sub-dimensional Outlier 69
5.5 Clustering 77
5.6 Experiments 79
5.6.1 Synthetic Dataset 79
5.6.2 KDD CUP '99 Dataset 81
5.6.3 The Effect of the Filter Parameter 84
5.6.4 Selecting ρ_0 85
5.6.5 Visualization 86
5.7 Conclusion 88
VI EFFICIENT ANOMALOUS CORRELATED ATTRIBUTE PATTERN DETECTION IN MULTI-DIMENSIONAL DATA 90
6.1 Motivation 90
6.2 Comparison with outlier detection and clustering 91
6.3 Related work 94
6.4 Relational Algebra 95
6.5 Core Rules and Patterns 96
6.6 Pattern Construction 97
6.7 Core Rule Construction 99
6.8 Attribute Selection for Core Rule 103
6.9 Framework 105
6.9.1 Building Core Rules 105
6.9.2 Pruning Candidates 105
6.9.3 Interval-Tuple Construction 106
6.10 Parameter Setting 107
6.11 Running-time Complexity 108
6.12 Experiments 109
6.12.1 Our Method 113

6.13 Performance 117
6.14 Conclusion 118
VII CONCLUSION 119
VIII FUTURE WORK 122
8.1 Randomized Algorithm 122
8.2 Anomalous Pattern Detection Using Correlated Attributes 123
8.3 Complexity in Data Model 124
REFERENCES 125
VITA 141

LIST OF TABLES

1 Outlier detection methods categorized based on data type and the availability of training data 21

2 Outlier detection methods categorized based on distribution assumption and the availability of training data 21

3 Magic, Physics, and KDD Cup '99 dataset description 37

4 Running time: Randomization versus LOF 40

5 Top 10 LOF outliers in Sam's club data 53

6 Clusters in Sam's club data generated by SNN 53

7 Interesting patterns clustered by Outcast 55

8 Notations and basic definitions 64

9 Seven outliers are detected 80

10 Nine outlier clusters are detected in 2D dataset 83

11 Detected attack connections in KDD CUP dataset 84

12 Dimensional and aggregated outlier scores for p_7 and p_{36} 88

13 An example dataset contains three groups. 92

14 A subset of correlated observations 93

15 Statistics of X_1, X_2, X_3 and X_4 93

16 List of attacks 110

17 Top outliers (10%) clustered by kmean 111

18 Top outliers (10%) clustered by SNN 112

19 The results of clustering using SNN on the entire dataset 112

20 Top 10 unusual patterns ordered by size 114

21 Other unusual patterns ordered by size 114

22 Unusual patterns ordered by score 115

23 The recall rate of the attack types in the top 10 patterns 115

LIST OF FIGURES

1 An example of a PST tree 17
2 Outliers with respect to their local subspaces. 27
3 Randomized local density-based outlier method 30
4 Compute candidate outliers 31
5 Partition a set using random split points 31
6 Tree structure of randomized LOF 32
7 2D test data for local outlier 35
8 Similarity in ranking for randomized outlier detection 36
9 Effect of dimensionality on the detection rate. 39
10 Running time: Randomization versus LOF 40
11 Running time of the randomized method 41
12 Convergence rate 42
13 k^{th} nearest neighbor of p and q 45
14 Adaptive nearest neighbors of p and q 45
15 Constructing adaptive nearest neighbors 50
16 Outcast pseudocode 51
17 The average daily sales in January for all items 52
18 The average daily sale for all items and top outliers 54
19 The average sale volumne for each day in January 56
20 The 2D outlier p 61
21 The outlier p is suppressed in the 3D space 62
22 Clustering pseudocode 78
23 Points p_1, p_2, p_3 and cluster C_1 are generated outliers 80
24 Detection curve of the filtering method on KDD Cup '99 82
25 Detection rate for the algorithm with/without using the filter 85
26 Point p_{36} is not an outlier in this 2d-subspace 86
27 Point p_{36} is an outlier in the 1^{st} and 26^{th} dimensions 87

28 Local region with p_{36} removed in the 1^{st} and 26^{th} dimensions 87
29 An example of core rule construction 102
30 Feature-based anomalous pattern detection 104
31 Constructing interval-tuples 108
32 LOF with varied Min_pts 111
33 Feature-based method versus LOF 111
34 Running time with different orders of attributes 116
35 Running time of the feature-based method 116

SUMMARY

Advances in computing have led to the generation and storage of extremely large amounts of data every day. Data mining is the process of discovering relationships within data. The identified relationships can be used for scientific discovery, business decision making, or data profiling. Among data mining techniques, outlier detection plays an important role. Outlier detection is the process of identifying events that deviate greatly from the masses. The detected outliers may signal a new trend in the process that produces the data or signal fraudulent activities in the dataset. This thesis shows that the efficiency and accuracy of unsupervised outlier detection methods for high dimensional tabular data can be greatly improved.

Local outlier factor (LOF) is an unsupervised method to detect local density-based outliers. The advantage of the method is that the outliers can be detected without training datasets or prior knowledge about the underlying process that produces the dataset. The method requires the computation of the k-nearest neighbors for the dataset. The main problem is that the efficiency and accuracy of the indexing method for computing k-nearest neighbors deteriorates in high dimensional data. The first contribution of this work is to develop a method that can compute the local density-based outliers very efficiently in high dimensional data. In our work, we have shown that this type of outlier is present even in any subset of the dataset. This property is used to partition the data set into random subsets to compute the outliers locally. The outliers are then combined from different subsets. Therefore, the local density-based outliers can be computed very efficiently. Another challenge in outlier detection in high dimensional data is that the outliers are often suppressed when the majority of dimensions do not exhibit outliers. The contribution of this work is to introduce

a filtering method where outlier scores are computed in sub-dimensions. The low sub-dimensional scores are filtered out and the high scores are aggregated into the final score. This aggregation with filtering eliminates the effect of accumulating delta deviations in multiple dimensions. Therefore, the outliers are identified correctly.

In some cases, the set of outliers that form micro patterns are more interesting than individual outliers. These micro patterns are considered anomalous with respect to the dominant patterns in the dataset. In the area of anomalous pattern detection, there are two challenges. The first challenge is that the anomalous patterns are often overlooked by the dominant patterns using the existing clustering techniques. A common approach is to cluster the dataset using the k-nearest neighbor algorithm. The contribution of this work is to introduce the adaptive nearest neighbor and the concept of dual-neighbor to detect micro patterns more accurately. The next challenge is to compute the anomalous patterns very fast. Our contribution is to compute the patterns based on the correlation between the attributes. The correlation implies that the data can be partitioned into groups based on each attribute to learn the candidate patterns within the groups. Thus, a feature-based method is developed that can compute these patterns very efficiently.

CHAPTER I

INTRODUCTION

In this decade, we have seen a widespread use of data mining applications by universities, government agencies, and business organizations. The data mining applications are used to discover the behaviors of the collected observations, which could have not been found manually. The corporations utilize this knowledge to gain a competitive advantage by being able to predict the market and user behaviors more accurately [62]. The government uses data mining methods to detect fraudulent activities [111]. Research institutions apply data mining methods to better understand the relationships in the dataset that may lead to scientific discovery [79]. Generally, data mining is used to infer the common patterns in a data set. The common techniques are association rule mining, clustering and classification. However, recently, along with pattern detection, the data mining community is showing substantial interest in detecting outliers in datasets. According to Hawkin [65], an outlier, or anomaly, is an observation that deviates from others so greatly that it raises suspicion in the dataset. Because of that, outlier detection has several important applications. Outliers can provide interesting insight about the dataset. For example, the network activities that is surprisingly high with respect to its network may indicate an error or a network attack in the system. The appearance of an outlier may indicate a new trend in the process that produces the data or an undiscovered error in the system. In some cases, outlier detection is the only technique that can be used to discover malicious activities that might have not been discovered by any other means.

Outlier or anomaly detection can be divided into different categories depending on the nature of the dataset. The first category is anomaly detection in sequential data

[122]. Sequential data is simply an ordered sequence of data. In database auditing, an anomalous sequence of data access may indicate a malicious unauthorized access in the database. Another category is anomaly detection in graph data. An example is a sudden bursts in network connections, which is an important indicator of a network attack. Another category is outlier detection in tabular data. In this category, a feature selection technique is used to select features representing the characteristics of the dataset. Then, the data is transformed into a set of records with respect to the feature set. An outlier in this category is a record that deviates from the rest in the transformed dataset. Outlier detection in tabular data is the primary focus of this thesis.

Outlier detection in tabular data further falls into three subcategories: supervised, semi-supervised, or unsupervised. Supervised outlier detection determines the class of an observation from the classifiers. The classifiers are the machine learning models whose parameters are learned from the provided training dataset. Neural network and decision trees are popular methods in this subcategory. The main challenge in constructing the classifiers is the skewness in the dataset between the normal class and anomaly. Because the training data for anomalies is too small relatively with respect to the data for the normal class, the penalty functions tend to favor the normal class. Even if the classifier misses the anomalies completely, it still achieves the correction rate of 99%. In the absence of training data for the outliers, the semi-supervised methods can be used. In this case, a machine learning model is selected to capture the boundaries of the normal class. A new observation that falls outside the boundaries is classified as an outlier. A one-class support vector machine is often used for semi-supervised outlier detection. Generally, the supervised method has a high precision rate; however, it depends on the availability of training datasets for both normal and outlying observations. A new type of outlier will not be detected. The semi-supervised method depends on the availability of the training dataset for normal

observations. The semi-supervised learning method can incorrectly identify a new normal observation that falls outside the trained boundary as an outlier. In practice, we usually do not have sample data for outliers or for all the normal observations. The unsupervised methods are used to detect outliers in such cases. This thesis addresses the problems in unsupervised outlier detection.

1.1 Outlier Detection in Large and High Dimensional Data

Historically, outliers have been studied extensively in statistics. The outliers can be detected by conducting hypothesis tests, e.g. Grubb's test [56], Dixon test [123]. Regardless of the type of statistical test, the statistical methods perform the hypothesis test against an assumed distribution of the underlying process that produces the dataset. In many cases, the distributions are unknown. Moreover, the statistical tests are generalized to multivariate tests for datasets with more than one attribute, The multivariate statistical methods, however, are only effective for a very small number of attributes. Therefore, distance-based outlier detection is practically preferred for multi-dimensional data by comparing the distance between the points in a dataset.

In Knorr's definition [51], the observations with the largest distances are outliers. These outliers are considered to be global outliers. Breunig [24] introduces the concept of local density-based outliers. An observation that deviates greatly from its neighbors with respect to its local density is considered to be an outlier. The density is measured by the length of the k-nearest neighbor distances of its neighbors. Even though a local outlier may not deviate from all the other observations, it, indeed, may signal interesting information. For example, a network activity may be considered exceptionally high depending on the nature of the network. Specifically, a server dedicated for content streaming likely requires more bandwidth than email servers.

However, in the calculation of local density-based outliers, the computation of

the k-nearest neighbor is costly. The local density-based method utilizes multi-dimensional index trees to speed up the k-nearest neighbor computation. A multi-dimensional index tree is a hierarchy of the boundaries of the set of points in the subtrees. When the number of dimensions increases, the number of possible boundaries to contain the subsets explodes. Therefore, the performance of the index trees deteriorates in high dimensions. One of our contributions in this thesis is to introduce a method that can compute the local density-based outliers efficiently. We observe that when the dataset is partitioned into multiple subspaces, the local outliers are still prominent in any of these subspaces. This leads to the randomized method for local density-based outlier detection in high dimensional data.

In many cases, an outlier deviates greatly from only some subsets of the attributes. In multi-dimensional data, the delta deviations can be accumulated. When the number of attributes is large, the accumulation can be significant. This accumulation can even be greater than the deviation of an outlier in the set of attributes when it shows up as a strong outlier. Therefore, such outliers will be overlooked in a high dimensional dataset. Another contribution of our work in local outlier detection is that we introduce a method to compute the deviations for each attribute. We filter the attributes with delta deviation. Then, we aggregate the deviations in the remaining attributes. Therefore, the outliers can be detected correctly.

1.2 Detecting Patterns of Anomalous Activities

The previous section described outlier detection in tabular data. The outliers in discussion are mainly individual outliers. The false alarm rate is usually high for outlier detection methods. A detected outlier may only have a random deviation from the dataset. When the outliers appear in groups, it may indicate that there is a pattern in the deviation. Therefore, in some cases, the groups of outliers are more interesting than individual outliers. A clustering algorithm, like the kernel method

[55], can be used to cluster the datasets. However, the clustering methods are not designed to discover groups of outliers. They either tend to cluster the groups of outliers with the dominant clusters or split a cluster into small clusters. Therefore, the groups of outliers are not detected correctly. Our contribution is to detect the groups of outliers. We introduce an adaptive dual-neighbor method that can detect anomalous patterns with different levels of granularity. The method is, however, inefficient for large datasets. To correct this problem, we introduce another type of anomalous pattern. We assume that the patterns are defined by the correlations between the attributes. This allows the pattern to be learned by partitioning the datasets based on attributes. During the learning process, a pattern will be pruned out as an anomalous candidate if its size is large. Therefore, the algorithm can run very efficiently for large datasets.

1.3 Organization of the Thesis

The thesis is organized as follows. In the following chapter, we will present important research in outlier detection. In the third chapter, we will discuss the randomization method to detect local density-based outliers efficiently. Chapter 4 presents the adaptive method to identify the anomalous patterns more accurately. Chapter 5 describes the problem of delta deviation accumulation in detecting individual outliers. We then describe the filtering method to improve the quality of outlier detection in such cases. The adaptive neighbor method introduced in Chapter 4 is applied to cluster datasets with delta deviation accumulation. Chapter 6 introduces an efficient method to detect attribute-based correlated anomalous patterns. Chapter 7 will give a summary of our work. The future work will be discussed in the last chapter.

CHAPTER II

RELATED WORK

Outlier detection has been used extensively in many applications. Banking institutions use neural network methods in order to detect credit card fraud. The neural network methods learn profiles of customers using their transaction histories. If a new transaction deviates from its profile, it will signal an alarm to the system. Outlier detection is also used in auditing [100]. Insurance industry uses outlier detection in order to identify fraudulent claims. In addition to neural networks, another commonly used technique is to apply the Benford law [71]. The Benford law assumes that the numbers in a financial statement tend to follow a non-uniform distribution which is called the first-digit law. The law can be used to identify fraudulent account statements by detecting the accounts that do not follow this law. The trajectory outlier detection can be used in applications such as video surveillance and shipment monitoring [25] [85]. For example, irregularities in shipping routes may indicate terrorists or smuggling activities. In video surveillance, outlier detection is an important tool to identify malicious activities such as intrusion or product thefts. In stock market, time-series anomaly detection methods are used to alert unusual trends in stock price to prevent financial losses. Sequential anomaly detection is used in biology to identify unusual DNA sequences [122]. In criminal investigation, the outlier detection method can be used to identify the networks of terrorists or criminals by analyzing and detecting their unusual connections.

Outlier detection methods can be categorized based on training dataset availability, distribution assumption, or data type. In terms of training data, a method can be supervised, semi-supervised, or unsupervised. When there is a training dataset

for both outlier and normal observations, this dataset is used to train a selected data model to classify future input. For semi-supervised methods, the training data is either available for normal observations or outliers but not both. In this cases, the models are learned only for the known classes of observations. If a new observation does not belong to the known classes, it is declared to be an outlier. The limitation of supervised and semi-supervised methods is that the training dataset must represent all possible classes. If there is a new class of observations, this observation will not be detected correctly. In contrast, unsupervised outlier detection methods use similarities between observations to detect outliers. An observation that deviates from the masses is an outlier. The advantage of an unsupervised method is that no training dataset is required. However, the unsupervised outlier detection methods usually have much higher false alarm rates than the false alarm rates of the supervised and semi-supervised methods.

Based on distribution assumption, an outlier detection method can be categorized into either parametric method or nonparametric method. A parametric method assumes that the dataset follows some statistical model. The main objective of the parametric method is to learn the parameters of the model. An outlier is an observation that does not fit into the model well. Statistic based outlier detection methods usually fall into this category. A nonparametric method does not rely on any assumed distribution. Therefore, a nonparametric method can also be called parametric free method. The outlier detection methods using k-nearest neighbor algorithm such as distance-based and density-based outlier method are examples of the nonparametric methods.

Outlier detection methods can also be categorized based on data type. Tabular, sequential, and graph-based data are the three most common data types. Even though time-series, trajectory, and spatial data are also common data types, they can be transformed into tabular, sequential, or graph-based data. In the following sections,

we will discuss some important outlier detection methods using tabular, sequential, and graph-based data.

2.1 Tabular data

2.1.1 Statistical Methods

Statistical methods assume some distribution. The first approach is to compute the probability $f(x;\Theta)$ of an observation x where Θ is the set of parameters of the probability model $f(x;\Theta)$. If the probability is small, we consider that the observation x is an outlier. The second approach is to perform a hypothesis test.

In univariate data, we can perform the Grubb's test[56]. First, we measure the distance of an observation x from the mean of the population and normalize it with the sample standard deviation s. We obtain the *z-score* of x where $z = \frac{|x-\bar{x}|}{s}$. It should be noted that the mean $\bar{x}$ follows a normal distribution due to the central limit theorem. Since in a normal distribution, only 5% of the data has the *z-score* less than 1.97, one may consider x to be an outlier if its corresponding *z-score* is larger than 1.97. However, the mean in the formula above may include outliers. Therefore, we need to reconsider the bound for the *z-score*. We observe that the *z-score* also includes the sample standard deviation. As a result, the *z-score* cannot be larger than $(N-1)/N$ where N is the data size. Using this observation, we can conclude that x is an outlier if its *z-score* $> \frac{N-1}{N}\sqrt{\frac{t^2_{\frac{\alpha}{2N},N-2}}{N-2+t^2_{\frac{\alpha}{2N},N-2}}}$, where $t^2_{\frac{\alpha}{2N},N-2}$ is the t-value at a significant level of $\alpha/2N$.

In multivariate data, we can test the value of an observation against $\mu \pm 3\sigma$ if we assume the data follows a normal distribution $N(\mu,\sigma)$. Since 99.7% of data is within $\mu \pm 3\sigma$, we may consider an observation an outlier if it is outside this range. The parameters μ and σ can be replaced with sample mean and sample standard deviation if the data is large and the outliers are not very extreme. In other words, the method will fail to work if the outliers are too extreme for the sample mean

and standard deviation to represent the data correctly. Another alternative is to reduce the multivariate test to a univariate test by transforming the data using the Mahalanobis distance $y^2 = (x - \bar{x})' S^{-1} (x - \bar{x})$ [94]. In the formula, y^2 measures the deviation of x from the estimated mean. The inverted covariance matrix S^{-1} is used to normalize the data. We then can perform the Grubb's test on y^2 as discussed previously.

In statistics, regression analysis is used to estimate the relationship between attributes. Linear regression and logistic regression are two common models. An outlier in regression analysis is an observation whose value is far from the prediction. To detect such outliers, the residuals of the observations are computed based on a trained model. If a residual is large, its corresponding observation is declared as an outlier [59]. Another alternative is to apply Akaike Information Content (AIC) [7] for outlier detection. In regression analysis, AIC is a metric to evaluate a model. AIC has the following form $AIC = 2L(\hat{\beta}; x) + 2p$ where $L(\hat{\beta}; x)$ is a log-likelihood function with the maximum likelihood estimator β. A regression model is selected if its AIC is minimum. For outlier detection, with respect to normal observations, the outliers will have negative effects on model training. Therefore, a model will not be optimum if the training data contains outliers. As a result, we can use AIC for outlier detection by selecting a subset of the dataset such that AIC is optimum [60]. The data that are eliminated from the subsets are considered outliers.

2.1.2 Supervised Methods

In supervised methods, classification is used to detect outliers when the training data is available. A training data consists of observations and their corresponding class labels. The class labels indicate whether an observation is normal or an outlier. The classification-based methods consist of two phases: training and classification. The labeled observations are used to train a classification model to predict the class labels

of new observations in the classification step. The challenge is to select a model that can detect the outliers with a high detection rate. Even though there are different classification methods that can be applied to outlier detection, neural networks and decision trees are the two most common methods.

Neural networks have been widely used to detect credit card fraud [136]. A neural network is a mathematical model $F : x \mapsto y$ that maps an input x to an output y. A neural network is a set of functions that interconnect with each other. Each composition of some functions is considered as a neuron. These neurons form three layers: input, hidden, and output. The objective of neural networks is to select a set of functions from a class of functions such that the network yields the highest accuracy from the training dataset.

A decision tree is usually used to detect malicious attacks in network intrusion detection system [96][67]. A decision tree is a classifier that learns the correlations between attributes and class labels. Each interior node in a tree corresponds to an attribute. Each child node corresponds to a set of values of the parent node. The leaf nodes are the class labels. Given an observation, we start with the root node and then follow the tree based on the values of the observation until we reach a leaf node. The leaf node correspond to the class of the observation. There are several ways to build a decision tree. One commonly used method is to split an attribute using the concept of mutual information. The mutual information metric measures the dependency between two variables. In other words, it indicates how much one can infer one variable when another variable is know. When the mutual information is zero, the variables are independent. The mutual information is used to choose a split point of an attribute to maximize the mutual information between the attribute and the class labels. The set of split points are used to construct the decision tree.

The classification based outlier detection methods can detect known types of outliers accurately. However, it requires the dataset to have enough training data for

the outliers. This is challenging since the number of outliers is small. In general, the classification methods use some penalty functions to adjust the parameters of a model in order to yields a higher detection rate. However, for the problem of outlier detection, the overall high detection rate is irrelevant. A classifier with 99% accuracy can still miss 100% of these outliers. This problem is known as data skew in classification with minority classes [32]. Chawla et al [32] propose a boosting method that assigns higher weight to the minor classes in order to improve the accuracy for the minor classes.

2.1.3 Semi-supervised Methods

When training data is either available for normal observations or outliers but not both, the semi-supervised methods are used to produce the boundaries for the known classes. For example, when the normal observations are known, an observation that falls outside the boundary of normal observations is an outlier. The one-class support vector machine [127] is a commonly used for semi-supervised outlier detection.

Support vector machine (SVM) is a classification method using hyperplanes to partition a training dataset. Let say we have training data $\{x_i\}$ and corresponding labels $\{c_i\}$ where $c_i \in [-1, 1]$. The support vector machine method learns parameters w and b such that $c_i(w \times x_i + b) > 0$. The hyperplanes $w \times x_i + b = 1$ and $w \times x_i + b = -1$ are the separation hyperplanes at the boundaries of two classes -1 and 1. The distance between two classes is $\frac{2}{\|w\|}$. The objective of the support vector machine is to maximize the distance between the hyperplanes. Thus, it is equivalent to minimize $1/2\|w\|^2$ with the constraints $c_i(w \times x_i + b) > 0, \forall i$. This optimization problem can be solved using Lagrange multipliers. The support vector machine method assumes that the training data is linearly separable. A training data set is said to be linearly separable if two classes in the data can be separated by a hyperplane. For nonlinear separable data, we can apply the generalized support vector machine method which

uses the kernel method for class separation. It should be noted that the support vector machine method requires at least two classes in order to construct the boundaries. In the case, there is only one class, Schölkopf et al[127] apply the kernel method to transform the data such that the points close to the origin are treated as another class. Therefore, the origin and the training data are used to construct the decision boundaries. The problem can also be generalized to the problem of classification with multiple classes by constructing multiple hyperplanes for each class. If a new observation does not belong to any class, it is an outlier.

2.1.4 Unsupervised Methods

In unsupervised methods, clustering methods are used to detect outliers. A clustering method groups similar observations using some objective function. Compared with the classification methods, the clustering methods can group data without training datasets. The clustering methods can be used to detect outliers by comparing the observations with the identified clusters. If an observation is far from the cluster centroids, it is declared an outlier. For an example, one can use kmean [101] to identify cluster centroids and compute the distances of the observations from the centroids. The top extreme observations are outliers. Although kmean is fast, it cannot detect clusters with different densities or shapes. In such cases, we can use density-based clustering methods such as the kernel method, DBSCAN [53], or SNN [52] in order to identify cluster centroids. There are several limitations of using clustering methods for outlier detection. The goal of clustering methods is to detect clusters, not outliers. It is not optimal in outlier detection. For instance, an outlier close to a centroid can still be an outlier. An outlier may be incorrectly flagged as a normal observation, whereas a normal observation may be flagged as an outlier. In this section, we will discuss two algorithms, namely kmean and SNN.

2.1.4.1 Kmean

The kmean [101] clustering algorithm partitions a dataset D into k clusters. The objective of the algorithm is to minimize the sum of distances from the observations in the clusters to their cluster means. Suppose we have a dataset D with n observations $\{X_1, X_2, \ldots, X_n\}$. Assume that we have D partitioned into k clusters $C_1, \ldots, C_k$ with the corresponding means $\mu_i, i \in [1, k]$. The objective function of the kmean algorithm can be formally defined as follows:

$$\underset{C_1,\ldots,C_k}{\arg min} \sum_{i=1}^{k} \sum_{X_j \in C_i} \|X_j - \mu_i\|^2$$

The kmean algorithm consists of two steps: a clustering step and an update step. Initially, k distinct cluster means are selected either randomly or heuristically. In the clustering step, the kmean algorithm clusters the dataset using the current k cluster means. A cluster C_i is a set that consists of all the observations X_j such that:

$$C_i = \{X_j \in D | \min_{l \in [1,k]} \|X_j - \mu_l\|^2\}$$

In the update step, the cluster means are updated as follows:

$$\mu_i = \frac{\sum X_j}{|C_i|}, \ \forall X_j \in C_i$$

The kmean algorithm is run multiple times until the means converge or until the the number of iterations is greater than a chosen threshold N_θ. In general, the kmean algorithm is very efficient. However, the number of clusters depends on the parameter k. In addition, the kmean algorithm is not designed to detect clusters with different densities and shapes.

2.1.4.2 SNN

The shared nearest neighbor algorithm (SNN) [52] applies the concept of strong link and topic threshold to cluster a dataset. The strength of a link between two points is the number of shared nearest neighbors. If a point with the number of strong links is

greater than a chosen topic threshold, then that point is selected to be a core point. The non-core points are eliminated from the dataset. The SNN clustering algorithm will cluster the updated dataset. Two points belong to the the same cluster if there exists a strong link between them. As a result, the SNN clustering algorithm can cluster datasets with different shapes and densities.

2.1.5 Distance-based and Density-based

In order to overcome the limitations of statistical and clustering methods in outlier detection, Knorr et al [51] introduced a definition of outlier using distance. According to Knorr et al [51], the top farthest observations are outliers. The computation is $O(N^2)$ to compute pairwise distances, where N is the size of a dataset. Knorr et al introduce a pruning strategy in order to speed up the algorithm. The advantage of the method is that it can detect outliers without any assumption about the underlying distribution of a dataset. However, in some applications, the outliers of interest may not be the farthest observations. An outlier can be defined by its most similar observations instead. Therefore, Breunig et al [24] introduce a density-based method that detect outliers with respect to their local densities. An observation that is far with respect to its local region is considered an outlier. The method appears to be useful in practice. However, its complexity is $O(N^2)$ [28].

2.2 Anomaly in Sequential Data

We have discussed the outlier detection methods with tabular data. An observation is modeled as a tuple with a fixed number of attributes. In practice, there may be a natural order to the attributes and a variable number of attributes. This type of data is sequential data. There are a variety of such data in practice. For an intrusion detection system, the data of a user is a sequence of commands. In DNA analysis, the data is a sequence of alphabets. In text mining, the data is a sequence of characters or words of a document.

A sequence is defined as follows. Given a set of alphabets of size n, $\Sigma = \{c_1, c_2, \ldots\ c_n\}$, a sequence S of length k is a sequence of $s_1 s_2 \ldots s_k$ where $s_i \in \Sigma, \forall i \in [1, k]$.

We may reduce the problem of outlier detection in sequential data to the problem of outlier detection in tabular data by transforming sequences into tuples using a fixed number of attributes. A metric is selected to measure the similarity between these tuples. However, this reduction still does not consider the natural order in the attributes. Two sequences that are shown to be similar in tabular data may not be similar in sequential data [122]. Therefore, the problem of outlier detection in sequential data is distinct from the problem of outlier detection in tabular data.

2.2.1 Detecting Unusual Sequences

The first problem in sequential data is to detect an unusual sequence in a set of sequences. As in tabular data, we need to measure the similarity between the sequences. Sequences that are detected of being different from other sequences in the set are outliers.

The main approach is to apply a Markovian model to estimate the probability of one value in a sequence with respect to the preceding values in the sequence, e.g. $Prob(s_1 s_2 \ldots s_k)$. The set of input sequences is used to construct the probabilities.

Using the Bayes' theorem, the probability $Prob(s_1 s_2 \ldots s_k)$ can be written as:

$$Prob(s_1)Prob(s_2/s_1)Prob(s_3/s_1 s_2) \ldots Prob(s_k/s_1 s_2 \ldots s_(k-1))$$

Therefore, we can construct a tree representing the conditional probability of a new value given a set of known values. The challenge is that the length of a sequence can be very long. However, in some applications, it is shown that they may have a property of short memory [41] such that $Prob(s_1/s_1 s_2 \ldots s_k) \approx Prob(s_1/s_1 s_2 \ldots s_L)$ for $k > l$. This property implies that we do not need to construct the tree for a sequence of more than L alphabets. Therefore, the maximum height of the tree will be L. Ron et al [41] introduce a Probabilistic Suffix Automata (PSA) to model the

conditional probability. Probabilistic Suffix Tree (PST) is an efficient implementation of PSA [41].

The PST is used for pattern mining in sequential analysis. For instance, Yang and Wang introduce a method to cluster sequences using PST[76]. In their method, they construct PST for each cluster and perform the membership test of a new sequence by using the Odds measure:

$$SIM_O = \frac{Prob^T(s_1)Prob^T(s_2/s_1)Prob^T(s_3/s_1s_2)\dots Prob^T(s_k/s_1s_2\dots s_(k-1))}{Prob(s_1)\dots Prob(s_k))}$$

The idea is that the probability of a sequence computed from the PST of the cluster to which it belongs must be larger than a random probability $Prob(s_1)\dots Prob(s_k)$. This can be extended to outlier detection. If a sequence does not belong to any cluster, it is an outlier. However, since the method is designed for clustering, it is inefficient for outlier detection. We need to construct PST for all the clusters. Sun et al [122] observe that outliers usually appear close to the root nodes. They suggest pruning the nodes that are far from the root by either using the count of the number of sequences in the node or the total probability of a sequence at the current node. Since only a fraction of the tree is computed using this method, it can boost the computation by many orders of magnitude. In addition to pruning, Sun et al also propose to use Normalized measure, $SIM_N = \frac{1}{l}(\log(P^T(s_1) + \sum_{j=2}^{l} \log P^T(s_j|s_1\dots s_{j-1})$ to compute the similarity between sequences. According to their evaluation, the Normalized probability similarity measure outperforms the Odds measure in terms of outlier detection.

Figure 1 shows an example of a PST tree. Each node contains a sequence of alphabets starting from the root node, the total number of corresponding sequences and its probability. Since the number of sequences starts with YX is only 10, the child nodes corresponding to YX will not be constructed.

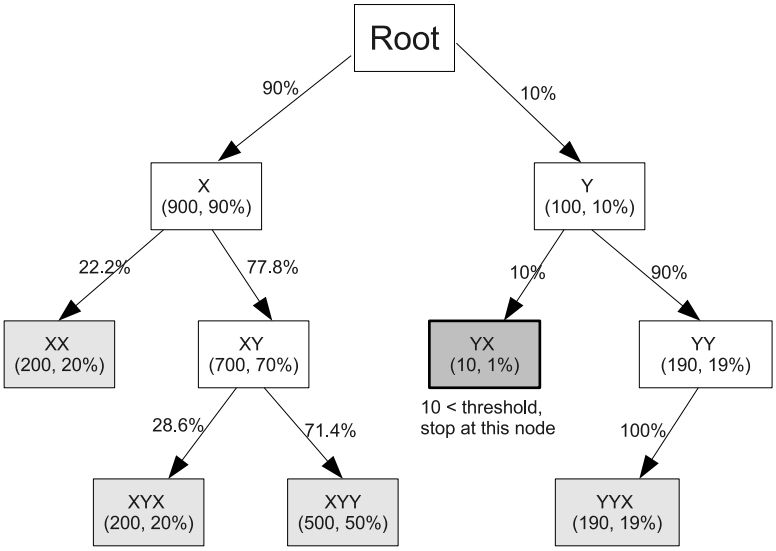


Figure 1: An example of a PST tree

2.2.2 Detecting Unusual Subsequences in a Sequence

Another problem of outlier detection in sequential data is to detect unusual subsequences in a long sequence. In this problem, we assume that the values in the sequence follow some patterns, e.g. $Prob^T(s_2/s_1)$. If a subsequence does not follow any pattern, it is unusual. The challenge is that a sequence can be very long and the length of an anomalous pattern can vary. We can use a brute force method to produce all possible subsequences in a long sequence and then compute the subsequences that deviate from the other subsequences. This method consists of two iterations. The first iteration is to compute all possible subsequences. For each subsequence, it is compared against all the possible subsequences. The computation for this method is expensive. Keogh et al [84] propose a pruning strategy to speed up the algorithm. Keogh et al observe that if we can find some subsequences similar to the current subsequence, we can safely prune the subsequence since it is not considered unusual. It leads to another observation. If there exists an order between subsequences such that the most similar subsequences are in the top, the current normal subsequences can be pruned quickly. The authors also observe that we do not need a perfect order

for the algorithm to be fast. The order just needs to be good enough so that the computation is efficient. Therefore, Keogh et al propose to use Symbolic Aggregate ApproXimation (SAX) [98] to approximate the order. As a result, their method can detect unusual subsequences in a long sequence efficiently.

2.2.3 Detecting Unusual Values in a Sequence

In this problem, we are interested in detecting an unusual value in a sequence of values [140] [126]. An example is to detect anomalies in network traffic flows [92]. The simplest case of this problem is when the variance of the values in a sequence is small. In such cases, we can compute the mean μ and standard deviation σ of the sequence. The interval $\mu \pm k\sigma$, where k is a natural number, corresponds to the upper bound and lower bound of the incoming values. A value that falls outside the bounds is unusual.

For more complex sequences, there are two methods. The first method is to compute the average and standard deviation of the most recent values in a sequence. The average and standard deviation are the predicted value and the bound of the incoming values. The performance of this method depends on window size. If the window is small, we may not have enough data to predict the value correctly. In the other hand, if the window is large, the old values may affect the accuracy of the predicted value. One possible solution is to assign the weight to the past values such that the more recent values have higher weights. Another method is to model the sequence using the Markovian model [131] to compute the probability of a new value using the past values. A value is flagged unusual if its probability is small. These two methods use the most recent values to detect anomalies in a sequence. However, both of them do not consider seasonal effect that may occur in the sequence. A solution is to fit a periodic function into the data model [87] so that the seasonal effect will be considered in the model.

2.3 Graph-based Anomaly

Recently, graph mining has been studied extensively in the area of data mining. Some examples of graph mining are discovering frequent subgraphs, mining correlation rules between subgraphs, detecting communities in a graph. Graph-based anomaly detection focuses on anomalies in a list of graphs or detecting anomalies within a graph. One possible solution for the first problem is to reduce this type of anomaly to point-based outlier detection by transforming the graphs into multidimensional records using graph kernels [23] [138]. However, the graph structure may be lost during the transformation.

Noble et al [113] propose the use of SUBDUE [38] for detecting anomalous substructures in a graph. The SUBDUE method identifies repetitive substructures in a graph [38]. The SUBDUE method uses Minimum Description Length (MDL) to identify substructures in a graph. The concept of MDL corresponds to the minimum number of bits used to describe an information. The objective of the SUBDUE is to identify a substructure in a graph that use the minimum of extra information to describe the graph. In their definition, a substructure is anomalous if it is infrequent. However, the authors observe that the frequency of a substructure is inversely proportional to its size. In other words, a large graph tends to less frequent. As a result, the degree of anomaly should be inversely proportional to its size. The authors multiply the frequency of a substructure with its size to take the size into account.

The graph-based anomalies discussed above are based on a static graph. Another problem of graph-based anomaly is anomaly detection in a dynamic graph. We can consider a dynamic graph as a network of messages being passed between the nodes. One example of anomaly detection in problem is to detect disruption in a network [72]. An unusual disruption in a network can be used to detect system failures or network intrusion activities.

2.4 Robust Outlier Methods

Researchers are not only interested in detecting outliers in datasets but also interested in developing methods that minimize the effect of outliers in data analysis. These methods are called robust outlier methods. Statistics that are robust to outliers are called robust statistics. In clustering methods that assume some underlying distribution, the appearance of an outlier can make estimated parameters to be incorrect [22]. For example, the mean of a Gaussian used in modeling a cluster can be affected by an extreme outlier. In classification, some methods are sensitive to outliers [86]. For example, support vector machines use the points on class boundaries to construct the hyperplanes which separate the observations from different classes. An outlier can make the hyperplane construction to be far from the true class boundaries. This results in classification with a high error rate. In regression analysis [128], the outliers may affect the estimated slope of a regression model. In order to reduce the effect of outliers on data analysis, we can run an outlier detection algorithm to remove the detected outliers from the dataset. Another alternative is to develop data analysis methods that are robust to outliers [137]. For example, one can use M-estimation for regression analysis. M-estimation [73] is a generalization of Maximum Likelihood Estimation (MLE) . The MLE method is a method to estimate the parameters of a density function by selecting the values that minimize the product of the likelihood functions. It is equivalent to minimizing the sum of the log likelihood functions. The MLE has the following form:

$$\hat{\theta} = \arg\min_{\theta}(-\sum_{i} \log f(X_i, \theta))$$

The function $f(X_i, \theta)$ is a density function and X_i is an observation. The M-estimation replaces the density function with any function $\rho(X_i)$ with some properties. By replacing the density function $f(x)$ with a function $\rho(x)$, the M-estimation method can be more resistant to outliers than the MLE method [105].

Table 1: Outlier detection methods categorized based on data type and the availability of training data

	Supervised Method	Semi-supervised Method	Unsupervised Method
Tabular	Classification (SVM, etc.)	One-class SVM	Statistical-based Distance-based Density-based Cluster-based
Sequential			Markovian methods (PST, etc.)
Graph-based	Classification using graph kernel		SUBDUE

Table 2: Outlier detection methods categorized based on distribution assumption and the availability of training data

	Supervised Method	Semi-supervised Method	Unsupervised Method
Parametric	Fisher discrimination		Statistical-based
Non-parametric	SVM	One-class SVM	Distance-based Density-based Cluster-based Markovian methods (PST, etc.)

2.5 Summary

Table 1 shows a summary of the outlier detection methods categorized based on data type and the availability of training data. Table 2 shows the summary based on distribution assumption and the availability of training data. In our research, our outlier detection methods fall into the category of non-parametric unsupervised outlier detection using tabular data. In this category, our contribution is to develop efficient and accurate outlier detection methods.

CHAPTER III

EFFICIENT LOCAL DENSITY-BASED OUTLIER DETECTION USING RANDOMIZATION

3.1 Motivation

Recently, a survey of different outlier detection methods [28] has appeared in the literature. Accordingly, we can roughly categorize the methods into parametric and nonparametric methods. The advantage of nonparametric methods is that they do not require the prior knowledge of the processes that produce the events (e.g. data distribution), which makes these methods capable of detecting outliers in any dataset. The nonparametric methods can be classified as globally based and locally based. The globally based methods [51] identify the observations that are considered to be the top outliers with respect to distance for the entire dataset. The locally based methods [24] compute the outliers with respect to their local subspaces. In this chapter, a subspace is a subset that is created from a dataset by using hyperplanes to partition the dataset. The nonparametric locally based methods are introduced by Breunig et al [24] by using the concept of local outlier factor (LOF). The method computes the LOF score for each point by computing the ratio between the k-nearest neighbor distance of the query point with the average of those of its neighbors. Since in real applications, an observation is, generally, an outlier because it deviates from its local region rather than the entire dataset, the LOF method is shown to be very useful in practice [28] [104] [89] [95].

In order to compute the LOF factor, we have to compute the k-nearest neighbors for the entire dataset. For a dataset of size n, the time complexity is $O(n^2)$. This is expensive. In very low dimensional data, one may use indexing methods to speedup

the nearest neighbor searches, namely R*-tree [17], X-tree [18], Kd-tree [57], etc. The main idea of the indexes is to create a hierarchical tree of boxes. A box represents a set of points and the boundaries of the those points. The box is further split into smaller boxes. In order to query a k^{th}-nearest neighbor for a given point, we start from the root and traverse the index tree for the box that contains the query point until we reach a leaf. The k-nearest neighbors are in the leaf and its surrounding boxes. This is efficient for very low dimensional datasets because the number of boxes to search is small. However, the number of boxes exponentially increases with the number of dimensions, so that we have to compute the distance for most of the points in the dataset. The approximate search for k-nearest neighbors can be used to improve the performance by relaxing the problem from computing the exact k-nearest neighbors to approximate k-nearest neighbor with some delta errors. However, the original problem in fact still exists and this approach also performs poorly with high dimensions. This is known as the curse of dimensionality [129].

Therefore, it is challenging to compute LOF for high dimensional datasets[28]. In this chapter, we show that it is possible to compute LOF efficiently for dataset with very high dimensionality without the limitation of using index trees. The method is made possible by our observations of the outlier consistency property of local outliers, which will be discussed later in the formalism section. Therefore, we can employ a randomization method to compute LOF. The implementation of this method requires small modifications to the existing LOF method.

From now on, we will refer to the original version of LOF with full k-nearest neighbor computation as the **nonrandomized version** of LOF. In the following sections, we will formally define the randomized method. In the experiment section, we will evaluate the effectiveness and efficiency of the **randomized version** against the nonrandomized version of LOF.

3.2 Related Work

Outliers have been studied extensively in the field of statistics [65] by computing the probability of an event against its underlying distribution. However, this method requires prior knowledge about the process that produces the events, which is usually unknown. Knorr et al [51] introduce a distance-based method to identify the outliers. The outliers are those points whose distance to other observations are the largest. Their method can detect global outliers. The advantage of this method is that no prior knowledge about the underlying distribution is required.

Breunig et al [24] introduce a local density based method for outlier detection. An outlier is a point that deviates from its neighbors. The local outlier factor is measured by the ratio of its distance to its neighbors and the local density. The outliers in this definition are locally based.

Spiros et al [132] introduce the method to detect outliers by using the multi-granularity deviation factor (MDEF). The authors then propose an approximate version to speed up the method. The method is based on the modification of an approximate nearest neighbor search algorithm (quad-tree) in order to avoid the cost of computing the MDEF scores for all the points in the dataset. Thus, the method depends on the performance of the index tree.

Recently, Kriegel et al [89] propose the angle-based method that computes outlier scores based on the angles of the points with respect to other points. The method aims to provide more accurate rankings of the outliers in high dimensions. However, the method can not detect outliers surrounded by other points. The naive implementation of the algorithm runs in $O(n^3)$.

Since LOF is a useful method to detect outliers, we will focus on improving the LOF method to make its running time fast in very large and high dimensional datasets.

3.3 Local Outlier Factor

In this section, we revisit the concept of local density-based outlier introduced by Breunig et al [24]. The local density-based outlier is based on the k-nearest neighbor distance, the local reachability and local outlier factor which Breunig et al formally defined as follows, I quote:

Definition 1 (k-distance of an object p)**.** *"For any positive integer k, the k-distance of object p, denoted as k-distance(p), is defined as the distance $d(p, o)$ between p and an object $o \in D$ such that:*

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

Definition 2 (k-distance neighborhood of an object p)**.** *"Given the k-distance of p, the k-distance neighborhood of p contains every object whose distance from p is not greater than the k-distance, i.e. $N_{k\text{-}distance(p)}(p) = \{q \in D \setminus \{p\} | d(p, q) \leq kdistance(p)\}$."*

Definition 3 (reachability distance of an object p with respect to object o)**.** *"Let k be a natural number. The reachability distance of object p with respect to object o is defined as $reach\text{-}dist_k(p, o) = \max\{k\text{-}distance(o), d(p, o)\}$."*

Definition 4 (local reachability density of an object p)**.** *"The local reachability density of p is defined as*

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

"

The k-distance of an object and its corresponding k-distance neighborhood are used to define the reachability of a point o with respect to p. The reachability of o

to p is the actual distance between p and o by the k-distance of o if o is sufficiently close to p. The purpose is to reduce the fluctuation of the outlier computation.

The local reachability density is the inverse of the average reachability distance of the neighbors of p. It measures the local density of p. The parameter $MinPts$ is used as a parameter of the local density density. It is the same as the parameter k to compute the *k-distance*. The parameter $MinPts$ specifies the number of neighbors used to compute the local density.

Definition 5 (local outlier factor of an object p)**.** *"The local outlier factor of p is defined as*

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

"

Intuitively, the local outlier factor (LOF) of p is the ratio between the average local reachability of its neighbors and its local reachability. Breunig et al also computed the lower bound and upper bound for LOF under some constraints. If p is in a deep cluster, the local outlier factor is close to 1. If p is outside the clusters, it is greater than 1. The local outlier factor measures the degree of the local deviation of p with respect to its neighbors.

3.4 Generalized Local Density-based Outlier

We observe that the main idea of the local outlier factor is in fact similar to computing the ratio between the distance from p to its nearest points with the density of its local subspace, in order to identify local outliers. Breunig et al measure the local density by using the average k-distance of the nearest neighbors of p. This metric, however, can be generalized to other local density functions without affecting the meaning of local density-based outlier. A reasonable choice can be a kernel density function. In this study and in the following theorems, we measure the local density by the average

closest distance between the points in S ($density(S)$). We say that S is approximately uniform if the following two conditions hold. The variance of the closest distances is less than a small ϵ and there is no k-nearest distance is larger than the average k-nearest distance with ϵ unit.

We also observe that if the distance of p to its nearest points is much greater than the density of any subset in D ($dist(d, S)$) that is approximately uniform, then p is not in any cluster. Clearly, p is an outlier in D. On the contrary, if there is a subset S' such that the difference is small, then p is likely to be generated from the same distribution of S'. We can not conclude that p is an outlier, so p is considered to be normal.

These two observations lead to the conclusion that the ratio between the distance and the density must be high for all the subsets in the dataset for a point to be an outlier. Thus, we can define a locally based outlier as follows:

Definition 6. *Given a point p, a dataset D, and for any subset S of D such that S_i is approximately uniform, then p is an outlier with respect to D iff* $\frac{dist(p,S)}{density(S)} \gg 1$.

Figure 2 illustrates two outliers p_1 and p_2 based on this definition. In the figure, p_1 is not only a local outlier for the cluster containing S_1, but p_1 is also an outlier with respect to S_2, S_3, and S_4. Similarly, p_2 is also an outlier with respect to S_1.

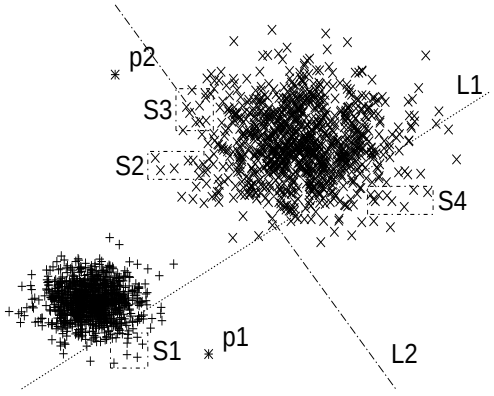


Figure 2: Outliers with respect to their local subspaces.

By this definition, we observe that if we take a random hyperplane to partition

a dataset into two subsets, then in most of the cases, the local outlier factors will not change dramatically. Hence, we can recursively partition the subsets into smaller subsets. We can partition the dataset until the subsets are small enough for us to compute the local outlier factors efficiently. As we see, we do not need to perform the nearest neighbor computation for the entire dataset in order to detect the local outliers.

Figure 2 illustrates an example of the partition. L_1 and L_2 partition the dataset. $S_1 \ldots S_4$ are unchanged after the partitions. L_2 cuts S_3 into two subspaces S_3' and S_3''. We see that S_3' and S_3'' are still approximately uniform after the partition. The points p_1 and p_2 remain to be outliers in the new subsets partitioned by L_1 and L_2.

The procedure to detect the local outliers assumes that the partition does not affect the density for the partitioned sets. There are two cases where it can go wrong. The first case is when a point q is on a cluster boundary and the partition isolates it from the cluster it belongs to. If the distance between the local subspace of q and the new clusters in the subset it belongs to is large, then q is incorrectly identified as an outlier with respect to the new clusters. The second case is when there are many points like q that are separated from their clusters. It may make an outlier p to be normal in the new subset contains only these points.

These problems in fact can be avoided if during the separation, the new subsets contain enough neighbors of these points. Fortunately, it can be shown that the probability of partitions that separate a normal point from all of their neighbors is small. This is due to the fact that if a set C which contains q (on the cluster boundary) is large, then the probability of drawing a hyperplane cutting C such that it only contains q is small.

Theorem 1. *Given a set D, a point p on its boundary, the probability of selecting k-nearest neighbors of p or less is k/N.*

Proof. The probability to choose a value k is $1/N$. Thus, the probability to choose

up to k-nearest neighbors is $\sum_{i=1}^{k} \frac{1}{N} = \frac{1}{N}$. □

If p is not an outlier, it should belong to a large cluster. This implies that $k \ll N$. The theorem shows that the probability of a point p being on the boundary to be separated from its cluster is small. This is an important observation because we can detect the local outliers effectively using randomization. If we randomly partition the dataset performed multiple times, in most partitions, q will appear to be normal. Thus, if a point appears to be normal in most partitions, then we can flag it as normal with high confidence.

The observations above are the principles of the randomized method for computing outliers by randomly partitioning a dataset and running the algorithm multiple times so that the false outliers can be ruled out.

3.5 Algorithm

The randomized algorithm is described in Figure 3. In this algorithm, PARTITION (Figure 4) takes a dataset D as input. Then, it will call SPLIT (Figure 5) to split the dataset into two subsets S_1 and S_2 in the following way. SPLIT randomly selects two points p_1 and p_2 in D. For every point in D, SPLIT computes the distance from it to p_1 and p_2. D will be split into S_1 and S_2 where S_1, S_2 contain all the points that are closer to p_1, p_2 respectively. This SPLIT is equivalent to choosing a hyperplane P to partition the dataset. Then, for $S \in \{S_1, S_2\}$, if the size of S is still greater than a threshold M_θ, PARTITION will be applied to S. This recursive PARTITION will be performed until the size of the result sets are smaller than a chosen size of M_θ. At this point, the LOF for all the points in S will be computed with respect to S. M_θ should be greater than the parameter $MinPts$ of LOF. Other than that, it can be any value that allows the outlier detection to be computed efficiently.

In the end, we will have all the outlier scores for D. As discussed in section 3.4, the result set of outliers may contain false outliers due to isolated points. Therefore,

```
1: procedure COMPUTEOUTLIER(Set D, N_iter)
2:     for all i ∈ [1, N_iter] do
3:         PARTITION(D)
4:     end for
5:     for all i ∈ [1, N_iter] do
6:         COMBINESCORES
7:     end for
8: end procedure
```

Figure 3: Randomized local density-based outlier method

we run PARTITION multiple times to rule out the false outliers. The final LOF for each point will be its minimum score over all the iterations. We use the parameter N_{iter} to set the number of iterations of the algorithm. According to the experiments, the output tends to be stable with $N_{iter} = 10$. We can speed up the algorithm by filtering points with low scores that are less than a threshold δ_{out}. The points with the scores computed in the first few iteration less than δ_{out} will not be considered in the next iterations.

It is expected that there will always be some small differences in the rankings between the original method and the randomized method. In the original LOF method, the ranking depends on $MinPts$. The choice of $MinPts$ is subjective. A small change in $MinPts$ will lead to a change in the ranking by LOF. Therefore, it is acceptable for the ranking to be slightly different. In the case that a more similar LOF ranking is desired, we can recompute the outlier scores for the top N outliers by using the original nonrandomized version. It will give the exact score for these points. The number of top outliers is small, thus the computation time is cheap. We call this version the **recompute version** of the randomized method, while we call the earlier one the **naive version**.

We can also run the algorithm multiple times with the new final score being the average of all the runs. We call it the **merge version**. We notice that even though the recompute version can produce a ranking which is nearly the same as the ranking

```
procedure PARTITION(Set D)
    SPLIT(D, S_1, S_2)
    if |S_1| > M_θ then
        PARTITION(S_1)
    else
        COMPUTECANDIDATES(S_1)
    end if
    if |S_2| > M_θ then
        PARTITION(S_2)
    else
        COMPUTECANDIDATES(S_2)
    end if
end procedure
```

Figure 4: Compute candidate outliers

```
procedure SPLIT((Set D, Set  S_1, Set S_2)
    p_1 ← random(S)
    p_2 ← random(S)
    for all p ∈ D do
        if ||p, p_1|| < ||p, p_2|| then
            put p into S_1
        else
            put p into S_2
        end if
    end for
end procedure
```

Figure 5: Partition a set using random split points

of the nonrandomized version, it is limited to the top N outliers. On the other hand, the output of the merge version is less similar for the top outliers, but the similarity can be improved for all the points. Thus, we first produce the average outlier scores using the merge version, then we recompute the score of the top outliers (**hybrid version**). Finally, we have rankings similar to that of the nonrandomized method for all the outliers.

3.5.1 Query Time of New Point

The partition can actually be treated as the creation of a binary tree with two-key nodes. Each key represents a new subset. The two keys are the selected points (called

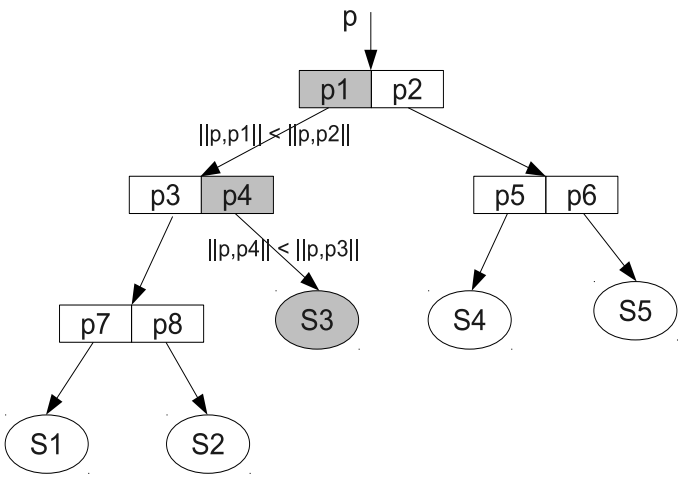


Figure 6: Tree structure of randomized LOF

split points) for the partition. Each key has a pointer to the child node. The structure is then recursively created. A leaf node is a node which represents a subset which will not be further partitioned. The leaf node contains all points of the subset. The keys of the root node are the first two randomly selected points. To traverse the tree, we start with the root node. We compare a query point p to the keys of the parent node and choose the key which is closest to p. Then, we traverse the tree to the child node referred by this key. We repeat this process until we reach a leaf node where we will compute the outlier score for p with respect to the leaf node.

An example of the tree structure is illustrated in Figure 6. Points p_1, p_2 are two split points at level one. p_3, p_4, p_5, and p_6 are split points at level two. The set represented by p_2 is partitioned into two subsets S_4 and S_5. We do not continue to partition these two subsets since the sizes of S_4 and S_5 are less than M_θ. We will have the final subsets S_1, S_2, S_3, S_4, and S_5. For a new query point p, we compare the distance $\|p, p_1\|$ and $\|p, p_2\|$. Since p is closer to p_1 than p_2, we traverse to the left child. At level two, p is closer to p_4, so we traverse to the right child. The right child is the final subset S_3. We will compute the candidate outlier score for p with respect to S_3.

The example illustrates how we can use a binary tree with two-key nodes to represent the algorithm. We maintain such trees to query the outlier score of a new point. Otherwise, we do not need to maintain the trees. The space to store the tree is $O(n)$ where n is the dataset size.

As we discussed earlier, we will maintain multiple trees for ruling out false outliers. The number of trees corresponds to the number of iterations. The score of a point will be the minimum score computed from all the trees. The time complexity of a query is $O(h + f(M_\theta))$, where h is the height of the tree and $f(M_\theta)$ is the time required to compute the outlier scores for the subset. If the trees are balanced, the number of steps to reach the leaf nodes is $O(\log n)$.

3.5.2 Time Complexity Analysis

We use the tree structure discussed in section 3.5.1 to analyze the time complexity of the algorithm. The algorithm consists of three main steps: partition, outlier score computation for local sets, and merge.

The *partition step* is fastest when the tree is perfectly balanced. Multiple partitions are required until the subsets are less than M_θ. For each level h, there are 2^h subsets, the size of each set is $\frac{n}{2^h}$, thus the partition cost at this level is $O(2^h \times \frac{n}{2^h}) = O(n)$. The total time for all levels is $O(H \times n)$, where H is the height of the tree. If the tree is balanced, $H \approx \log n$. The total time will be $O(n \log n)$. In the *outlier score computation step*, we consider it a constant $O(c)$ because the sizes of the subsets are very small. The maximum number of subsets is n, the worst time complexity to compute the scores is $O(n)$. In the worst case, the *merging process* for different runs and iterations can be done in $O(n)$.

In total, the upper bound for the balanced tree is $0(n \log n)$. In practice, we may not have a balanced tree, however, if we assume that most of the time the ratio of the sizes of subsets after a split is a reasonable value, the time complexity can be roughly

approximated as in the balanced tree. It is possible that a partition may result in two completely unbalanced subsets where one set contains most of the points. Therefore, the key is to ensure that the probability of completely unbalanced subsets is rare. Fortunately, we can show that under some assumptions such extreme cases are rare.

Theorem 2. *Given a uniformly distributed set D, a point p, and a random hyperplane P that divides D into S and S' where $p \in S$, we have $prob(|S| < k) \ll 1$, where $k \ll |D|$.*

Proof. According to the definition of the uniform distribution, the probability of choosing a subspace S in D is roughly the same as $\frac{|S|}{|D|}$. Thus, $prob(|S| < k) \leq \frac{k}{|D|}$. Since $k \ll |D|$, we have $prob(|S| < k) \ll 1$. □

Theorem 3. *Let say we have a set D which consists of D_1 and D_2 where $|D_1| \ll |D_2|$ and D_2 is uniformly distributed, a point p, and a random hyperplane P divides D into S and S' where $p \in S$, we have $prob(|S| < k) \ll 1$, where $k \ll |D|$.*

Proof. There are two cases. The first case is that at least one of two representative points p or q belongs to D_1. Since $|D_1| \ll |D_2|$, we have $prob(p \in D_1 \vee q \in D_1) \ll 1$. The second case is where both p, q belong to D_2 and we can apply Theorem 2 on D_2. We conclude that $prob(|S| < k) \ll 1$. □

In such datasets the probability of producing two unbalanced subsets is rare, which means that the probability for the algorithm to approach $O(n^2)$ is small. We see that the theorems can be extended to any dataset with multiple uniform distributions. The speed is guaranteed under this assumption; however, in practice, when the assumption is relaxed, the algorithm can still yield fast performance.

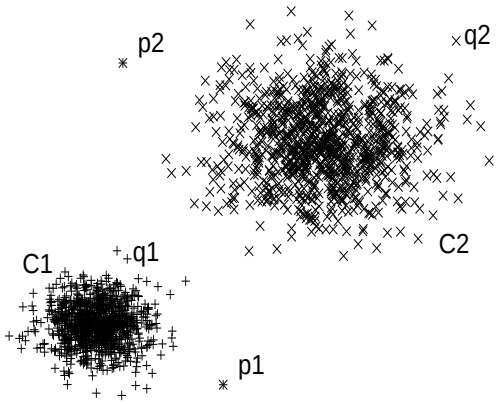
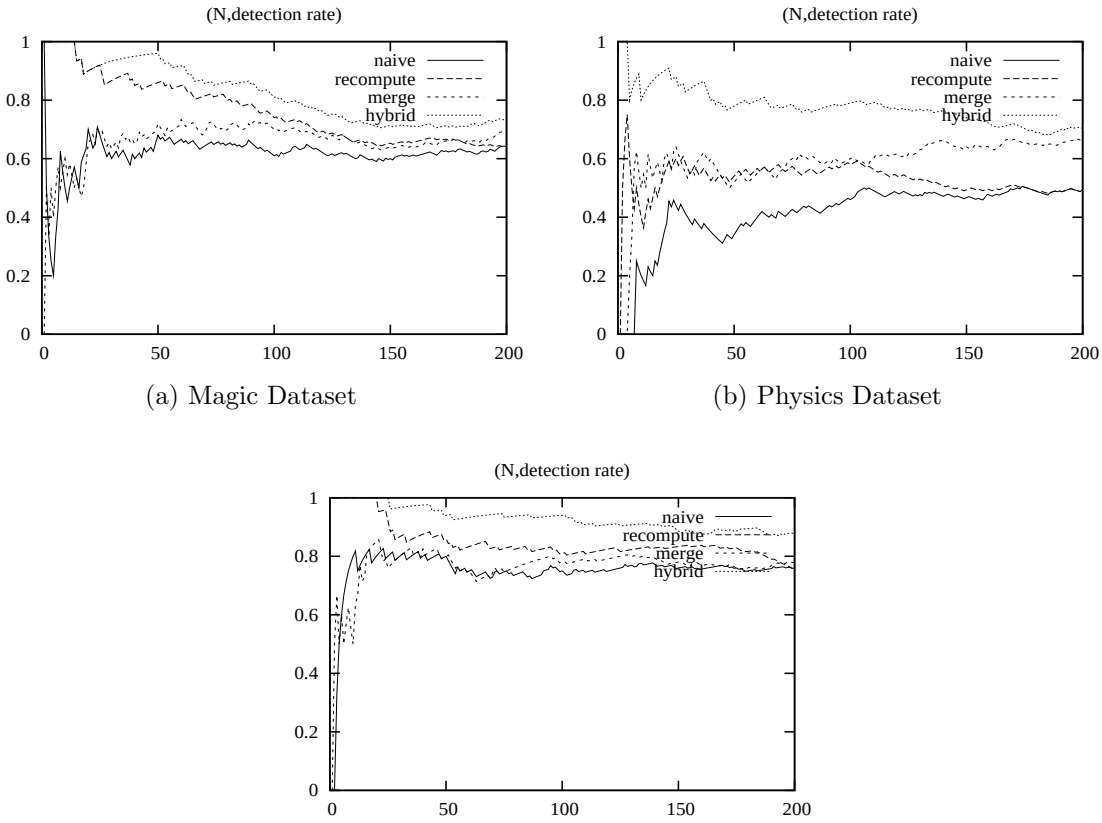


Figure 7: 2D test data for local outlier

3.6 Experiments

3.6.1 2D Example

We use a two dimensional dataset to show that the randomized method can detect local outliers correctly. We generate two Gaussians with different means and standard deviations. We then generate two local outliers p_1 and p_2 for clusters C_1 and C_2. The dataset is illustrated in Figure 7. First, we compute the outlier scores using the nonrandomized version. The LOF method detects two outliers p_2 (2.75) and p_1 (2.4) as two top outliers. In addition, it returns two other outliers q_2 (2.2) and q_1 (2.1). These outliers are synthetically generated by the Gaussian. Then, we compute the scores using the merge version. We set $M_\theta = 100$ and $N_{run} = 6$. The points p_2 and p_1 are consistently detected as the top two outliers for all the different runs. Their final scores are 2.65 and 2.35 respectively, which are very close to the original scores. In contrast with p_1 and p_2, the rankings for q_2 and q_1 are not consistent. However, when using the merge version, they were ranked correctly. The scores are 2.1 and 1.9 respectively. The experiment shows that the randomized method is as good as the original method using full nearest neighbor computation. In some cases, some outliers may be ranked differently but on the average the output of the randomized method converges to the original method.



(a) Magic Dataset

(b) Physics Dataset

(c) KDD CUP '99 Dataset

Figure 8: Similarity in ranking for randomized outlier detection

3.6.2 Real Datasets

3.6.2.1 Dataset Description

We evaluate the performance of our method against the original LOF method with three different datasets: MAGIC Gamma Telescope [112], Physics [30], and KDD Cup '99 Network Intrusion [112]. The details of the datasets after normalization and after removing nonnumerical attributes are shown in Table 3. We will refer to the outliers computed by the nonrandomized LOF as nonrandomized outliers.

Table 3: Magic, Physics, and KDD Cup '99 dataset description

Dataset	**D**	**N**
Magic	10	19K
Physics	69	50K
KDD Cup '99	34	80K

3.6.2.2 Evaluation Metrics

Before proceeding with the experiments, we first discuss the metrics for evaluating the effectiveness of the randomized method. The LOF method returns two values which are the local outlier factor (which we call score in our method) and the ranking of the points according to the local outlier factor. We observe that the LOF method is sensitive to the parameter $MinPts$. With the same LOF method, a small change in $MinPts$ can lead to changes in the ranking and the scores. Except for very strong outliers where the scores are distinct, the ranking is sensitive to the scores. It is even more sensitive for points with low outlier scores. For an example, there is not much difference between the rankings of 200 and 203 for the outliers with the scores of 1.90 and 1.85 due to the statistical variation. Therefore, the objective is not to have the exact same scores and rankings between the original and randomized versions. Instead, the main objective is to have similar scores with some acceptable statistical variation. Since we are interested in the top outliers, we try to preserve the ranking for these outliers. This preservation is important if there are strong and distinct outliers in the dataset. Therefore, we evaluate the method using the following metrics:

We use the "detection rate" $\frac{N-N_{lof}}{N}$ to evaluate the top outliers, where N_{lof} is the number of outliers in the top N outliers in our method that also appear in the top N nonrandomized outliers. According to the experiments, the scores drop quickly when the points are outside the top 100 outliers, which makes the ranking sensitive to small changes of the scores. Thus, we vary N up to 200 outliers. For the weak outliers, we compute the mean and standard deviation of the ratios of the absolute

differences between the methods for every point. If they are small, the two methods produce similar scores.

3.6.2.3 Effectiveness of the Randomized Method

We evaluate the effectiveness of the randomized method as follows:

First, we run the nonrandomized LOF on the datasets to compute the outlier scores ($minpts = 20$). Then, we run the randomized method on the datasets ($M_\theta = 500$, $N_{iter} = 20$, and $N_{run} = 6$). The results are shown in Figure 8a, 8b, and 8c. In all the figures, the naive version performs worst in comparison with the others. Nonetheless, in all the experiments, it still guarantees the detection rate of 40% for $N = 25$. It means that at least the top ten outliers are detected. The method performs best for the KDD dataset where the top 20 outliers are identified. The merge version produces slightly better results for the Magic and KDD datasets. At least 50% of the top 50 outliers are detected. The performance of the merge version is more stable compared with the naive versions when N increases. As expected, the recompute version boosts the performance for all the datasets. In the figures, all top five outliers are correctly detected. At least 80% of the top 50 outliers are detected in the Magic and KDD datasets. However, the differences in the rankings start to increase when N increases. By using the hybrid approach, the performance of the randomized version becomes stable with high accuracy. As we can see, this approach is the best in all the experiments.

By manually examining the results, we found that the KDD dataset contained many strong outliers. The outlier scores for the KDD dataset are high while those in the Magic and Physics datasets are low. It can be explained by the fact that the KDD dataset contains many intrusion attack connections. This makes the distinction between the outlier scores in the KDD dataset more obvious. Therefore, the results in the KDD dataset are more stable than those in the other two datasets.

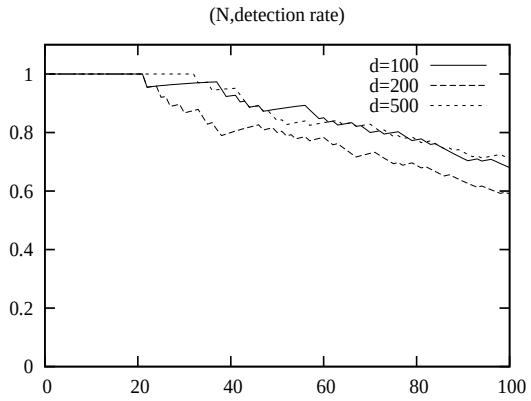


Figure 9: Effect of dimensionality on the detection rate.

For the weak outliers, we compute the mean and standard deviation as mentioned earlier. We found that the top 180, 167, and 151 outliers had the exact same scores with the outliers computed by the original LOF in the Magic, Physics, and KDD datasets respectively. The statistics imply that our method and the original LOF method produce similar results.

3.6.3 Dimensionality

We want to answer the question whether the effectiveness of the randomized method will also be reduced by the "curse of dimensionality" as is the case for index trees. We generate synthetic datasets with the dimensionality up to 500. We run the experiments with $d = 100$, 200, and 500, where d is the number of dimensions. The datasets consist of the Gaussians with randomly generated means and standard deviations. We also inject ten randomly generated outliers into the datasets. According to Figure 9, the ten injected outliers are correctly identified and the top 20 outliers are correctly identified in all the experiments. We notice that there is a slight decrease in the detection rate when d increases. When we examine the outliers manually, we find that it is due to the fact that the scores of the outliers become closer when d increases which makes the ranking fluctuate. This experiment shows that the randomized method is still viable in very high dimensions.

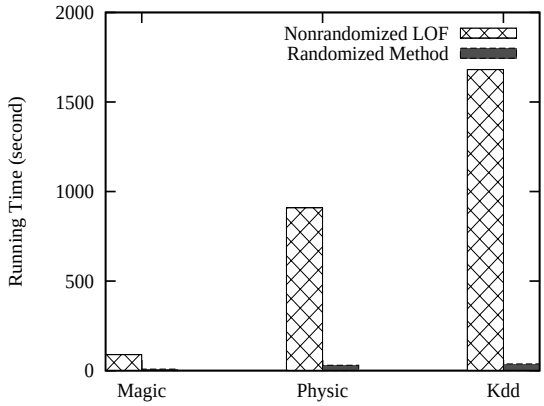


Figure 10: Running time: Randomization versus LOF

Table 4: Running time: Randomization versus LOF

Dataset	**Nonrandomized LOF (seconds)**	**Randomized LOF (seconds)**
Magic	89	8
Physics	909	30
Kdd Cup '99	1681	37
Synthetic	4848	106

3.6.4 Speed Comparison

We evaluate the running time of the randomized method against the nonrandomized version of LOF using the Magic, Physics, and Kdd Cup '99 datasets. In these datasets, Magic is the smallest (19K points) while Kdd is the largest (80K points). In Figure 10, the running time of the nonrandomized version grows quickly when the size of the datasets increase from 19K to 80K. However, the running time of the randomized method grows slower. In addition to Magic, Physics, and KDD Cup '99 datasets, we use a synthetic dataset with 200 dimensions and 100K points. The synthetic dataset contains five randomly generated Gaussians and ten random outliers. According to the experiment, the randomized method is consistently faster than the original LOF method. The running times of the methods, in seconds, are shown in Table 4.

3.6.5 Performance

We randomly generate the Gaussian clusters with different means and standard deviations for the sizes from 50K to 400K. We randomly injects the top 10 outliers in the datasets. We generate the datasets for $d = 50$, 100, and 200. According to the results, all the generated outliers are detected as the top outliers. Figure 11 shows the running time for different datasets. The vertical axis shows the running time in seconds. In the figure, the running time is linear with the size of the dataset for different dimensions. The experiments show that the algorithm can scale well with high dimensionality.

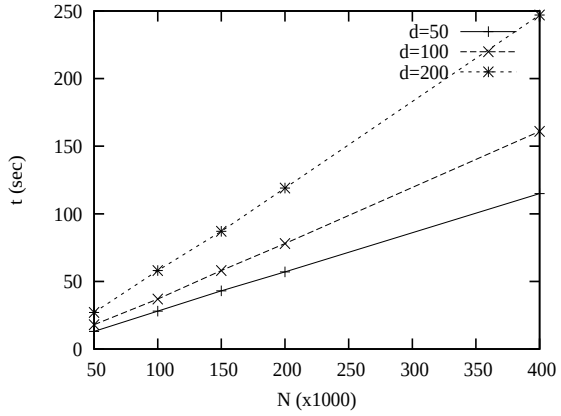


Figure 11: Running time of the randomized method

3.6.6 Convergence Rate

The method relies on the multiple iterations in order to rule out the false outliers. We will evaluate how the iterations affect the effectiveness of the method. We observe that in the first iteration there will be many false outliers. However, when the number of iterations (N_{iter}) increases, these outliers will be ruled out in subsequent iterations. The quality of detected outliers will become stable at some iteration. We will evaluate it based on the changes in the scores. This experiment aims to identify a good value of N_{iter} in practice.

Figure 12 shows the rate of change in the size of outliers for the Magic, Physics, and KDD dataset (after filtering out the low score outliers). As expected, the figure shows that the number of outliers changes rapidly in the first few iterations and the rate of change becomes stable when N_{iter} approaches 10. The rate of change is insignificant when $N_{iter} > 10$. We performed multiple runs with the datasets and found that in general, $N_{iter} = 10$ is a reasonable choice for the randomized algorithm.

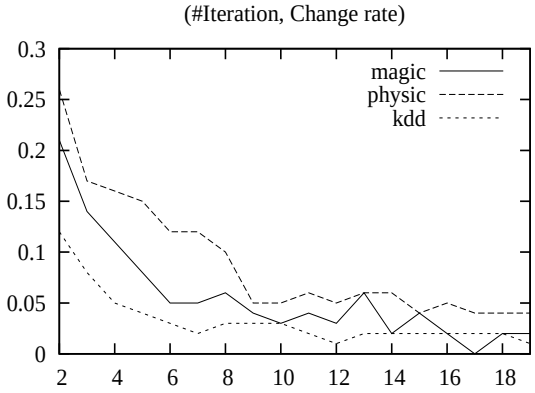


Figure 12: Convergence rate

3.6.7 Other Parameters

The parameter M_θ is the stop condition for the partition step. The partition will stop if there is less than M_θ points in the partition. As discussed earlier, the value for M_θ should not affect the quality of the algorithm as long as it is large enough. In this case, M_θ should be at least larger than $MinPts$. In our experiments, the scores are not affected when we increase M_θ.

In the algorithm, δ_{out} is used to limit the number of candidate outliers. In practice, δ_{out} can be set with any value that is smaller than the minimum outlier score we want to detect. The choice of δ_{out} does not affect the running time significantly since the merge operation is fast. The choice of δ_{out} depends on the datasets. For example, in the KDD dataset, we can set $\delta_{out} = 2$ because KDD returns many outliers with very

high scores, whereas we can set $\delta_{out} = 1.2$ and 1.5 for Magic and Physics datasets respectively since their scores are small.

3.7 Conclusion

We have shown that it is unnecessary to perform the k-nearest neighbor computation for the entire dataset in order to identify local density-based outliers. We introduced a randomized method to compute the local outlier scores very fast with high probability without finding k-nearest neighbors for all data points by exploiting the outlier consistency property of local outliers. We also introduced different versions for the randomized method to improve its accuracy and stability. The parameters can be selected intuitively. We have evaluated the performance of our method on a variety of real and synthetic datasets. The experiments have shown that the scores computed by the randomized method and the original LOF are similar. The experiments also confirm that the randomized method is fast and scalable for very high dimensional data. A natural extension of this method is to develop an incremental version of this method so that it can be adapted to real-time applications where the datasets are dynamically updated.

CHAPTER IV

ANOMALOUS PATTERN DETECTION USING ADAPTIVE NEAREST NEIGHBOR APPROACH

4.1 Motivation

Clustering algorithms divide the similar observations into groups in order to extract the common patterns of the data. In order to learn the general patterns, small clusters and non-dominant patterns are discarded or simply undetected . Despite their relatively small size, these clusters may be invaluable because their non-dominant patterns may reveal important knowledge. In this chapter, We introduce a new metric and a new algorithm that can discover small clusters in high dimensional and heterogeneous datasets. We have shown that our algorithm can effectively discover these clusters. In addition, our algorithm has discovered novel patterns based on our proposed metric of interestingness for unusual observations.

4.2 Adaptive Nearest Neighbors

Our approach is based on a variation of k-nearest neighbors and the concept of dual-neighbor to cluster the data set. In clustering, k-nearest neighbor algorithm is used to cluster the dataset by constructing a list of k-nearest neighbors for each point in the data set. The distance from a point to its k^{th} nearest neighbor is considered as its neighborhood distance. A point and its neighbors are considered to be similar to each other. The definition of similarity can be misleading since the close points may not be actually close to each other as illustrated in Figure 13. Point q belongs to a dense region while point p is in a less dense region. With $k = 5$, s is in the list of k-nearest neighbors of p and s is considered to be similar to p. However, as shown

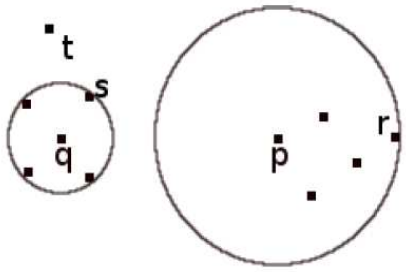


Figure 13: k^{th} nearest neighbor of p and q

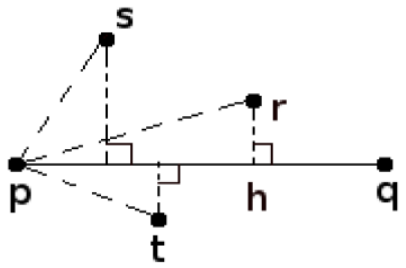


Figure 14: Adaptive nearest neighbors of p and q

in the figure, s is not similar to p because the distance between q and its nearest neighbors is less than that between q and p. Those two dissimilar points will be in the same cluster.

To solve the problem, Jarvis and Patrick introduced the concept of shared nearest neighbor [75]. The strength of the similarity between two points is measured by the number of nearest neighbors shared between them. Two points belong to the same cluster if the strength of the link exceeds a certain threshold. The clustering algorithm can produce excellent results. However, it is non-trivial to select an appropriate value of k and to justify the results of SNN in high dimensions. Ertoz et al improved the SNN by introducing the topic threshold [52]. A point with the number of strong links exceeding the topic threshold will represent its neighbors. Their clustering algorithm is based on the number of strong links and the link strength of the point in the data set. In high dimensions, the points in small clusters can not have a sufficient number of strong links to form a cluster. The points in this cluster will be broken into

smaller clusters even though they may be only slightly different from other points. Another problem is that the parameter k is the same for all points in the data set. As illustrated in Figure 13, the result will be inconsistent with a global parameter k. Figure 13 illustrates a simplified case when k is small. In the figure, the distance from p to its 4^{th} nearest neighbor is twice the distance from q to its 4^{th} nearest neighbor even though the distance from p and q to their 2^{nd} nearest neighbor are the same. The volumes of k-distances of p and q will differ significantly with a small increase in k.

In this paper, we propose the use of adaptive nearest neighbors (ANN) to define the neighborhood distance. The approach has three parameters which can be used to fine tune the adaptive neighborhood distance. From our perspective, the concept of neighborhood distance of a point, say p, is a relative concept since it can not be defined without surrounding points. As illustrated in 13. The neighborhood distance of p is greater than that of q because the first two nearest neighbors of p are farther than those of q.

With this observation, the first few nearest neighbors are used to define the initial neighborhood distance. Those neighbors are called the initial neighbors or i-neighbors in short. The distance from p to its i-neighbors is called i-distance. The i-distance defines the minimum neighborhood distance of p regardless of k. When p is in a dense cluster, the i-distance tends to be smaller.

The next parameter, α, is used to control the local variation of the neighborhood distance around p. In Figure 14, r, s and t are i-neighbors of p whereas q is the 4^{th} nearest neighbor of p. First, we project r, s and t on the line passing two points p and q. The line segment ph is chosen for it is the longest projected line segment of r,s and t on pq. If the ratio between the line segment pq and ph is less than α, then q is included in the list of neighbors of p. Point q is called an adaptive neighbor of p. The list of adaptive nearest neighbors of p is denoted by ANN(p). This process is repeated

until there is a point w in the k-nearest neighbor list of p whose ratio is greater than α. Point w and all the nearest neighbors of p farther than w are excluded from the adaptive nearest neighbor list of p. Point w is called the boundary point. The parameter α controls the local maximum variation of the nearest neighbors. The idea behind α is that the neighbor should be excluded from the list of nearest neighbors when it is significantly different from the others in the list and α measures the level of differences. The choice of α is application dependent. Intuitively, we can set the value of α with 1.2. The value of 1.2 says that if the change in distance is larger than 20%, then the change is significant and the neighbor corresponding to that change should not be in the list of adaptive neighbors.

The last parameter to adjust is the granularity level. For a small data set as in Figure 13, it makes sense to partition it into two distinct clusters. But in a larger data set, the two clusters should be merged into one if the distinction between them is small compared with others. The boundary points can be used for controlling the granularity. We use the parameter z for this. The procedure for constructing the lists of ANNs is modified as follows. Instead of stopping the construction of the ANN list for p when a boundary point is reached, we continue to put it into the ANN list of p. The process is stopped when z equals the number of times we reach the boundary points. The algorithm achieves the finest granularity level when $z = 1$. The detailed procedure for constructing the ANN list is described in Figure 15. In Figure 15, s is the number of i-neighbors and z is the granularity tuning parameter. Also, k is the maximum number of nearest neighbors that are computed for a point p. The value of k should be less than the largest size of the anomalous patterns in which we are interested. Otherwise, the anomalous pattern will be combined with normal patterns for some large value of z.

With adaptive nearest neighbors, we can define the neighborhood distance of a point independent of k with different levels of granularity. This neighborhood

distance is called the adaptive neighbor distance, denoted by adistance. According to the discussion above, we can say that any point within the adaptive neighborhood of a point p is truly a natural neighbor of p. Also, we observe that the similarity must be a mutual relation. In other words, if two points are considered naturally close to each other, they should be in the list of ANNs of each other. We formally define the closeness as follows:

Definition 7. *Given any two points p and q in dataset D, p and q have a dual-neighbor relationship, denoted by dual(p, q) ≡ true, if and only if p ∈ ANN(q) and q ∈ ANN(p).*

In the definition, two points are considered neighbors to each other when they have a dual-neighbor relationship. With this definition, we can address the problem of k-nearest neighbors as illustrated in Figure 13 when p and q have no dual-neighbor relationship where $z = 1$. In a coarser granularity level, i.e. $z = 2$, p and q become neighbors to each other. Another useful concept is the indirect dual-neighbor relationship.

Definition 8. *Given any two points p and q in dataset D, p and q have an indirect dual-neighbor relationship, denoted by indual(p, q) ≡ true, if and only if*

(i) dual(p, q) ≡ true, or (ii) ∃r ∈ D : dual(p, r) ≡ true ∧ indual(r, q) ≡ true

As discussed above, we are interested in discovering unusual patterns. With the definition of the indirect dual-neighbor, we can formally define the concepts of usual and unusual patterns as follows:

Definition 9. *An observation is usual if it belongs to a large group of observations where each observation has at least one observation in the group that is similar to it. The group that contains those usual observations is called a usual pattern.*

Definition 10. *An observation is said to have an unusual pattern if it belongs to a small group of observations where each observation has at least one observation in the group that is similar to it and has no other observation outside of the group similar to it. The group that contains those unusual observations is called an unusual pattern.*

In this chapter, ANN and the dual-neighbor are used to define the similarity between two points. The indirect dual-neighbor shows the indirect similarity between two observations belonging to the same pattern. With the definitions of usual and unusual patterns, the clustering criteria of our approach is stated as follows:

Given two points p, q in dataset D, p and q belong to the same cluster C if and only if $indual(p, q) \equiv true$.

This definition implies the chain effect and it can produce very large clusters. This, however, is acceptable because the observations in large clusters are usual. As mentioned above, we are interested in discovering unusual patterns. To be unusual, the observations should deviate from other usual patterns. Therefore, the chain effect will have no impact on the results for discovering unusual patterns. The parameters i-neighbors, α and z play an important role in defining the level of similarity. In a uniformly distributed region, the choice of the number of i-neighbors has less effect since all points should belong to the same cluster. The concept of i-neighbor is useful in non-uniformly distributed regions. In this case, the number of i-neighbors should be small, which is usually less than 10. The parameter α is used to control the local variance of the neighborhood distance according to the i-neighbors. The parameter α defines the upper bound of the acceptable deviation between the neighbors. The last parameter is z which is used for adjusting the level of granularity. When $z = 1$, we can see all natural clusters in terms of ANN. When z is increased, the nearby clusters are merged together. In practice, the number of i-neighbors and α are less important than z since they can be easily selected without affecting the results. Intuitively, we can set $\alpha = 1.2$ and $z \in [2, 4]$.

```
1: function ANN(p)
2:     for i ← s, k do
3:         r ← i^th neighbor(p)
4:         τ_max ← 0
5:         π_max ← 0
6:         for j ← 1, (i − 1) do
7:             q ← j^th neighbor(p)
8:             π ← pq⃗ pr⃗ / ‖pr⃗‖
9:             τ ← ‖pr⃗‖ / π
10:            if π > π_max then
11:                π_max ← π
12:                τ_max ← τ
13:                idx ← j
14:            end if
15:        end for
16:        if π_max > α ∥ π_max = 0 then
17:            if level < z then
18:                level ← level + 1
19:            else
20:                return all j^th neighbor(p), where j ≤ idx
21:            end if
22:        end if
23:    end for
24: end function
```

Figure 15: Constructing adaptive nearest neighbors

```
1: procedure CLUSTER(HashSet D)
2:     Stack S
3:     Vector clsSet
4:     HashSet C
5:     while D ≠ ∅ do
6:         p ← remove D
7:         push p into S
8:         C ← new HashSet
9:         add C into clsSet
10:        while S ≠ ∅ do
11:            q ← pop S
12:            add q into C
13:            for all r ∈ ANN(q) ∧ dual(q, r) = 0 do
14:                push r ← S
15:                remove r from D
16:            end for
17:        end while
18:    end while
19: end procedure
```

Figure 16: Outcast pseudocode

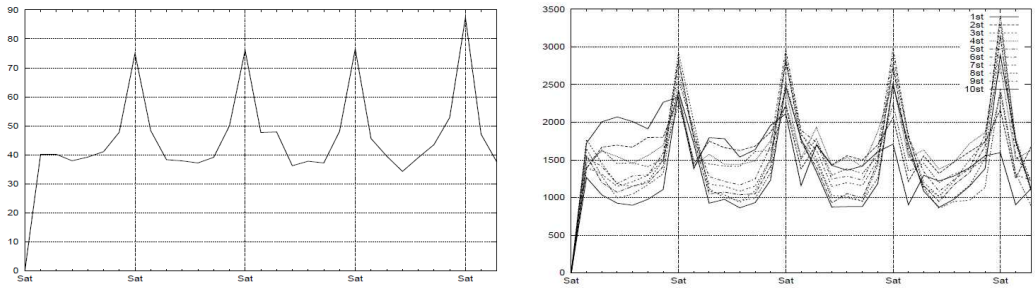
Figure 15 shows the linear time processing steps to cluster the data set after the lists of adaptive nearest neighbors have been computed according to Figure 16. For every unclustered point, we randomly select a point to form a new cluster where the selected point is the representative of the cluster. Then, we expand the cluster by including all the dual-neighbors and the indirect dual-neighbors of the point into the cluster. To facilitate the algorithm, we create a stack S to store the dual-neighbors. As shown in steps 11-12, an unclustered point p is removed from the data set. Since p does not belong to any cluster, a new cluster C is created for p before pushing p onto stack S. In steps 13-16, a point q is popped from S and q is added to cluster C. Besides, all dual-neighbors of q are pushed onto the stack. Those steps are repeated until S is empty, which means the inner while loop is stopped when indirect dual-neighbors of the points in cluster C are included in the cluster.

4.3 Experiments

In this section, we present experimental results using the Sam's Club data set [139]. The data set contains the sales transaction data for 18 Sams club stores between the dates of January 1 and January 31, 2000. From the sales transactions, we create a new data set of 34,250 tuples with 31 attributes. Each tuple represents a sale item and the attributes represent the total sales quantities for each individual item for the days in January. The total sale varies from 0 to 16,788. The purpose of this experiment is to apply the well-known local outlier detection method LOF and the density-based clustering algorithm SNN on the data set in order to detect any unusual sales patterns. We first ran LOF on the data set to determine the top local outliers. We then ran kmean and SNN on the top 5% outliers to produce a summary of the outliers. We also ran SNN on the whole data set with different values of k in the attempt to discover unusual patterns by studying the small clusters returned by SNN. We then compared the results with those from our algorithm.

4.3.1 LOF, KMEAN, SNN

Figure 17a shows the average sales for each day in January for all items in the dataset. According to the figure, the sale follows the same pattern every week. Sales gradually decrease from the start of the week toward the middle of the week and then slightly



(a) For all items in the dataset (b) For top 10 LOF items

Figure 17: The average daily sales in January for all items

Table 5: Top 10 LOF outliers in Sam's club data

Item	**LOF**	**Item**	**LOF**
1	2681.28	6	1798.9
2	2205.68	7	1789.0
3	1907.18	8	1710.38
4	1895.92	9	1699.56
5	1841.24	10	1686.28

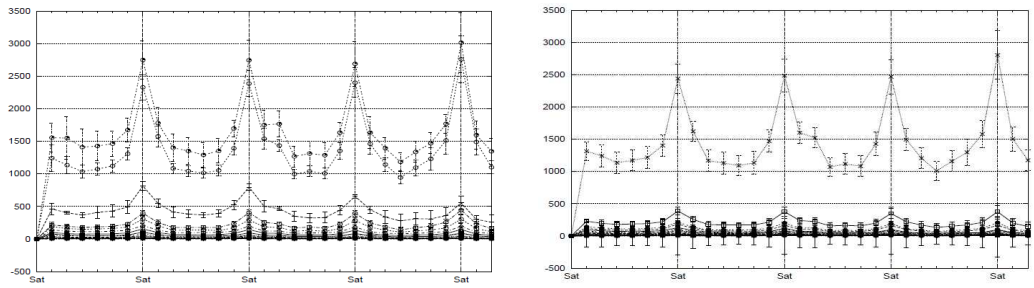
Table 6: Clusters in Sam's club data generated by SNN

k	**cluster**	**size**	$\overline{\mu}$	$\overline{\sigma}$
80	86	30	73.75	195.02
	1571	41	101.07	267.16
110	85	33	122.96	300.23
	1522	82	87.76	213.85
140	85	33	122.96	300.23
	1561	112	74.66	213.85
170	14	32	90.83	267.01
	1600	155	78.66	207.4
200	1668	185	89.16	208.46

increase toward the end of the week before achieving its peak on Saturday. The sales quickly drop on Sunday. This pattern repeats every week in January. The figure illustrates that most customers tend to go shopping on Saturdays.

For the first test, we computed the LOF values for all items. The LOF values vary greatly from 0.012 to 2681.28. The values of the top 10 outliers and their sale information are shown in Table 5 and Figure 17b. The strongest outlier is item 1 whose pattern deviates from the norm since its sales increase slightly on Saturdays and tends to fall toward the end of the month. For the next 9 outliers ranked by the LOF approach, the sale pattern resembles that in Figure 17a.

We take the top 5% of the items ranked by LOF to form a new dataset with the size of 1712 and then apply several clustering algorithms on the new data set. The purpose is to group the top outliers together in order to learn the common patterns of these outliers in an attempt to explain their significance. In this experiment, we use kmean and SNN to cluster the dataset.



(a) For all items in the dataset (b) For top 10 LOF items

Figure 18: The average daily sale for all items and top outliers

Figure 18a shows the average sales amount and its standard deviation for items in the clusters produced by kmean when $k = 20$. According to Figure 18a, kmean clusters the outliers into groups with different ranges of sale volume (less than 500, 500, 1000 and 1500) and the average size of the clusters is 85.6. The sale patterns for those clusters are the same as the common pattern of the whole data set. Similar results are obtained when we ran kmean with different values of k.

Figure 18b shows the results of SNN when $k = 20$. There are 32 clusters with the average size of 53.5. Clusters 1 and 17 are two main clusters with the size of 791 and 100 respectively. The average sale of cluster 1 ranges from 32.3 to 89.3 and its standard deviation ranges from 167 to 419.4. The sales volume of the items in the cluster are quite different even though they belong to the same cluster. As illustrated, the sales pattern of the clusters resembles the common sales pattern of the whole data set.

In the next experiment, we ran SNN on the whole data set, varying k from 20 to 200. Table 6 shows the list of clusters with the size greater than 30 for each k. In Table 6, μ is the average sales for each cluster and σ is the average standard deviation of the sales for the dates in January. We found that most items form a cluster by themselves and that there are at most two clusters with the size greater than 30 for each k. Also, the fourth and fifth columns of Table 6 show that σ is twice μ. It means

Table 7: Interesting patterns clustered by Outcast

Cluster	Size	Cluster	Size
93	70	652	40
363	54	663	40
241	49	444	209

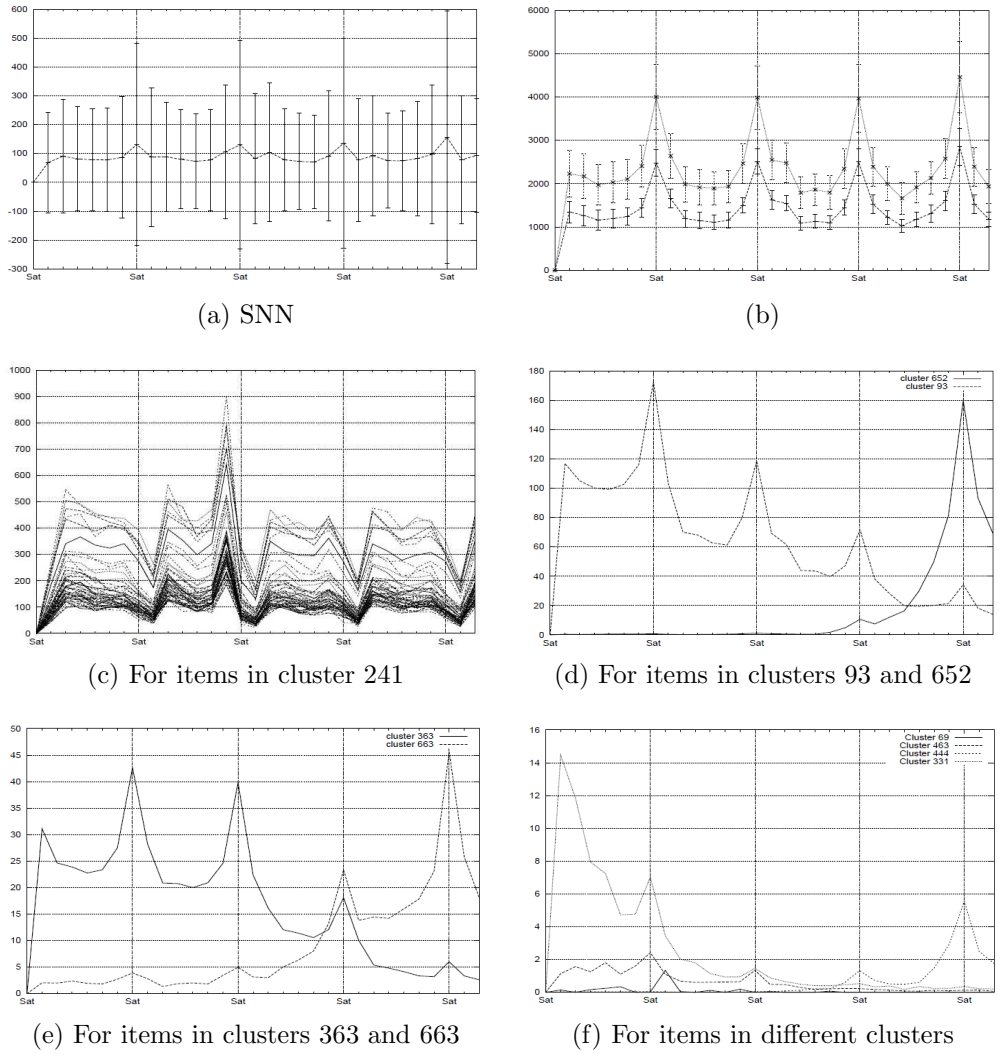
that the sale quantity varies greatly for the items in the same clusters as shown in Figure 19a. Consequently, we found no interesting patterns in this experiment.

4.3.2 Outcast

Table 7 shows the size of the interesting clusters found by our algorithm with the granularity level 2. There is one major cluster with the size of 6203 and 16 small clusters with their size ranging from 40 to 209. Among them, cluster 1 and 110 (Figure 19b) have sale patterns that resemble the common pattern. We found that the top 14 outliers recognized by LOF belong to cluster 1 and that 51 out of 58 items in cluster 1 are in the top 100 outliers.

Cluster 241 with the size of 49 is the most interesting pattern found by our algorithm. Figure 19c shows the sales pattern of the items in the cluster. Even though the average sale volumes of the items vary from 80.74 to 389.35, they follow the same pattern which is reversed from the common sale pattern (Figure 17a). The sale achieves the peak at the beginning of the week instead of on Saturday and then slightly decreases toward the weekend before reaching its lowest point on Sunday. It is interesting to find that all the sales in the second week of the items in this cluster jump sharply on Friday instead of Saturday as the common pattern and then the sale drops quickly on Saturday and Sunday. The sales on this day is almost double the sales on the peaks of the other weeks. When we further investigate the items in the clusters, we found that all of those items are cigarettes. Even though the items have interesting sales patterns, their LOF ranking is very low.

Two other interesting patterns occur in clusters 93 and 652 as shown in Figure



(a) SNN (b)

(c) For items in cluster 241 (d) For items in clusters 93 and 652

(e) For items in clusters 363 and 663 (f) For items in different clusters

Figure 19: The average sale volumne for each day in January

19d. Even though cluster 93 resembles the weekly common sales pattern in the way that the sale is highest on Saturday as compared with the other days in the same week, the overall sales in every week tends to decrease toward the end of the month. In contrast, items in cluster 652 almost have no sale for the first three weeks. In the last week, the sales increase rapidly toward the end of the month and achieve their peak on the last Saturday of the month. Figure 19e shows the sale pattern for clusters 363 and 663. Those two clusters are similar to clusters 93 and 652 except that the sales for those clusters are four times less than that of clusters 93 and 652.

Figure 19f shows the sale patterns for clusters 60, 463, 444 and 331. Cluster 444 contains 209 items and those items have almost no sales except for a few sales on the last Saturday. The other clusters are less interesting than the ones mentioned above due to their small sale volume.

In summary, we ran experiments with different combinations of the outlier detection and clustering algorithms. With LOF, most items in the data set were classified as outliers. When examining the top 10 outliers. We found that the sales pattern of the top outliers is slightly different from the common weekly sales pattern. The top 10 outliers have high sale volumes and their sales pattern follow the weekly pattern. We then clustered the dataset with kmean and SNN. Those clustering algorithms divide the top 5% of the outliers into groups of different sale volumes but no interesting patterns are found. It is the same when we ran SNN on the whole data set. However, when we tested the data set with our algorithm, we discover six unusual patterns. Among them, the sale pattern of cluster 1 does not differ from the weekly sales pattern. We found that 89% of the items in the cluster are in the top 100 outliers ranked by LOF. Cluster 241 is the most interesting since we found that cigarrete sales follow the Friday sales pattern rather than the Saturday pattern. The other four clusters do not follow the common sales pattern. The experiment confirms that interesting patterns may not be discovered by simply clustering the top outliers.

4.4 Conclusion

Clustering and outlier detection are two different approaches that can be used to learn general patterns and novel events. However, both of these approaches can not detect unusual patterns that appear in small clusters, which may be interesting. For most clustering algorithms, small size clusters are sacrificed in order to discover large size clusters. In contrast, the outlier detection approach simply focuses on single outliers rather than groups of outliers. Top outliers are the most interesting events. In our

experiments, we have shown that top outliers are not always interesting since they may simply be noise in high dimensions, all data points may be considered outliers due to the sparsity of the data. We present an alternative approach for knowledge learning by introducing the concept of an unusual pattern, based on the size of the small clusters and their deviation from the common patterns. We have developed an algorithm to detect those unusual patterns. The running time of the algorithm is quadratic. The parameters of the algorithm are used to adjust the granularity level of the output. Our experiments on a real world data set show that our algorithm can discover interesting unusual patterns which are undetected by two well-known oultier detection and clustering techniques , namely LOF and SNN, and their combination.

CHAPTER V

ACCURATELY RANKING OUTLIERS WITH DELTA DEVIATION ACCUMULATION AND MIXTURE OF VARIANCES

5.1 Motivation

In Chapter 3, we discussed a randomization method to detect local density-based outliers efficiently. In this chapter, we will introduce a method to improve the accuracy of the local density-based outlier detection method. As we have discussed, the distance-based [51] and density-based [24] approaches were introduced in order to overcome the limitations of the statistic approaches. With regard to the distance-based approach, a point whose distance to the remaining points is large is considered to be an outlier [51]. For the density-based outlier, the degree of being an outlier of an observation is measured by the ratio of the k-nearest neighbor distance between the observation and its local density. The main advantage of this approach over the statistic ones is that the outliers can be detected without the knowledge of the distribution of the dataset. It is effective in datasets with a small number of attributes. In practice, the dataset may contain many attributes. In certain types of datasets, e.g. image data, the number of attributes may be in the thousands. Feature selection methods can be used to reduce the number of attributes, for an example in clustering and classification. In some cases, the number of reduced attributes is significant. The attributes are selected or transformed based on the objective functions of the clustering/classification techniques. The objective is to keep the attributes that are most relevant to the clustering/classification criteria. The attributes from the dataset that do not show a clear structure for clusters or do not correlate with the class labels are

not selected. However, for the problem of outlier detection, we do not know in general which attributes will play an important role in determining outliers. An observation may show up as a strong outlier with respect to the attributes that are eliminated by the feature selection method. If we dismiss any attribute, we may not be able to discover the outliers as shown by Breunig et al [24]. Therefore, in some cases, we need to run the algorithm on the entire feature space to detect outliers. This, however, poses problems for the distance-based and density-based outlier detection methods in high dimensional data. These problems which arise because of mixture of variances and accumulated sub-dimensional variations will be discussed in the following sections.

5.1.1 Mixture of Variances

We consider a dataset with seven data points to illustrate the problem of using k-nearest neighbors (L2) to detect outliers. The data has three attributes x, y and z. The domains of x and y are the intervals of [0, 2]. The domain of z is the interval [0, 8]. Figure 20 shows a two dimensional plot for the data points in attributes x and y. We then compute the nearest neighbor distance for these points. Except for point p, the nearest distances are less than 0.64. The nearest neighbor distance of p is 1.39. From those values, we see that p has an exceptionally large nearest neighbor distance compared with the other points. Figure 20 shows that p clearly deviates from the other six points in the plot. Thus, we conclude that point p is an outlier.

Figure 21 shows the complete plot for the data points using all three attributes x, y and z. The variation in the attribute z is four times those of attributes x and y. In the complete space, the nearest neighbor distance of p is the same as those of the other points. The variation in attribute z suppresses the variations in attribute x and y. For the 3-dimensional space, p does not appear as an outlier. This is incorrect if we look at the subspace using the dimensions x and y.

This example shows the problem of using the distance to detect outliers. A possible

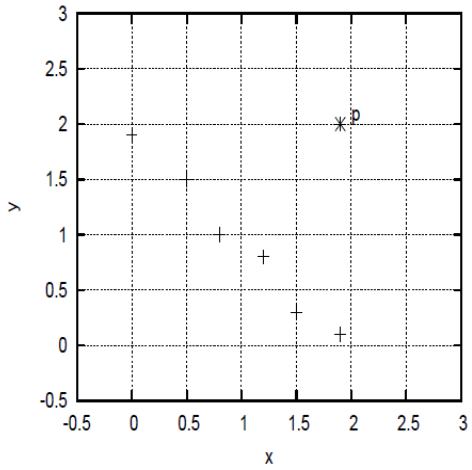


Figure 20: The 2D outlier p

solution is to consider the attribute variations to compute the distance between the points by normalizing the values. The normalization, however, is globally based. Thus, it is ineffective for regions with local variations. We can consider the following situation. The data points from the example above belong to a subset of a dataset. The ranges of x, y, and z above are subsets of the domains that are already normalized. If the points are also nearest neighbors of each other in the larger dataset, the problem still remains. Therefore, the normalization will not tackle the problem.

We can formulate the problem with an arbitrary number of attributes as follows. Let us say we have a point q which is an outlier in a subspace. Let $\{\sigma_i\}$ be the variances of the attributes in this subspace. The variances can be computed either from the local region of point q or from the entire dataset. The method of computing the variations corresponds to the problem of local outlier and global outlier detection respectively. Suppose we have an attribute j with the variance of σ_j where $\sigma_j = k_i \times \sigma_i$ and k_i are some large numbers. Point q is no longer an outlier in the new subspace that contains attribute j.

Instead of considering every attribute, one possible solution is to compute the outlier scores for the data points for each possible combination of attributes. If a

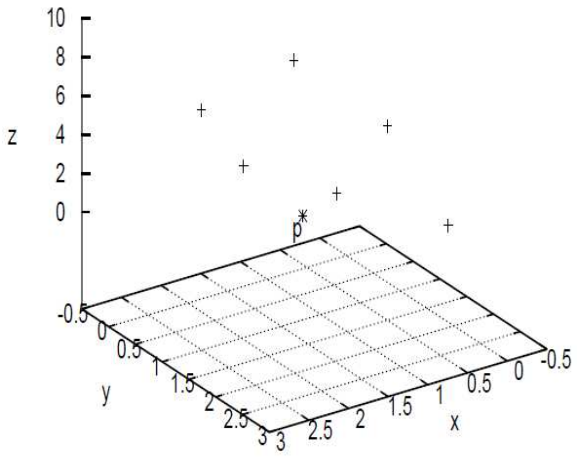


Figure 21: The outlier p is suppressed in the 3D space

point is an outlier in any combination of attributes, it is considered to be an outlier. However, the number of possible combinations is exponential. Therefore, it can not be done for high dimensional data.

5.1.2 Accumulated Sub-dimensional Variations

In this section, we consider another problem with respect to outlier detection in high dimensions. Let us consider three points p, q, and r in an n-dimensional space. In this example, p and q are normal points. Point r is an outlier in an m-dimensional subspace. Let us denote the value of attribute i of a point p with subscript i (i.e., p_i) and let us assume that the difference between p_i and q_i is δ_n for every attribute $i \in [1, n]$, we have

$$d\,(p, q) = \sqrt{\sum_{i}^{n} \delta_n^2} \tag{1}$$

We further assume that $|p_i - r_i| = \delta_m$ for $\forall i \in [1, m]$ and $|p_i - r_i| = 0$ for $\forall i \in [m+1, n]$. We have

$$d\,(p, r) = \sqrt{\sum_{i}^{m} \delta_m^2} = \delta_m \sqrt{m} \tag{2}$$

Equations 1 and 2 show the distance from p to q and r in terms of δ_n and δ_m. If two distances equal ($d(p,r) = d(p,q)$), we have

$$\delta_n\sqrt{n} = \delta_m\sqrt{m} \Rightarrow \frac{\delta_m}{\delta_n} = \sqrt{\frac{n}{m}} \quad \text{,where} \quad \delta_n, \delta_m \neq 0 \tag{3}$$

Let define $\tau = \frac{\delta_m}{\delta_n}$. We obtain the following expression:

$$\tau = \sqrt{\frac{n}{m}} \tag{4}$$

In Equation 4, τ is the ratio of the distance of p to q and r in one attribute in the m-dimensional space. It measures the deviation of r with respect to p and q in this attribute. If τ is large, we can conclude that r is an outlier in such cases. However, in n-dimensional space, the distances from r to p and q are the same. It is because of the accumulation of insignificant statistical changes δ_n. The accumulation is large enough to suppress the strong deviation of r in the m-dimensional subspace. Therefore, r is not an outlier. This shows that an outlier in a subspace can be suppressed by random deviations in a larger super-space.

Equation 4 implies that the ratio of the nearest neighbor distance between two points can be as large as $\sqrt{n/m}$ so that an outlier in an m-dimensional space will not be an outlier in n-dimensional space. For an example, with $n = 100$ and $m = 4$, we will have $\tau = \sqrt{100/4} = \sqrt{25} = 5$. Hence, outliers which have a ratio of 5 :1 or less of the distance of their nearest normal group of points to the density of the group may not be detected. The number of 5-dimensional subspaces is large (approximately 7.5×10^7). Similar to the mixture of local variances, we can not compute the outliers in all possible subspaces.

The problem of not being able to distinguish if a possible outlier is a true outlier or a normal point in this example is the problem of accumulated sub-dimensional delta variations.

Table 8: Notations and basic definitions

$KNN(p)$	The set of k-nearest neighbors of p
p_i	The value in attribute i of point p
$d_i(p,q)$	The difference of p and q in attribute i
$L_i(p)$	Ordered list of points $q \in KNN(p)$ ordered by $d_i(p,q)$
ξ_i^j	$\frac{d_i^j(p,q^j)-d_i^j(p,q^{j-1})}{d_i^j(p,q^{j-1})}$

5.2 Related Work

Feature selection is a method for selecting a subset of attributes or features from the full feature space that preserve the characteristics of the dataset the most. The effect of feature selection is to reduce the number of attributes in data analysis to avoid the curse of dimensionality and to improve the quality of data analysis. Feature selection is used in clustering and classification. The characteristics of a dataset depends on the objective function of a clustering or classification method.

Information entropy is a common measure used in selecting the best attributes for classification. In information theory, the information entropy of an event X,$(H(X))$, measures the amount of information carried by the event X. The mutual information, an extension of entropy, measures the correlation between the two events X and Y, $(I(X,Y))$. It measures how much one can infer X given that Y is known or vice versa. When information entropy is applied to the problem of classification, an event corresponds to an attribute of the dataset. The class labels also correspond to one event. The mutual information measures the correlation between the class labels and the attributes. In feature selection, the attributes with top mutual information are correlated to the class labels the most, therefore they will be preserved.

In clustering, principal component analysis or PCA is often used for feature selection. PCA is a method that computes the eigenvectors and eigenvalues of a sample dataset. A subset of eigenvectors define a subspace. The corresponding eigenvalues

measure the variances of the dataset in that subspace. They represent the most information of a dataset. Therefore, the eigenvectors with the top eigenvalues are selected for clustering.

Even though feature selection is useful in clustering/classification, it cannot be used to detect the outliers in our case. The feature selection methods select the attributes that are most relevant to the majority of the data. However, it does not necessarily indicate the attributes that an observation will likely deviate with respect to the other data points. Therefore, feature selection is not applicable to the problem of detecting these outliers.

Another related work is subspace clustering. This method tries to discover clusters in the subspaces that are hidden in the data spaces. A popular method is to measure the variances in densities in some subspace to detect these clusters. Similar to feature selection, subspace clustering focuses on the concentration of the cluster density, not the outliers; therefore, it is not relevant to outlier detection.

Understanding the limitation of subspace outlier detection, Lazarevic et al [95] proposed the feature bagging method. The method randomly selects subsets of attributes and detects outliers in those subspaces. The detected outliers will be aggregated together. The method, however, only selects some subspaces randomly. It may miss the subspaces where a point will be an outlier. Since the number of possible subspaces is very large in high dimensions, the probability that a subspace will be missed is high. If a point is an outlier in only one subspace, it is likely that it can not be detected. In addition, feature bagging does not solve the problem of mixtures of local variances as discussed above.

In the following section, we will discuss our method to tackle these challenges.

5.3 Concept of Outlier in High Dimensions

Even though the concept of outliers based on distance is clear in low dimensions, the distance metric is not intuitive in high dimensions. In this section, we will provide an intuition for what it means to be an outlier in high dimensions and will give precise definitions of outlier score functions.

In previous work, the distance between a point and its neighbors, which is used to define the degree of an outlier for a point, is based on the Euclidean distance. It is self explanatory in low dimensions. However, as our related discussion in section 5.1 shows, outliers can not be detected accurately using the Euclidean distance. Therefore, we propose Chebyshev distance (L_∞) in place of Euclidean distance (L_2). By definition, the Chebyshev distance between any two points p and q is the maximum of $|p_i - r_i|, \forall i \in [1, n]$. We argue that the problem of delta accumulation can be avoided since the variances are not cumulative in high dimensions in L_∞.

Let us say we have a sample S such that each attribute of the points in S follows the distribution $N(\mu_i, \sigma), \ \forall i = 1 \ldots n$. With L_2 norm, the distance between two points can vary from 0 to $2\sigma\sqrt{n}$. The variance is proportional to the number of dimensions. However, in Chebyshev space, the range of the difference will be limited to the interval $[0, 2\sigma]$ regardless of n.

We will first assume there is a boundary of a point p such that any point that is outside the boundary is an outlier. A boundary can be a squared rectangle R (or hypercube in high dimensions) with p as its center. The sides of the rectangle are parallel with the axis. R define a region in which every point is considered to be a neighbor of p. We say that point q is an outlier with respect to p in region R with length $2d$ (the distance between any two parallel sides) if its distance to R is significantly larger than the bounds, denoted by $\|p - R\| \gg d$.

To be more precise, an outlier is defined in the following postulate:

Postulate 1. *Given a boundary hypercube R with length $2d$ for a point p, A point q*

is an outlier with respect to point p if distance $(p, R) > \kappa d$ *for some large* κ.

Postulate 1 says that every point will have a boundary defining the neighborhood region. At this moment, we assume that the region is already defined. The method to construct the boundary will be discussed later in the next section. The definition of outlier and the use of the Chebyshev distance lead to the following theorem:

Theorem 4. *A point q is an outlier with respect to p in region R with length 2d in n-dimensional space iff q is an outlier with respect to p in at least one dimension i, where* $i \in [1, n]$.

Proof. The projected hypercube on a dimension i is a line segment D_i where p is its center. Since the length of the hypercube is $2d$, the length of the line segment is $2d$. Since q is an outlier with respect to p, we have $distance\,(p, R) > \kappa d$. As defined, the distance from a point to a hypercube is the maximum distance from the point to the surfaces of the hypercube in the Chebyshev space. Since the surfaces are orthogonal or parallel to the line segment, $\exists i : \ distance\,(p_i, D_i) > \kappa d$. Thus, p is an outlier in at least one dimension i. Conversely, if q is an outlier with respect to p in at least one dimension i, we have $distance\,(p, R) > \kappa d$ by the Chebyshev distance definition. Therefore, q is an outlier with respect to p in the n-dimensional space. □

The Chebyshev distance and encapsulating hypercube provide the property to exam the outliers in each dimension individually. This property is later used to refine outlier scores to compute the outliers more accurately.

Postulate 1 defines what it means to be an outlier with respect to an individual point and we can extend the concept of outlier with respect to a set of points S.

Postulate 2. *Given a set of points S, if a point q is an outlier with respect to all points p in S for some rectangle R of p, then q is an outlier in S.*

The postulate is straightforward. We can see that given a set of points S, if p is an outlier to all points in S, then we can consider p to be an outlier with respect to set

S. However, if p is outlier only to a few points in S, we can not conclude that p is an outlier with respect to S. The criterion is self explanatory when the boundaries for the points in S are defined. We will discuss how the boundaries can be constructed and how the problems of data with a mixture of variances can be handled using this concept of outlier.

5.3.1 Filtering and Normalization

From Theorem 4, we observe that we can compute the outlier score in each attribute instead of computing the outlier in individual attributes. We will take a set of points from the local region of a query point. We then measure the deviation of the points in each attribute using Chebyshev distance with respect to the boundaries of each point in the local set. If in some attributes, the deviation of the point is not statistically significant, we will discard these attributes. Then, we can aggregate the deviations in the remaining attributes into a final score. The filtering allows us to deal with the effect of accumulating insignificant fluctuations in many attributes.

From the problem of mixtures of variances in Figures 20 and 21, we observe that the differences in the local variances suppress the outliers. The dimensions with high local variances will dominate those with low local variances. In our study, we are interested in unsupervised outlier detection. We do not have training datasets to learn the attributes that are more relevant. Therefore, it is conservative to treat the attributes equally. In addition, the rate of deviation is more important than the unit of deviation. These suggest that we could compute the ratio of the deviation against the variances in each dimension of a point instead of computing the ratio based on all attributes. The ratio based on an individual attribute measures the deviation of the point normalized with the local variance in the attribute. The local variances will be considered when we aggregate the deviations in all dimension. Therefore, it solves the problem associated with a mixture of variances discussed in the previous

sections. In the following section, we will discuss how to compute the outlier ratio for each dimension.

5.4 Sub-dimensional Outlier

We use $k^{th}nn(p)$ to denote the k^{th} nearest neighbor of p in Chebyshev space and $kdist(p)$ (k-distance) is the distance from p to its k^{th} nearest neighbor. In this chapter, where there is no confusion, the distance will be computed in Chebyshev space. The k-distance measures the relative density of the points in a dataset. Next, we want to compute the density of a point p projected into each dimension which we call dimensional density. The densities are used to construct the boundary hypercube for the point. A simple approach to compute the dimensional densities is to average the local distances from a point p to its neighbors for the dimension under consideration. However, the result depends on parameter k. It raises the question of how many neighbors should we consider in computing the dimensional densities? With small k, the dimensional density reflects the average pairwise distances between the points in the local region. However, the pairwise distances can vary greatly in nonuniform data. For an example, in a dataset where the points tend to appear in groups of less than k points (where k is very small), the k-distance will not measure the density correctly. In contrast, the dimensional density will be more biased with large k.

In Chapter 4, we introduced the definition of adaptive nearest neighbors which allows us to determine the natural dimensional density in terms of the level of granularity at each point. The method will automatically determine k using more intuitive parameters. In this method, we observe that when a point belongs to a nearly uniformly distributed region, the pairwise distance can approximately represent the local density of the point. Therefore, k should be small.

To measure the uniformity, we look at the neighbors with significant changes in the k-distance with respect to p. We say that the neighbors associated with the changes

are in a decision boundary of p. At each decision boundary, we will determine whether we should include more points to measure the density. The level of granularity will be used in such cases. Thus, we will use the concept of adaptive neighbor to define local dimensional density.

Denote that $d_i(p, q)$ the distance between p and q in dimension i, we will create an ordered list L_i of the nearest neighbors of p ordered by $d_i(p, q)$ (where q is a k-nearest neighbor of p) for every dimension i. For each neighbor q, if $d_i(p, q) = 0$, p should be eliminated from the list. To simplify the problem, in this work, we assume that there is no q such that $d_i(p, q) = 0$.

Let say we have $L_i \equiv \{q^1, \ldots, q^k\}$, where q^j denotes the projected j^{th} nearest neighbor of p on dimension i. We then compute the change of distances for the points in the list. The change can be computed using the differences $d_i(q^j, p)$ and $d_i(q^{j+1}, p)$. To normalize the change due to local density variation, we compute the rate of change by normalizing it with the distance between p and its nearer neighbor in the L_i. We define the ratio ξ_i^j which is $\frac{d_i^j(p,q^j) - d_i^j(p,q^{j-1})}{d_i^j(p,q^{j-1})}$ (for each $j \in [2, \ldots, k]$).

If p is in a uniformly distributed region, ξ_i^j will increase with j without any disruptive change. In such cases, we can use $d_i(p, q^1)$ to estimate the local dimensional density of p in dimension i. A point where there is a sharp increase in ξ_i^j is called the decision boundary of the local distance of point p. We will consider a change to be significant when it is greater than a threshold λ to , i.e. $\xi_i^j \geq \lambda$. The decision boundaries correspond to the levels of granularity. The levels of granularity determine how many significant changes should be allowed to estimate the density.

We use a parameter z to determine the level of granularity in detecting the outliers. We then define the local dimensional density of a point p with a granularity of level z as follows:

Definition 11. *Given that q^{j_z} is the z^{th} decision boundary point of a point p, the*

local dimensional density of p with the granularity level z in dimension i is

$$\gamma_i(p) = \begin{cases} d_i(p, q^1), & \xi_i^l < \lambda \text{ or } z = 1, \quad \forall l \in [1, \ldots, k] \\ d_i(p, q^{j_z}), & otherwise \end{cases} \tag{5}$$

In the definition above, if there is no sharp change in the neighbor distance in dimension i, i.e., the rate of change is less than a chosen threshold for every neighbor ($\xi_i^l < \lambda$, $\forall l \in [1, \ldots, k]$), we will choose the distance to the first neighbor in the list (L_i) as an estimate of the local density in the dimension i of p. However, if there is a sharp change, it may indicate that the neighbors prior to the decision boundary are too close to p with respect to the other points in the local region. We assume that the number of close neighbors is small. In such cases, we need to decide whether the distance corresponding to a change should be chosen in order to reflect the density more correctly. The value of z corresponds to the maximum number of changes that we should tolerate. Therefore, we will use the distance to the z^{th} decision boundary to estimate the density. Another case we need to consider is when the maximum number of decision boundaries in the current list of neighbors is less than z. One option is to extend the list until the z^{th} decision boundary is met. However, it is possible that the list will become too large in order to embrace this boundary. Therefore, we will set the limit at the k^{th} nearest neighbor. We will choose the distance to the k^{th} nearest neighbor to estimate the density if there is no z^{th} decision boundary.

Next, we compute the average local distance in each dimension for a local region S. Region S is a set of points in a local region of the dataset. With $|S|$ large enough, Formula 12 is the estimate of the expected mean of local dimensional densities of the points in the region. In the formula, the local distances whose value is zero are removed from the computation.

Definition 12 (Dimensional average local distance)**.**

$$\overline{\delta_i} = \frac{\sum \gamma_i(q)}{m}, \; m = \left| \left\{ \gamma_i(q) \,/\, q \in S \bigwedge \gamma_i(q) \neq 0 \right\} \right| \tag{6}$$

In Definition 12, $\overline{\delta_i}$ is simply the average of the local distance of every point in region S as computed in Definition 11. As mentioned earlier, we will not consider the points with zero local distances. m is the number of points in S which has the local distances that are not zero. The variable $\overline{\delta_i}$ is used to measure the local density of set S in dimension i.

As mentioned earlier, the absolute distance does not consider that the local variances may differ in each attribute. Since the local density of the region of a point is defined, we can use this density to construct a boundary such that every point outside it can be an outlier candidate. We will normalize the dimensional distance to the point with respect to the density. We will call it the dimensional variance ratio.

Definition 13 (Dimensional variance ratio)**.**

$$r_i(p,q) = \frac{|p_i - q_i|}{\overline{\delta_i}} \tag{7}$$

Formula 7 measures the deviation of point p from point q with respect to the average variance of the points within the proximity of q in attribute i. By normalizing $\left\{\overline{\delta_i}\right\}$, we can consider the ratio to be the distance from p to q with respect to a hypercube of length of 2 units.

We observe that the ratio is close to 1 if p is within the proximity of q. In contrast, those with a large dimensional variance ratio ($r_i \gg 1$) imply that they deviate greatly from the normal local distance in terms of dimension i. We can conclude that they are outliers with respect to q in dimension i.

We have shown in Theorem 4 that an outlier in an m-dimensional space will be an outlier in at least one dimension. Formula 7 is sufficient to detect outliers with respect to q in any subspace. We will show this in the following theorem.

Theorem 5. *Let denote* $\tau(p,q) = \max\{r_i(p,q)\}$, $\forall i$. *If* $\tau(p,q) > \kappa$, *for some large* κ, *then* p *is an outlier to* q.

Proof. We can consider $\{\overline{\delta_i}\}$ as the normalizing constants for all points in region S. Since S is small, we can consider the points within a hypercube R with unit length of 2 where q is its center are normal neighbors of q. Then, $\tau(p,q) > \kappa$ is the distance from p to hypercube R. Since $\tau(p,q) > \kappa$, for some large κ, then p is an outlier to q according to Postulate 1. □

The theorem above shows that if we can detect a point as an outlier in any attribute with respect to a point q using the boundary computed in its local region, we can conclude that it is an outlier. We then can extend the concept of outlier to a set of points as follows:

Theorem 6. *Given a set S, a point q is an outlier in S if $\tau(p,q) > \kappa$, $\forall p \in S$.*

Proof. The result follows directly from Postulate 2 and Theorem 5. □

Since a point can be an outlier in some attributes (with respect to its nearest neighbors), it is natural to aggregate the dimensional variance ratios into one unified metric to represent the total deviation of point p. However, a naive aggregation of the ratios in all dimensions can lead to the problem that the outliers can be suppressed due to the accumulation of random deviation. The accumulation of many small random deviations can make the strong deviations in some attributes less significant. For an example, if the dimensional variance ratios in a region follow the distribution $N(1,\epsilon)$, the total ratio can be as large as $(1+\epsilon)\sqrt{n}$, for normal points according to Equation 4. This value is significant when n is large. The ratio is large not because the point deviates from others but because the small dimensional variations are accumulated during the aggregation. Therefore, we introduce a cutoff threshold ρ_0. We use it to eliminate the insignificant changes when computing the final ratio. The ratios that are less than ρ_0 are not aggregated in order to compute the total deviation.

Definition 14 (Aggregated variance ratio)**.**

$$r\left(p,q\right)=\sqrt{\sum_{i}r_{i}^{2}\left(p,q\right)}\quad,\forall r_{i}\left(p,q\right)>\ \rho_{0} \tag{8}$$

As shown in the definition, instead of naively combining all the ratios, we only combine the ratios that are significant. The cutoff threshold ρ_0 is used as a filter to remove the attributes that do not contribute to the outlier score of point p. As shown later in our experiments, this is effective in improving the quality of outlier detection.

Property 1. *If p is not an outlier with respect to q, then $r\left(p,q\right)=\ 0$.*

Proof. If p is not an outlier with respect to q, then $r_i\left(p,q\right)\leq\kappa,\ \forall i$. If we set $\rho_0=\kappa$, then $r_i\left(p,q\right)\leq\rho_0,\ \forall i$. Thus, we have $r\left(p,q\right)=0$. □

We use the cutoff threshold to eliminate the attributes where p does not appear as an outlier. As a result, the aggregated variance ratio will be zero if there is no attributes such that p is an outlier. This property leads to the next property.

Property 2. *p is an outlier with respect to q if and only if $r\left(p,q\right)>\ \rho_0$ for some ρ_0.*

Proof. If p is an outlier with respect to q, there is at least one dimension i such that $r_i\left(p,q\right)>\kappa$. If we set $\rho_0=\kappa$, $r_i\left(p,q\right)>\rho_0$. Thus, $r\left(p,q\right)>\ r_i(p,q)$. Since $r_i\left(p,q\right)>\rho_0$, we have $r\left(p,q\right)>\ \rho_0$.

Similarly, we can also prove the reverse direction. According to property 1, if p is not an outlier, then $r\left(p,q\right)=0$. Therefore, if $r\left(p,q\right)>0$, there must be at least one attribute such that p is an outlier with respect to q. □

The property shows that when p is an outlier, it is guaranteed that the aggregated value will always be greater than some value. In this case, the value is the cutoff threshold. As we have discussed previously, a point is an outlier with respect to a set if it is an outlier with respect to every point in the set. Therefore, we can define a score function to measure the degree of p as an outlier as follows:

Definition 15 (Outlier score)**.**

$$o\text{-}score\,(p/S) = \min_{q \in S} r(p, q) \tag{9}$$

Formula 9 aggregates the outlier information for a point with respect to every point in S. Since the dimensions where p is not an outlier are excluded, we can guarantee that p is an outlier in S if its *o-score* is high. In other words, even if p is an outlier in only a subset of some attributes in very high dimensional data, the value of $o - mhyphenscore$ for p will still be high (Theorem 6). Thus, $o - mhyphenscore$ is sufficient to measure the degree of outlierness of a point.

The properties above show that Formula 9 can be used to define the degree to which a single point in the dataset is considered an outlier. For a point in a dataset, we will select a region where p is the center. We will compute the dimensional variances for every point in the region. Then, we will compute the aggregated variance ratios between p and every point in the region. If all the ratios are large, we will conclude that p is an outlier. The way we compute the outliers is similar to the LOF method mentioned in Chapter 3. However, the differences are the way we measure the deviation between points. Instead of computing the *k-distance* using all the attributes. We measure the deviation in each attribute. Therefore, the method is more effective in high dimensional datasets.

It should be noted that it is possible for points to appear as a group of outliers. The current definition above only measures the deviation of one individual point. However, we observe that if two points are relatively close in every attribute with respect to the other points in its local region, the value of $o - mhyphenscore$ must be zero for these points. Therefore, if a set of points appear as a group, then *o-score* will be zero for every pair of points. As a result, the *o-score* can be used to cluster a dataset. A cluster is a set of points such that every pairwise *o-score* is zero.

In such cases, we observe that every point in a small group C of outliers should have a large value for *o-score* with respect to any point that is not in the same group.

If there exists a point q that is not in the cluster whose *o-score* value with respect to C is zero, then q should be in the same cluster C. If C grows to be a large cluster, then the points in C are considered to be normal. If there is no more points in the original dataset can not be included in C and C is small, C can be considered to be a set of outliers. Using these observations, we can define a cluster of outliers as follows:

Definition 16. *Outlier cluster in a set S is a set of points C such that o-score $(p/S - C)$ $> \rho_0, \ \forall p \in C$ and $r(p,q) = 0, \ \forall p, q \in C$.*

When the pairwise deviation between the outliers is small with respect to the average local distance in all dimensions, the outliers naturally appear as a cluster. This fact is captured by the second condition in the formula. In the following definition, we can define the degree of an outlier cluster as follows:

Definition 17 (Outlier cluster score)**.**

$$o\text{-}score\,(C/S) = \min_{q \in C} o\text{-}score(p/S - C) \tag{10}$$

Thus far, we have introduced the definitions of outlier which conform to the intuitive outlier criteria. The hypercubes for the points in a sample are bounded by $\{\overline{\delta_i}\}$. Definition 13 defines the ratio of deviation between any two points with respect to the average local variance in a dimension. We can interpret this as a similarity function between two points relative to the average variance in one dimension. If a point is dissimilar to all points in at least one dimension, it is an outlier. Definitions 16 and 17 extend the concept of outlier to an outlier cluster. It allows us to detect the clusters of outliers in a dataset. The clusters can be detected using Definition 16. The degree of being a cluster of outliers can be computed by $o\text{-}score(C/S)$.

One interesting effect of the dimensional ratio is that we can identify the attribute in which a point is an outlier. This can then be used to visualize the outliers by plotting these attributes. This is useful for datasets which have a large number of attributes.

5.5 Clustering

As discussed above, clusters of outliers can be detected by using the outlier score function. We can use an edge to represent the link between two points. If the aggregated variance ratio between two points is zero, there will be an edge connecting two points. A cluster is a set of connected points. When the size of a cluster grows large, we are certain that the points in the cluster are normal since a point can always find at least one point close to it in the graph. However, if the points are outliers, there will be no edge that connects the outliers with other points. Thus, the cluster will be small. This is, in fact, similar to the clustering algorithm, the algorithm in Figure 16 based on the dual-neighbor relationship in Chapter 4 where two points appear in the same cluster if they have a dual-neighbor relationship. Similarly, in our case, the dual-neighbor relationship can be measured by the aggregated variance ratio. This method is a natural extension of the adaptive dual-neighbor method. The difference is that in this chapter, we consider the variance in each attribute, so only line 13 is changed.

As shown in Figure 22, we put a point p into a stack S and create a new cluster C. Then, we take point p and put it in C. In addition, all of the points which have the aggregated variance ratio of zero with respect to p will be put into S. For each q in S, we expand C by removing q from S and adding q to C. The points with the aggregated variance ratio of zero with respect to q are then put into S. These steps are repeated until no point can be added to C. We then create a new cluster C'. These steps are repeated until S is empty.

We now show that the algorithm can detect clusters according to our definition in the following theorem.

Theorem 7. *Let say $\{C_i\}$ is the set of clusters produced by the algorithm, C_i contains no outlier with respect to C_i, $\forall i$.*

```
1: procedure CLUSTER(HashSet D)
2:     Stack S
3:     Vector clsSet
4:     HashSet C
5:     while D ≠ ∅ do
6:         p ← remove D
7:         push p into S
8:         C ← new HashSet
9:         add C into clsSet
10:        while S ≠ ∅ do
11:            q ← pop S
12:            add q into C
13:            for all r ∈ neighbors(q) ∧ r(q, r) = 0 do
14:                push r ← S
15:                remove r from D
16:            end for
17:        end while
18:    end while
19: end procedure
```

Figure 22: Clustering pseudocode

Proof. Assuming that a point $r \in C_i$ is an outlier in C_i, we have $r(q,r) > \rho_0, \forall q \in C_i$ (property 2 and Postulate 2). According to the algorithm from lines 10 to 17, a neighbor r of a point q is put into C_i iff $r(q,r) = 0$, which contradicts the condition above. Therefore, C_i contains no outlier with respect to C_i. □

Theorem 7 shows that the clusters produced by the algorithm do not contain any outlier. Therefore, an individual outlier will be a point that does not belong to any cluster computed using our algorithm. For a large cluster C, we consider it as a set of normal points. If C is very small, we will compute the outlier cluster score for C. If the score is significantly large, C is a cluster of outliers. Therefore, the algorithm using the outlier cluster score allows us to detect not only individual outliers but also groups of outliers.

5.6 Experiments

5.6.1 Synthetic Dataset

We create a small synthetic dataset D to illustrate our outlier score function. We use a two dimensional dataset so that we can validate the result of our algorithm by showing that the outliers and groups of outliers can be detected. The data consists of 3000 data points following a normal distribution $N(0, 1)$. We generate three individual outliers $\{p_1, p_2, p_3\}$ and a group C_1 of 10 outliers $\{p_1, \ldots, p_{10}\}$. These additional points are injected into the dataset. We place the three outliers randomly, whereas we place cluster C_1 next to the synthetically generated Gaussian. The injected points and clusters are used to verify that the algorithm can detect outliers correctly. The dataset is illustrated in Figure 23.

First, we compute the *o-score* for all the points in D with $\rho = 2$ and $\alpha = 0.4$. The algorithm detected 5 outliers. The outliers consist of three individual injected outliers. The other two outliers are in the synthetically generated Gaussian. In the figure, we see that the points are also outliers in the Gaussian. In this list, we see that the injected outliers $\{p_1, p_2, p_3\}$ appear in the top three outliers. The next two outliers which are generated from the distribution have low scores. The scores are approximately half of the scores of the injected outliers. The low scores can be explained by their random deviation from the center of the Gaussian. The scores are shown in Table 9. In other words, the injected outliers are correctly identified as top outliers.

Next, we run the clustering algorithm based on the computed *o-score* as described in the clustering section. The algorithm detected 9 clusters. Among them, two clusters have the scores of zero. Thus, seven outlier clusters are detected. We see that ten points in the manually generated cluster are detected. The algorithm also groups them into one cluster correctly. These points appear to be the highest ranked outliers. In addition to the injected cluster, a micro cluster C_2 of five outliers is also

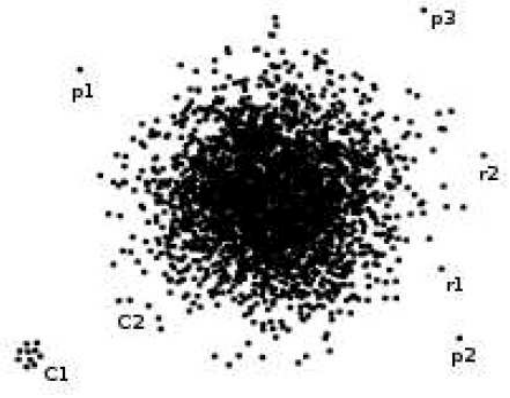


Figure 23: Points p_1, p_2, p_3 and cluster C_1 are generated outliers

Table 9: Seven outliers are detected

Point	Score
p_1	7.24
p_2	6.68
p_3	5.98
r_1	2.97
r_2	2.92
others	0

detected. This cluster is in fact a subset of the Gaussian. Similar to the outliers generated by the Gaussian discussed earlier, the points in the cluster have low scores. It is due to the fact that they are just randomly generated from a normal distribution.

In this example, we have shown that our algorithm can discover micro clusters. However, it should be noted that our algorithm can detect clusters of any size. Therefore, the algorithm can also be used to detect any cluster for any application. As a side effect, there is no limit on the size of the micro clusters that the algorithm can detect. We can adjust the threshold for the size of the micro cluster. It is useful for applications where outlier clusters may be large (but still relatively small with respect to the size of the major clusters in the dataset).

5.6.2 KDD CUP '99 Dataset

The experiment above shows that our algorithm can detect outliers. In this experiment, we will show that our method can also detect outliers in practice. We use the KDD CUP 99 network connection dataset from the UCI repository [112] for this experiment. We test the ability of outlier detection in detecting the attack connections without any prior knowledge about the characteristics of the network intrusion attacks. Our outlier detection will be based on the hypothesis that the attack connections may behave differently from the normal network activities. The deviation from the normal network activities makes them outliers.

We created a test dataset from the KDD original dataset with 97,476 connections. Each record has 34 continuous attributes. The attributes represent the statistics of a connection and its associated connection type, i.e. normal, buffer overflow attack. The original dataset contains a large number of attack connections. For the outlier detection purpose, it is unrealistic since the number of outliers should be very small. Otherwise, we should treat them as normal activities. Therefore, we randomly select a very small number of attack connections in the dataset. The dataset contains 22 types of attacks. The sizes vary from 2 to 16. Totally, there are 198 attack connections which account for only 0.2% of the test dataset.

In this experiment, we run the local outlier factor (LOF) method described in Chapter 3 as a baseline to test our method since it is an effective detection method that can detect density based outliers. Since LOF is sensitive to $MinPts$, we run LOF on the dataset with different values of $MinPts$ from 10 to 30. In this dataset, LOF has the best result with $MinPts = 20$ has in terms of the number of detected attacks. However, even with the best result, the top 200 outliers do not contain any attack connections. The detected attacks are ranked below 200. In the experiment, only 20 attacks are detected. The ranking of those attacks varies from 200 to 1000. The top 2000 outliers contain only 41 attacks. The experiment shows that the LOF

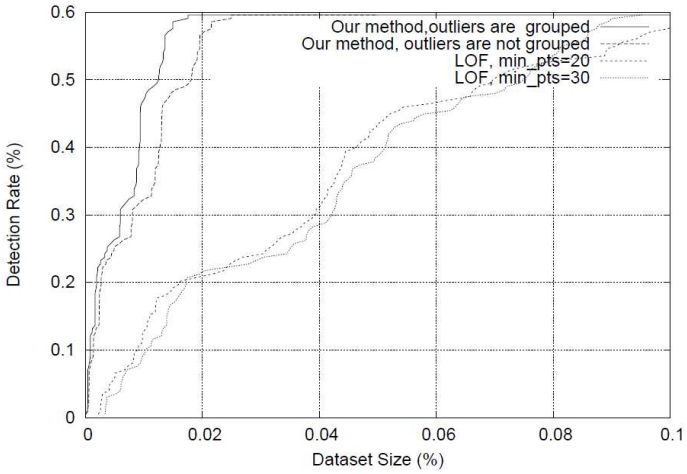


Figure 24: Detection curve of the filtering method on KDD Cup '99

method is ineffective in this experiment.

We then ran our method on the dataset with the same value of ρ_0 (2.2) and α used in the synthetic dataset. However, we increase the size of the local region because the dataset for KDD is larger than the synthetic dataset. The size is set to 100. The method returns the list of outlier clusters ordered by score. The sizes of those clusters are small. Most of clusters contain one outlier. The outlier clusters are ranked by outlier score. According to the results, one attack is found in the top 10 outlier clusters and 16 attacks are found in the top 50 outlier clusters. Between these clusters, the cluster with rank of 38 contains 9 outliers. Interestingly, all of these outliers are attack connections. We studied the outliers and found that all outliers in this group are warezmaster attacks. It shows that our algorithm can discover micro clusters. Since the dataset contains only 12 warezmaster connections in a dataset of more than $90,000$ points, the method achieves high accuracy for this tiny cluster. When we studied the results for the next outliers in the list. We found that 42 attacks are discovered the top 200 outliers and 94 attacks are detected in top 1000.

The experiments show that our method outperforms LOF. In LOF, no outliers are detected in the top 200. In our method, one attack is detected in the top 10, 16

Table 10: Nine outlier clusters are detected in 2D dataset

Cluster	Size	Score	Items
1	10	7.7	$q_1, q_2,\ q_3, q_4, q_5, q_6$ $, q_7, q_8, q_9,\ q_{10}\ (C1)$
2	1	7.24	p_1
3	1	6.68	p_2
4	1	5.98	p_3
5	1	2.98	r_1
6	1	2.92	r_2
7	5	2.43	$r_3, r_4, r_5, r_6, r_7(C2)$
8	2	0	r_8, r_{10}
9	1	0	r_{11}

in the top 50, and 42 in the top 200. In the top 1000, we detect 94 outliers whereas LOF only detects 20 outliers. In conclusion, our algorithm yields a higher order of magnitude for accuracy.

Figure 24 shows the detection curve with respect to the number of outliers. In this curve, we show the detection rate for LOF with $MinPts = 20$ and $MinPts = 30$. In addition, the figure also contains the curves for our method with the ranking in terms of individual outliers. Since our method can cluster the outliers, we also show the curve in terms of clusters of outliers. In the ranking by cluster, each individual outlier is considered to be a cluster of size 1. Figure 24 shows that LOF with $MinPts = 20$ is slightly better than LOF with $MinPts = 30$. However, the detection rate of our method is significantly better than the rate of LOF in both cases. The rate of our algorithm is 60% when the size of outliers is 0.02% of the dataset, whereas that of LOF is 21%. Given the context that outlier detection in general can have very high false alarm rates, our method can detect a very small number of attack connections in a large dataset. Also as shown in Figure 24 , the detection of our method in terms of clusters is slightly improved. This is because our method clusters the similar points together. The number of outliers for both normal and attack connections is reduced. Therefore, the outliers appear to be in the higher ranks.

Table 11: Detected attack connections in KDD CUP dataset

Rank	Size	Score	Rank	Size	Score
7th	1	152.6	72nd	1	15.7
30th	1	38.7	79th	6	14.8
32nd	1	34.4	80th	1	14.7
36th	1	32.5	111st	1	11.9
37th	1	32.2	113rd	1	11.5
38th	9	32.1	158th	1	8.5
54th	1	22.3	159th	9	8.5
62nd	1	19.4	163rd	1	8.3

Table 7 shows the ranking and the cluster size for the top 200 outlier clusters. According to the table, three clusters of attacks are found. The first cluster whose ranking is 38^{th} contains nine warezmaster attacks (recall rate = 75%). The next cluster contains six satan attacks (recall rate = 75%). The last cluster in the table contains nine neptune attacks (recall rate = 100%). The misclassification rate for those clusters is zero. The recall rate for those attacks is very high given that each of them accounts for less than $1.2 \times 10 - 4\%$ of the dataset.

The experiments show that our dimensional ratio method improves the quality of outlier detection.

5.6.3 The Effect of the Filter Parameter

In this experiment, we study the effectiveness of the filter parameter on the quality of the detection rate of our method. We evaluate the effect of the filter parameter by adjusting its value. With the value of 1 ($\rho_0 = 1$), it is equivalent to having no filter because all the variances will be aggregated.

Figure 25 shows the detection rate for our algorithm when $\rho_0 = 2.2$ and $\rho_0 = 1$ respectively. We also show the curve for LOF with $MinPts = 20$. Even without using the filter, according to the figure, our method still consistently performs better than the LOF method. The graph also shows that the algorithm can discover 27 attacks in the top 200 outliers, whereas LOF can not detect any attack connection in the top

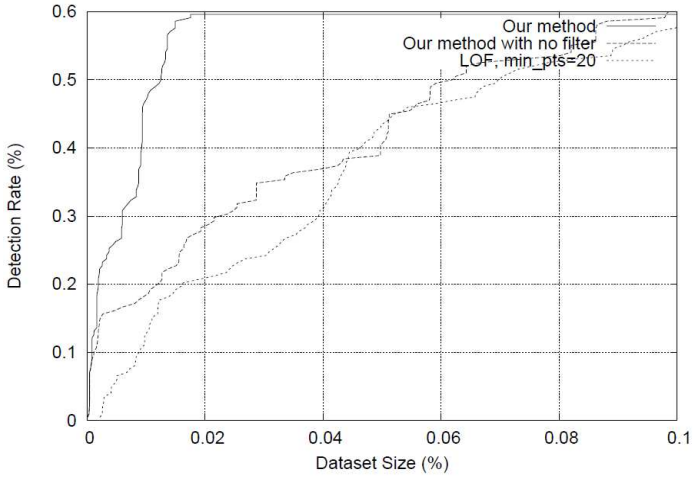


Figure 25: Detection rate for the algorithm with/without using the filter

200 outliers. The better performance can be explained by the fact that the variances for all dimensions are normalized by using the dimensional ratios. When we applied the filter when aggregating the score, the detection rate is boosted greatly. As shown in the figure, in the top 200 outliers, the detection rate for the filter approach is twice that of the test without the filter. Therefore, we can conclude that the filter is effective in eliminating the random deviation in computing outlier scores. The quality of outlier detection is improved.

5.6.4 Selecting ρ_0

The experiment in the previous section shows the result of the algorithm when the filter is applied with $\rho_0 = 2.2$. However, we have not discussed how to choose a value of ρ_0. One guideline is to select ρ_0 based on the minimum rate of deviation that we are interested in. For an example, the choice of 2.2 implies that we are only interested in the points with the deviation with respect to its dimensional variance greater than 2.2. Otherwise, we will not consider them as outliers. Another guideline is based on the number of detected outliers in the dataset. If the dataset contains many strong outliers, we can increase ρ_0 so that the weak outliers will be eliminated. From the experiments, 2.2 seems to be a reasonable value for ρ_0.

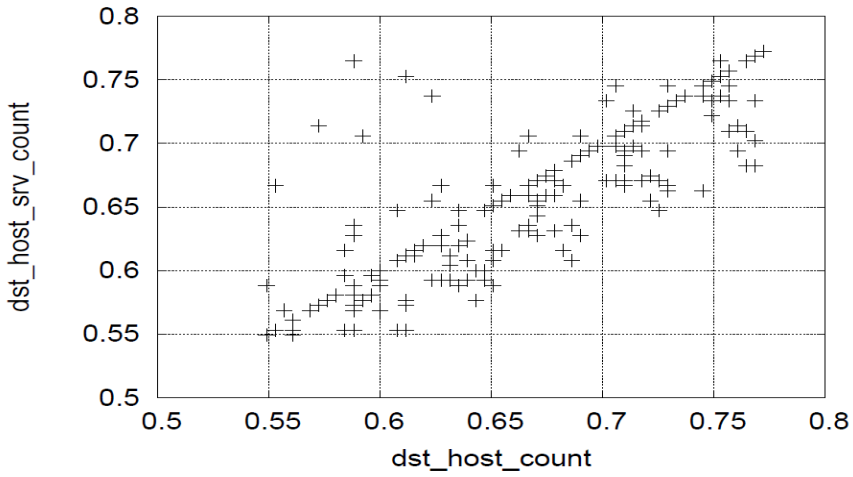


Figure 26: Point p_{36} is not an outlier in this 2d-subspace

5.6.5 Visualization

As we discussed in section 5.4, an outlier in our definition must appear as an outlier in at least one attribute by Theorem 6. This property allows us to study why a point is an outlier. For an outlier, we can select the attributes where its dimensional variance ratios are high. The subsets of attributes form low dimensional spaces, i.e. 2D and 3D. Therefore, we can visualize the outliers with respect to their local region in order to study the significance of the outliers.

In this section, we take the results of the KDD experiments to study the outliers. First, it is noted that in addition to the ranking of the outliers, our method also returns the attributes in which a point p becomes an outlier by checking for an attribute i in which $r_i(p) > \rho_0$. Table 12 shows the dimensional score for two points p_7 and p_{36}. The points p_7 and p_{36} are multihop and back attacks respectively. According to the table, p_7 is an outlier in the 2^{nd} and 29^{th} dimensions. These dimensions correspond to the attributes dist_bytes and dst_host_srv_diff_host_rate. The point p_{36} is an outlier in the 1^{st} and 26^{nd} dimensions. They correspond to attributes src_bytes and dst_host_same_srv_rate.

Figures 26 and 27 show two 2-dimensional subspaces for point p_{36} and its local

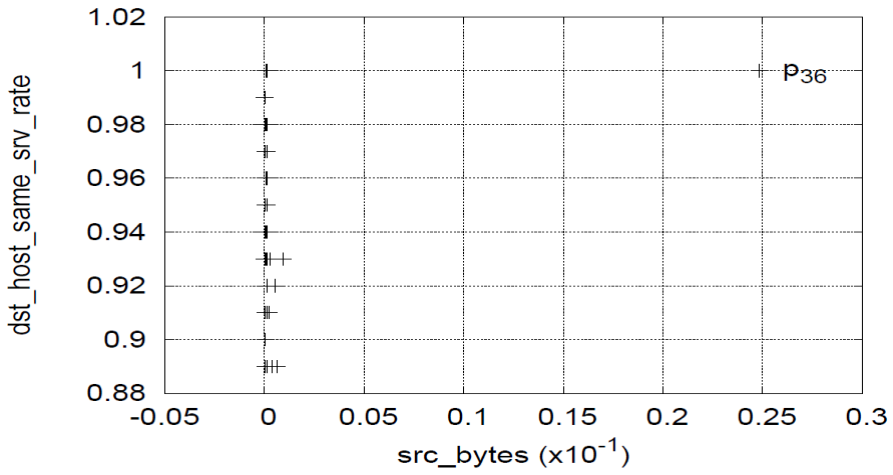


Figure 27: Point p_{36} is an outlier in the 1^{st} and 26^{th} dimensions

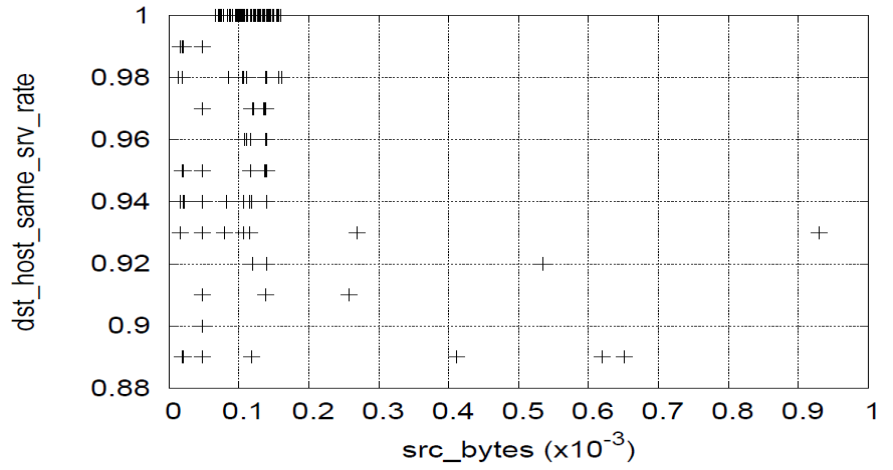


Figure 28: Local region with p_{36} removed in the 1^{st} and 26^{th} dimensions

region projected on these two subspaces. In Figure 26, the subspace consists of two dimensions where p_{36} is not an outlier. As we can see, we cannot recognize p_{36} in the region. There is no strong outlier in the figure. However, when we plot a 2-dimensional subspaces of the 1^{st} (src_bytes) and 26^{th} (dst_host_same_srv_rate) dimensions, p_{36} appears as an outlier as shown in Figure 27. Point p_{36} is clearly distinct from its surrounding points. Figure 28 shows the distribution of local region of the 1^{st} (src_bytes) and 26^{th} dimensions with p_{36} removed from the region.

By using the visualization, we can study the distribution of the local region of an

Table 12: Dimensional and aggregated outlier scores for p_7 and p_{36}

Point	Rank	Total Score	r_{i}	r_{i}
p_7	7	152.6	r_2= 152.57	r_{29}=2.3
p_{36}	36	32.5	r_1= 32.4	r_{26}= 2.3

outlier as shown in Figures 26 and 27. The figures also allow us to explain why p_{36} is not well detected by the LOF method. According to LOF, p_{36} has a score of 2.1. The point ranks 6793^{th} in the list of outliers. The score implies that its k-distance in Euclidean space is only twice the average of k-distance of its neighbors. If we use the Chebyshev distance, we will have $k - dist(p_{36}/k = 30) = 0.066$. The average $k - dist(q_i/k = 30)$ is 0.04 for $\{q_i\}$ is one of the 4-nearest neighbors of p. The k-dist of p_{36} approximates that of its surrounding neighbors in both Euclidean and Chebyshev space. As a result, p_{36} cannot be detected in the traditional approach. In our sub dimensional score aggregation approach, we can identify p_{36} as a strong outlier in the 1st dimension. During the aggregation step, the other insignificant deviations are removed. Therefore, p_{36} shows up correctly as an outlier.

5.7 Conclusion

We have shown that the problems are created by a mixture of variances and the accumulation of noise in high dimensional datasets for outlier detection. We introduced a bottom-up approach to compute the outlier score by computing the ratios of deviations for each dimension. We then aggregate the dimensional scores into one final score. Only dimensions in which the ratios are high will be aggregated. Since the dimensions with high variances are treated the same as those with low variances in our approach, this method solves the mixture of variances problem. In addition, we also introduce the use of the filter threshold to solve the problem of random deviation in a high dimensional dataset by preventing many small deviations from being accumulated into the final outlier score. According to the experiment, the filter has significantly boosted the performance of outlier detection. In addition, our method

allows us to visualize the outliers by drawing the graphs in the dimensions where the points deviate from others. By studying the graphs, we can eliminate the dimensions in which the outliers are not interesting to us and it can be explained why the outliers are interesting. We also apply the clustering technique from [109] to cluster the outliers by observing that two points whose *o-score* between them is zero are close to each other and should be in the same cluster. Thus, our method can also produce clusters of outliers. The experiments with the KDD CUP '99 dataset have shown that the detection rate in our method is improved compared with that in the traditional density-based outlier detection method.

CHAPTER VI

EFFICIENT ANOMALOUS CORRELATED ATTRIBUTE PATTERN DETECTION IN MULTI-DIMENSIONAL DATA

6.1 Motivation

In Chapter 4, we discussed the adaptive dual-neighbor method to discover anomalous patterns. In our definition, an anomalous pattern is a micro cluster that deviates greatly from the masses. The anomalous pattern is cluster-based. In this chapter, we introduce another type of anomalous pattern. It is not cluster-based. The anomalous pattern is rather based on the correlation between the attributes. For an example, if an attribute X of an observation belongs to an interval I_1, then its attribute Y will belong to an interval I_2. A set of observations follow the same rule is said to belong to the same pattern. The problem of anomalous patterns using correlated attributes is different from the problem of cluster-based anomalous pattern. It can be explained using the following example. Let say we have a dataset $D = \{< 1, 100 >, < 2, 200 > , < 3, 300 >, < 100, 1 >, < 200, 2 >, < 300, 3 >\}$. In terms of attribute correlation, D will be divided into two different rules $D_1 = \{< 1, 100 >, < 2, 200 >, < 3, 300 >\}$ and $D_2 = \{< 100, 1 >, < 200, 2 >, < 300, 3 >\}$. However, in terms of cluster-based, all the observations in D belongs to the same cluster. The choice of anomalous pattern type is application dependent. If we are interested in the correlation between the attributes, the anomalous pattern based on attribute correlation should be used. If we are interested in anomalous clusters of observations, the cluster based anomalous pattern method should be used.

Although this anomalous pattern detection seems to be similar to quantitative

correlation analysis, there are two distinct differences. First, the correlation analysis focuses on mining sets of strongly correlated intervals, whereas, we are only interested in observations that follows an anomalous pattern. In correlation analysis, a small set of strongly correlated observations is not equivalent to an anomalous pattern. It is due to the fact that a large pattern may be divided into smaller patterns. However, in our cases, if there is a similarity among the observations, then they are still considered as one pattern

The main challenge is to define a set of strongly correlated values. Our contribution is to introduce a method to identify strongly correlated values very efficiently by partitioning the dataset on attributes and learning the patterns locally. We, then, introduce a method to combine the patterns together efficiently. The remaining small patterns are anomalous patterns. We will discuss our method in detail in the next sections.

6.2 Comparison with outlier detection and clustering

As mentioned earlier, we are interested in detecting rules such as if X is in I_1, then Y should be in I_2. This rule has many practical applications. For an example, by mining a dataset, we can learn that most people within the age of $[0, 5]$ do not own a house. Therefore, it is unusual for a small subset of people within this age group to own a house.

One possible solution is to apply an outlier detection or clustering method to detect this kind of anomaly. An outlier or a micro cluster is an anomalous pattern. However, we will show that this does not always work using the following example.

Let say we have a small dataset of ten items. There are three attributes. The domains of the attributes are $[0, 1000]$. Since the ranges of the attributes are the same, we do not have to normalize the dataset. The dataset is shown in Table 13. In the table, we see that the dataset can be clearly divided into three groups of items based

Table 13: An example dataset contains three groups.

ID	A_1	A_2	A_3
X_1	8	250	300
X_2	9	250	250
X_3	10	350	150
X_4	50	300	200
X_5	40	320	800
X_6	30	250	900
X_7	20	200	1000
X_8	1000	800	10
X_9	900	900	20
X_{10}	800	1000	30

on the similarity between the attributes. The groups are $S_1 = \{X_1, X_2, X_3, X_4\}$, $S_2 = \{X_5, X_6, X_7\}$, and $S_3 = \{X_8, X_9, X_{10}\}$.

We will study $S_1 = \{X_1, X_2, X_3, X_4\}$. Table 14 shows the four items in S_1 with three attributes A_1, A_2 and A_3. We compute the statistics about the distances between the items. We measure k-distances, the average k-distance, and the local outlier factor[24] of the items. The statistics are shown in Table 15. In the table, the rows in the table show the k-distance [24] with $k = 3$, the average distance of all k-nearest neighbors (3 nearest neighbors), and the LOF factors of the items. As mentioned earlier, because the domains of the attributes are $[0, 1000]$, normalization is unnecessary. According to the table, X_1 and X_2 are highly ranked outliers due to their high LOF scores and X_4 is the lowest ranked outlier in the group. X_1 and X_2 are more unusual than X_4. However, a closer look at Table 14 shows that this is not necessarily true.

According to the table, the values for attributes A_2 and A_3 are actually almost the same for all the items. The values vary uniformly from 150 to 350. The items are similar in terms of A_2 and A_3. For attribute A_1, there are two distinct groups: the group of X_1, X_2 and X_3 with the mean of 9 and a group of X_4. The first attribute of X_4 deviates significantly from the other items in the table. It is five times greater than the average of group of X_1, X_2 and X_3. Despite this abnormality, X_4 is considered

Table 14: A subset of correlated observations

ID	A_1	A_2	A_3
X_1	8	250	300
X_2	9	250	250
X_3	10	350	150
X_4	**50**	300	200

Table 15: Statistics of X_1, X_2, X_3 and X_4

	X_1	X_2	X_3	X_4
k-distance	180.29	141.42	180.29	119.43
Mean	116.58	91.06	134.32	94.14
LOF Score	1.55	1.55	1.34	1.27

less of an outlier than X_1 and X_2 according to the definition of LOF. Even though the range of A_1 is smaller than those in A_2 and A_3, the deviation is significant. In this example, X_4 is unusual whereas X_1, X_2 and X_3 are not.

We consider an alternative approach to discover the unusual item X_4. It is based on computing the outliers on all subspaces. There are two problems. First, the number of subspaces is exponential. Second, in this case, item X_4 can not be detected. For instance, if we look at attribute A_1, X_4 seems to be similar to the other items. This example shows that the problem of anomaly detection based on the correlation between attributes is orthogonal to the clustering-based method.

In the following sections, we formally define the concept of feature-based anomalous pattern. Afterwards, we present a method to detect such patterns based on the correlations between the attributes. It should be noted that correlation is not the same as cluster. In clustering, the goal is to find cluster boundaries of a dataset. An item must be belong to one cluster. However, in a high dimensional dataset, there may be many different ways to cluster a dataset. In correlation analysis, an item can belong to different rules. We just need to find a rule to which an item belongs. Therefore, it allows the anomalous patterns to be detected more efficiently.

6.3 Related work

Association rule mining is first introduced by Agrawal et al [5]. The method is used to mine relationships between items in a market basket. In this method, transactions (i.e., baskets) are examined to see if certain items appear together in enough of the transactions to make that pattern interesting. For example, maybe items A and B occur together in 20% of the transactions. Because of this, the pattern A and B may be considered interesting. The 20% is referred to as the support of the AB pattern. The percentage of basket containing B given A is the confidence of the rule. The confidence measures the certainty that A implies B. This is similar to our method in terms of finding correlations between two attributes. However, it is only for binary attributes. In addition, it does not detect anomalous patterns.

An association rule is a predicate if there is an X, then there will be an Y. However, in many cases, an attribute may have continuous values. Ke et al [82] extend the association rule mining to quantitative correlated pattern mining. A quantitative correlated pattern states that if X is within an interval A, then Y is within an interval B. Quantitative correlated pattern mining is similar to searching for anomalous patterns in terms of detecting correlated intervals between attributes. However, there are several differences between two methods. Quantitative correlated pattern mining counts the correlated intervals with high support. The method can return an exceptional large number of correlated intervals. The number of correlated intervals grows rapidly with the number of attributes. However, a strong correlated pattern does not mean that all items in a pattern anomalous. For example, a large set of similar records can be partitioned into multiple quantitative correlated patterns because the values of the records may be concentrated more on some particular intervals. In our approach, to be an anomalous pattern, a correlated pattern must deviate greatly from all other patterns. Therefore, quantitative correlated pattern mining is not suitable for anomalous pattern detection. One suggestion is to compute the distances

between the correlated patterns to identify anomalous patterns. The patterns that deviate from the other patterns are anomalous. However, there exists two problems. The first problem is that the number of patterns returned by the quantitative correlated pattern is very large. The second problem is that it is in fact reduced to the problem of anomalous detection whose task is to compute anomalous patterns efficiently.

Density-based clustering methods cluster a dataset based on the local density. The nearby items with the same density are clustered together [52] [75]. The density-based clustering methods can discover clusters with different sizes and shapes [52]. Although an anomalous pattern can be considered as a small cluster, a small cluster is not always anomalous. For some density-based clustering methods, a small cluster may be the result of a large cluster that is partitioned into multiple smaller clusters. In our method, the items in an anomalous pattern must show be a strong correlation and must deviate from the remaining items in the dataset. In addition, since we focus on discovering anomalous patterns, the items with high similarity are quickly pruned from the dataset. We do not have to cluster them together. As a result, our algorithm runs faster.

6.4 Relational Algebra

In this section, we will extend the definition of relational algebra to incorporate the concept of interval tuples and the operations on interval tuples. Recall that a relation R consists of n attributes $A_1, \ldots, A_n$, a tuple t of R is an ordered list of values corresponding to the attributes. The notation $t.A_i$ refers to attribute A_i of tuple t. For instance, in Table 14, we have a relation R of four tuples X_1, X_2, X_3, X_4. For tuple X_4, we have $X_4 =< 50, 300, 200 >$. The value of attribute A_1 of X_4 is denoted as $X_4.A_1 = 50$. Where applicable, we drop the attribute name and use the subscript, i.e. t_i, to refer to the attribute for brevity. In addition, we use the terms feature and

attribute interchangeably.

An interval is a set of real numbers bounded by two end points, which can be represented by $[a, b]$. An interval-tuple I is an ordered list of intervals $\{I_i\}$. For an example, an interval-tuple $I \equiv< [7, 10], [250, 250], [250, 350] >$ consists of three intervals $I_1 = [7, 10]$, $I_2 = [250, 250]$, and $I_3 = [250, 350]$. It should be noted that in our notation, an interval can be *null*. Thus, we can have an interval-tuple as follows: $I \equiv< [7, 10], [250, 250], null >$. In this case, I_3 is *null*.

With the concept of interval-tuples, we can perform the set selection operation using the intervals as the upper bounds and lower bounds for the attributes in a relation. The interval selection operation $\sigma_I(R)$ selects a subset S of the tuples from R such that each tuple t in S satisfies the following condition: $t_i \in I_i$,$\forall I_i \neq NULL$. We say that the selected set S is covered by I. I is a cover interval-tuple of S and I_i is a cover interval of S on attribute i. If I has only one not *null* interval, say I_i, $\sigma_{I_i}(R)$ can be used. In this case, we say $\sigma_{I_i}(R)$ is an interval selection operation on interval I_i for R.

In Table 14, we have a relation R that consists of X_1, X_2, X_3 and X_4. The operation $\sigma_I(R)$ on $I \equiv< [7, 10], [250, 250], [250, 350] >$ returns $\{X_1, X_2\}$. The operation $\sigma_{I_1}(R)$ with $I_1 \equiv< [7, 10]$ returns $\{X_1, X_2, X_3\}$. The set $S \equiv \{X_1, X_2\}$ is covered by I.

We also define a function $\omega(S)$ that returns the smallest intervals that cover S. It is an inverse function of the function σ. In the example above, $\omega(S)$ returns $< [8, 9], [250, 250], [250, 300] >$. In other words, the function $\omega(S)$ computes the lower bounds and upper bounds for every attribute in S.

6.5 Core Rules and Patterns

Before proceeding with the formal definitions, we will briefly discuss about our correlated attributes based approach.

We can say that if two observations belong to the same pattern, then they are similar. However, the definition based on similarity in terms of attribute values is restrictive. The size of a pattern will be small. If a pattern is large, the observations in the pattern may be actually very different. It contradicts the definition based on similarity above. We observe that in anomalous pattern detection, the similarity between observations is not as important as the continuity in the pattern. There should be no gap between the observations in a pattern. Otherwise, the pattern should be split into two patterns. Therefore, we divide our method into two phases. The first stage is to detect core rules . A core rule is a set of strongly correlated observations. We use the core rules to build patterns. In the second phase, we will consider the grouping of a core rule with other core rules because of the similarity between some observations among the rules. Therefore, we will combine all the similar core rules. A set of combined rules is a pattern. The next section will discuss how to identify core rules and how to combine patterns.

6.6 Pattern Construction

In this section, we will assume that the core rules of a dataset are already found. We will discuss how to construct the core rules in the next section.

First, we will introduce an α function to measure the similarity between two intervals as follows:

Definition 18. *α function between two interval-tuples on attribute i is defined by:*

$$\alpha(I_i, J_i) = \frac{\max(\inf I_i, \inf J_i) - \min(\sup I_i, \sup J_i)}{\min(\sup I_i, \sup J_i)}, \tag{11}$$

$$\textit{if } I_i \cap J_i \equiv \varnothing$$

$$\textit{and } \alpha(I_i, J_i) = 0 \textit{ if } I_i \cap J_i \neq \varnothing \tag{12}$$

In the function α, we compute the ratio of the difference between two intervals

instead of the distance between two intervals. Since we are only interested in anomalous pattern, we will not consider two overlapping intervals. Thus, we set α to zero. If two intervals are close, the ratio will be small. The α function can be extended to interval-tuple. If there exists an attribute i such that the α between two intervals is large, we consider those interval-tuples distinguishable. The concept of the α function can be used to measure the closeness between two sets of tuples as follows:

Definition 19. *Two sets S and S' are close under α_c, denoted by $close(S, S'/\alpha_c) \equiv$ true if and only if $\alpha(\omega_i(S), \omega_i(S')) < \alpha_c, \forall i \in 1 \ldots n$.*

To measure the closeness between two sets, first, we will compute the corresponding cover interval-tuples of two sets S and S'. Let say we have $I \equiv \omega(S)$ and $I' \equiv \omega(S')$. The ω function computes the lower bounds and upper bounds for each attribute of S and S'. Next, we compare the closeness between two interval-tuples using the α function. If there is no attribute such that the ratio is large, we conclude that S and S' are indistinguishable. They are considered to be close. The parameter α_c is used to tune the closeness.

As discussed in the previous section, after computing the core rules, we need to combine the similar rules into patterns. We use the $close(S, S')$ function two measure the similarity between core rules. If two core rules are close under α_c, we combine them into a pattern. We can extend a pattern by combining it with other rules. Therefore, a pattern is defined as:

Definition 20. *Given $P = \{S_i\}$, if $\forall S_i \in P$, there is at least one $S_j \in P$ such that S_i and S_j are close under α_c, we say $\{S_i\}$ forms a pattern P under α_c. We define $supp(P) = \sum |S_i|$ as the support of P.*

The definition above shows that we can chain a pattern with other patterns if there exists two core rules that belong to two patterns and they are close. Therefore, a pattern can be extended to be a large pattern. The *supp* function measures the

number of observations that a pattern contains. If a pattern contains only a small number of observations or has low support, we can conclude that there are a small number of observations that are similar to them. Therefore, the observations in the pattern are unusual. We can formally define an unusual pattern as follows:

Definition 21. *We say a pattern P is unusual under N_u if $supp(P) \leq N_u$.*

The parameter N_u limits the size of a pattern to be considered to be unusual. A pattern that is not unusual is considered normal. Thus, we have:

Definition 22. *We say a pattern P is normal under N_u if $supp(P) > N_u$, where N_u is a user defined value.*

We have introduced several definitions to measure the similarity between core rules. We have also introduced a metric to combine rules into patterns. In the next section, we will discuss how to construct core rules from a dataset.

6.7 Core Rule Construction

As mentioned in section 6.5, it is a non-trivial task to produce patterns directly from a dataset since two observations in the same pattern may be in deed very different. We overcome this problem by introducing the concept of a core rule. We construct the patterns based on core rules instead of constructing the patterns directly from the dataset. In a pattern, a core rule is a set of strongly correlated observations. Before defining the strength of the correlation between observations, we first look at only one attribute of the dataset using Table 13. We discuss the intuition behind our method.

Let say we have a dataset $S \equiv \{8, 9, 10, 20, 30, 50, 800, 900, 1000\}$. Obviously, based on only attribute A_1, we can split S into two groups based on the differences in the values. The groups are $S_1 \equiv \{8, 9, 10, 20, 30, 50\}$ and $S_2 \equiv \{800, 900, 1000\}$ which correspond to $\{X_1, \ldots, X_7\}$ and $\{X_8, X_9, X_{10}\}$ respectively. In our work, we

call the difference a gap. Thus, the first criteria to split one attribute is to find for the gaps and then split the values based on those gaps. However, with multiple attributes, this criteria is not sufficient. As in Table 13, there is a gap for attribute A_3 between the values 300 and 800. Therefore, S_1 should be partitioned in two subsets $\{X_1, X_2, X_3, X_4\}$ and $\{X_5, X_6, X_7\}$. As a result, we see that we can perform the partition of a dataset based on each attribute in order to obtain the core rules. First, we can pick up one attribute, say A_1, and partition the dataset based on this attribute. We then perform the similar partitions on the remaining attributes. This method will produce a set of candidate core rules. However, there is still one remaining issue with this method.

Let say we have a dataset $V \equiv \{< 0, 5 >, < 1, 6 >, < 2, 7 >, < 5, 4 >, < 4, 3 > , < 3, 2 >\}$. We will partition the data set using the method discussed above. If we consider the first attribute, we will have an interval containing $\{0, 1, 2, 3, 4, 5\}$. There is no gap between them. The dataset will not be split. We then consider the next attribute which consists of $\{2, 3, 4, 5, 6, 7\}$. Similar to the previous attribute, the dataset will not be split. Thus, the dataset is not partitioned into core rules. However, when we study V, we see that V actually consists of two different subsets $V_1 \equiv \{< 0, 5 >, < 1, 6 >, < 2, 7 >\}$ and $V_2 \equiv \{< 5, 4 >, < 4, 3 >, < 3, 2 >\}$.

Therefore, partitioning a dataset using the gaps is not sufficient to capture the core rules. Fortunately, we have the following observations. If a dataset consists of some core rules, we can just partition the dataset into equal intervals. The core rules may be detected. Using the earlier example, when we see that there is no gap in the first attribute for $\{0, 1, 2, 3, 4, 5\}$, we just continue to split it into smaller equal intervals. In the example above, we will split the first attribute into two intervals, which will be $[0, 1, 2]$ and $[3, 4, 5]$. Similarly, we can apply the same method on the second attribute. By performing all possible combinations of intervals between two attributes, we will obtain four interval-tuples $< [0, 2], [2, 4] >$, $< [0, 2], [5, 7] >$,

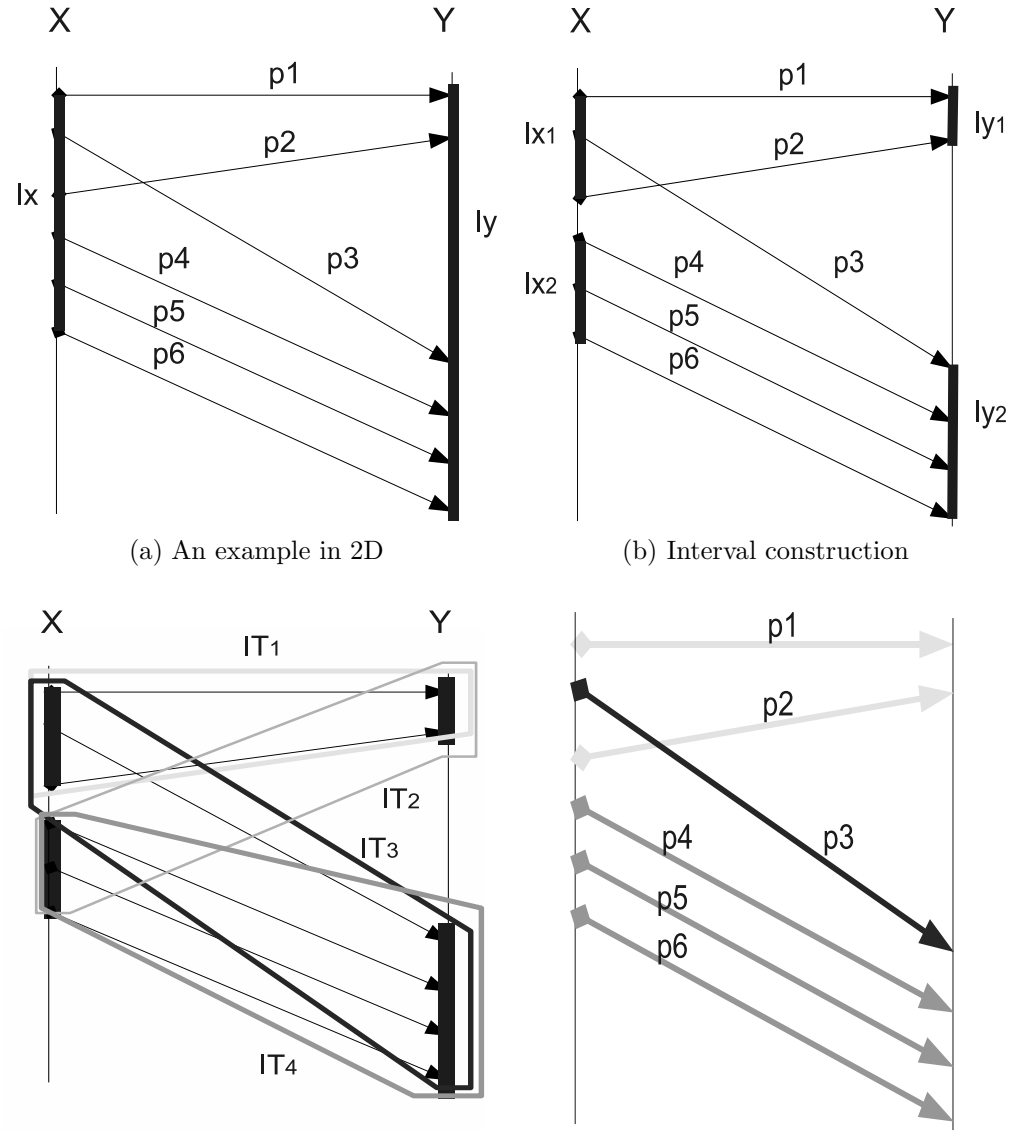
$< [3,5],[2,4] >$, $< [3,5],[5,7] >$. We then apply the interval selection operation σ_I on V using the four interval-tuples. As a result, it will return two core rules V_1 and V_2 correctly.

In the example above, we show that by partitioning intervals and applying the interval selection operation on the combinations of the intervals, we may be able to identify core rules. One challenge is how to partition an interval. We make the following two observations. If there is no core rule, the choice of a partition is not important. Because if a set is partitioned into two core rules, the merge operation always combines them into the same pattern. In other words, we are free to partition a core rule into smaller rules. For a dataset, we can partition a set multiple times until the core rules are detected using the method above.

We will use a split function f^S to denote the operation that generates core rules. To sum up, the split function on a set R divides each attribute into at least k intervals. Each interval is split into smaller intervals if the ratio of the difference between two consecutive ordered values from R on the same attribute in the interval is greater than α_s. The combinations of the intervals from different attributes produces the interval-tuples. We use $f^S(R/k, \alpha_s)$ to denote the set of interval-tuples $V \equiv f^S(R/k, \alpha_s)$ returned from the split function on R.

We then select observations from R using those interval-tuples. We create a set S of tuples from R where $S \equiv \sigma_I(R)$. If S does not satisfy the condition $|f_i^S(S/\alpha_s)| = 1, \forall i$, which means S can be divided into smaller sets. We then split an interval tuple I into smaller interval tuples by performing the split function $f^S(S/k = 1, \alpha_s)$ on S (the value of one for k means that we only split an interval when there is a change of the values in the interval). We say $\{S\}$ are the core rules generated from R.

Figure 29 illustrates an example of core rule construction. The example consists of six observations with two attributes X and Y. The observations are illustrated in Figure 29a. The values of observations on X and Y form two intervals I_x and I_y.



(a) An example in 2D

(b) Interval construction

(c) Interval-tuple construction

(d) Three identified core rules

Figure 29: An example of core rule construction

Since there is a gap in attribute Y, I_y is split into I_{y_1} and I_{y_2}. Since there is no gap in attribute X, I_x is split into two equal intervals I_{x_1} and I_{x_2}. The new intervals are shown in Figure 29b. By combining all the intervals in different ways, we obtains four interval-tuples IT_1, IT_2, IT_3, and IT_4 as shown in Figure 29c. By applying the tuple selection operation on the data using IT_1, IT_2, IT_3, and IT_4, we obtain three patterns $\{p_1, p_2\}$, $\{p_2\}$ and $\{p_3, p_4, p_6\}$.

As mentioned earlier, the core rules are the elements used to produce patterns. We use Definition 20 to produce patterns from the core rules by measuring the similarity between the core rules. We can also combine the patterns into larger patterns. Since we have already constructed the interval-tuples when combining the core rules, we can use the set of interval-tuples as a representation of a pattern. Therefore, we can compare the interval-tuples in order to combine patterns directly. We have the following definition for this:

Definition 23. *The o-score function between two interval tuples is defined by*

$$o\text{-}score(I, J) \equiv \sqrt{\sum \alpha^2(I_i, J_i)} \tag{13}$$

Definition 23 defines the score between two interval tuples. The score of a pattern against another pattern $o\text{-}score(P, P')$ is the smallest score between the two interval tuples of two patterns. As we see, the score can also be used to measure the deviation between patterns. If a pattern is small and it deviates from other patterns greatly, the pattern can be considered an unusual pattern candidate. Therefore, if we define the *u-score* of a pattern P to be the smallest score between it and the remaining patterns, then the *u-score* is the unusual score of pattern P.

6.8 Attribute Selection for Core Rule

In our framework, the core rules are produced based on the first selected attribute. However, a set of observations that is flagged as an anomalous pattern may be just a normal pattern. This can be shown in the following example.

```
1: procedure RuleBase(D)
2:     for all i ∈ n do
3:         U ← makeSamples(D, i)
4:         for all R ∈ U do
5:             V ← f^S(R/k, α_s)
6:             for all I ∈ L do
7:                 S ← makeRules(R, I)
8:                 put S into RP
9:             end for
10:            makePatterns(RP, NP, UP)
11:            for all p ∈ UP do
12:                compute o-score(p, NP)
13:            end for
14:            P ← matchNorm(NP, P)
15:            UP ← matchUnusual(UP, P)
16:            P ← combine(P, UP)
17:        end for
18:    end for
19: end procedure
```

Figure 30: Feature-based anomalous pattern detection

Let say after performing a split, there are three observations $< 1, 1 >, < 2, 2 >, < 3, 1000 >$. In this set, the third observation, $< 3, 1000 >$, has an unusual high value of 1000 for the second attribute. Therefore, $< 3, 1000 >$ is anomalous with respect to the split on the first attribute. However, let us assume that $< 3, 1000 >$ will be in the set of $< 3, 1000 >, < 4, 2000 >, < 5, 3000 >$ by performing the split on the second attribute. In this set, the differences between $< 3, 1000 >, < 4, 2000 >$ and between $< 4, 2000 >, < 5, 3000 >$ are the same for both the first and second attribute. As we see, $< 3, 1000 >$ is not anomalous by the split on the second attribute.

This example shows that if a set of observations is detected as anomalous in the core rule construction step in one attribute, they may not be anomalous on some other attributes. Therefore, we have to perform the core rule construction using all the attributes. Then, if an observation belongs to an anomalous pattern in one attribute and if it belongs to a normal pattern in some attributes, we will consider all the observations in the corresponding anomalous pattern normal.

6.9 Framework

6.9.1 Building Core Rules

In the previous sections, we have discussed how to identify core rules and how to combine the core rules into patterns. In this section, we will present the framework of our method. The outline for the framework is shown in Figure 30. Similar to the steps mentioned in section 6.7, first, we select an attribute i to construct core rules. We can treat the entire dataset D as an input. The dataset is sorted on attribute i. Since the dataset D is usually very large, we will divide D into chunks. As mentioned earlier, the result of the method is not affected by the split, therefore, we can choose the chunk size small enough to be processed efficiently. We will construct core rules from each chunk R.

We apply the split function f^S on R. The split function will output a set V of interval-tuples. We then apply the interval selection function σ_I on R to produce the core rules $\{S\}$.

In the next step, we combine all the core rules with the existing normal and anomalous patterns by calling the function makePatterns. It will produce two updated lists, one of normal patterns NP and the other of anomalous candidate patterns UP according to Definitions 20, 22, and 21. The anomalous score $o - score$ of the patterns in UP is computed against NP (lines 11 - 13). As a result, the algorithm from line 3 to line 13 produces the normal patterns and unusual pattern candidates for each chunk R.

6.9.2 Pruning Candidates

As mentioned in section 6.8, we need to prune the unusual pattern candidates. The algorithm consists of n rounds where n is the number of attributes. For each round, we perform the same steps to produce the patterns as mentioned above. In the i^{th} round, we will validate an anomalous candidate. For each candidate $p \in P$, p will

be removed from P if there exists a normal pattern $p_n \in NP$ such that p and p_n are close under α_c (line 14). However, for the first round, no action is performed on UP from line 15 to 16 and all the unusual pattern candidates are put into P. P contains the set of candidates for the first attribute. The items which do not belong to any pattern in this set of candidates are normal.

For the next rounds, the unusual pattern candidates will be removed from UP if they are not in P (line 15) since they were flagged as normal. The function at line 16 combines the unusual pattern candidates into larger patterns according to the close function under parameter α_c. The candidates which are normal after the combination are removed from P. The new unusual score of each new pattern is the lowest score from the candidates for which the new pattern is created. When the computation is finished for all rounds, all the unusual pattern candidates in P that do not match any normal pattern or do not have the support high enough are unusual patterns.

The close and score functions are computed from the α function which requires the dividends to be non zero. For each round, we replace the zeros with the average of the next c items when the dataset is sorted on attribute i. The zeros can be replaced by 0.5 if the dataset contains only 0 and 1 for attribute i.

6.9.3 Interval-Tuple Construction

When we produce the core rules, there are four steps. First, we need to compute the intervals. Then, the interval-tuples are constructed from all the possible combinations of intervals. Next, we apply the interval selection operation on the set using the interval-tuples. This step will produce the core rules. Then, we apply the ω function on the core rules to update the correct interval-tuple associated with a core rule. At this step, we also verify that the core rule does not contain different sets of observations. If it does, the core rule needs to be split again. It should be noted that it is challenging to produce these interval-tuples in high dimensional data. The

number of possible combinations is exponential with the number of attributes. However, we observe that the number of core rules as the result of the interval selection operation using those interval-tuples is not greater the number of tuples in the set used to construct the intervals. Therefore, we can implement an efficient algorithm to construct the interval-tuples as follow.

Let say we have the list of intervals $L_i = \{I_i^1, \ldots, I_i^k\}$ for each attribute i. Instead of combining all the tuples directly before applying the interval selection operation, we combine these two steps. For the first tuple in the provided dataset, we select an interval from each attribute such that this set of intervals cover the tuple. The set of intervals is an interval-tuple. We put the interval-tuple and its associated tuple into an interval-tuple list. For the next tuple, we will scan the interval-tuple if there is an interval-tuple that covers the tuple. If there is, we will put the tuple in the list corresponding to the found interval-tuple. Otherwise, we will construct a new interval-tuple that can cover the tuple using the intervals as shown earlier. If there is an interval-tuple that covers it, then the time complexity for this operation is based on the size of the interval-tuple list. If there is no such interval-tuple, then the time complexity for this operation is based on the number of intervals. Therefore, the time complexity for this step is the number of tuples multiplied with the maximum of the two following values: the size of the interval-tuple list and the number of intervals. As a result, we can construct the interval-tuples and their corresponding tuples efficiently. The sketch of the algorithm is shown in Figure 31.

6.10 Parameter Setting

The algorithm consists of five parameters: α_c, α_s, k, N_c, N_u, which can be determined in a straightforward manner. The user-defined parameter N_u indicates the size of a pattern which is to be considered unusual. The parameter, N_c, is the chunk size. The value of k is chosen based on the number of possible patterns in a chunk. The number

```
1: procedure ProduceInterval-Tuple(D, L_1 ... L_n, L)
2:     for all  dot ∈ D
3:         for all I ∈ L do
4:             if I covers t then
5:                 put t into σ_I
6:             else
7:                 for all i ∈ {1, ..., n} do
8:                     I ≡ new interval-tuple
9:                     for all J_i ∈ L_i do
10:                        if J_i covers t_i then
11:                            I_i ≡ J_i
12:                        end if
13:                    end for
14:                end for
15:                put I into L
16:                put t into σ_I
17:            end if
18:        end for
19:    end for
20: end procedure
```

Figure 31: Constructing interval-tuples

of patterns increases in the chunk when k increases. However, if those patterns are similar, they will eventually be combined together. The value of k does not impact the output significantly. Typically, k can be around 4 and N_c is $k \times N_u$. Besides k and α_s are used to split an interval if there is an abnormal change in the interval. The parameter α_c defines the cutoff point for the unusual observations. Heuristically, we can choose α_c and α_s between 0.3 and 0.6. The value of 0.3 says that the change of 30% is significant and an interval with 30% should be split into two intervals.

6.11 Running-time Complexity

We denote $|NP|$, $|UP|$ and $|P|$ as the total number of items in NP, UP and P respectively. For lines 5 to 9, it takes $O(N_c)$ time to make the rules. The makePattern function needs to combine all the patterns and has a worst case of $O(N_c^2)$. The running time from line 11 to 13 is $O(|UP| \times |NP|)$. We can use hash tables to store patterns

for NP, UP and P in order to execute the matching functions (line 14, 15) in linear time with respect to the number of items. The execution time for the two lines are $O(min(|NP|, |P|))$ and $O(min(|UP|, |P|))$ respectively. Since $|NP| + |UP| = N_c$, the execution time for them is less than $O(N_c)$. The running time for line 16 is $O(|UP| \times |P|)$. The total running time from line 3 to 16 is $O(N_c + N_c^2 + |UP| \times |NP| + N_c + |UP| \times |P|)$. Since N_c and $|UP| * |NP|$ are less than N_c^2, the formula can be reduced to $O(max(N_c^2, |UP| \times |P|))$. In general N_c^2 is small and does not grow with the dataset and $|UP| \times |P|$ is small on the average. Therefore, the running time can be considered constant, $O(c)$.

There are n rounds. It takes $O(NlogN)$ time to sort the data according to the attribute of the current round. Each round has $\frac{N}{N_c}$ chunks and the execution time for each chunk is $O(c)$. Therefore, the total execution time is $O(c \times n \times \frac{N}{N_c} + n \times NlogN)$. Since $(\frac{c}{N_c \times logN} + 1)$ is small and inversely proportional to N, we replace it by ς, the formula can be written as $O(\varsigma \times n \times N \times logN)$. The formula shows that the algorithm runs very fast.

6.12 Experiments

We ran experiments using the KDD CUP 99 Network Connections Data Set from the UCI repository [112]. The data was compiled from a wide variety of intrusions simulated in a military network environment prepared by MIT Lincoln Labs. Each record has 42 features containing the network connection information. Among them, 34 features are continuous and 8 features are symbolic. The last feature labels the type of connection. About 80% of the dataset are normal connections and 20% are attack connections. Because the rate of attack connections is very high for regular network activities, we follow the data generation approach in [42] to create a new dataset with a very low number of attacks randomly drawn from the KDD Cup Dataset to test whether the small patterns would be discarded by the sampling method in

Table 16: List of attacks

Type	#attacks	Type	#attacks	Type	#attacks
pod	210	teardrop	179	loadmodule	9
ipsweep	209	back	168	ftp write	8
warezclient	203	guess passwd	53	multihop	7
portsweep	200	bufferoverflow	30	phf	4
nmap	200	land	21	perl	3
smurf	200	warezmaster	20	spy	2
satan	197	imap	12		
neptune	196	rootkit	10		

a large dataset. The new dataset contains 95,174 normalized connections in total and the total number of attacks account for only 2.2% of the dataset. To make the experimental results less biased by the possible special characteristics of some types of attack, the dataset contains 22 types of attack with the number of records for each attack type varying from 2 to 210. The attack with the largest size accounts for only 0.2% of the dataset. The details of the number of connections for each attack are shown in Table 16.

6.12.0.1 Competing Methods

First, we run an oultier detection algorithm on the dataset to obtain the list of outliers. In this experiment, we use the well-known outlier detection method LOF [24] to compute outliers in the experiments. The local outlier factor of a point p is the ratio between the kdist(p) and the average of the kdists of all the k-nearest neighbors of p where kdist(p) is the distance from p to the k^{th} nearest neighbor of p. The records with high LOF scores are considered as outliers. In LOF, min_pts^{th} is the number of nearest neighbors to compute kdist.

Figure 32 shows the LOF precision/recall curve for different values of min_pts. Generally, $min\_pts = 100$ performs better than the other two values of min_pts. As we see, LOF obtains the highest precision rate (56%) when the recall rate = 2.6%. When the recall rate reaches 20%, the precision rate drops dramatically to 10%.

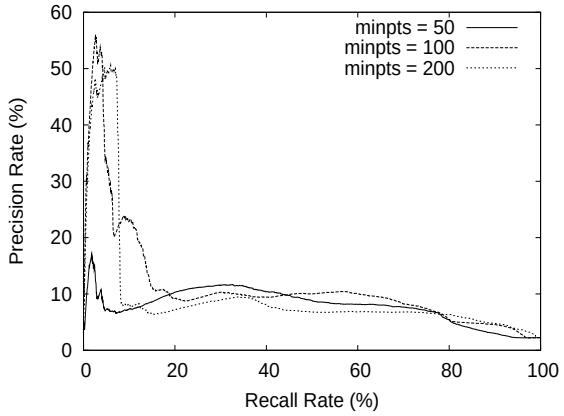


Figure 32: LOF with varied Min_pts

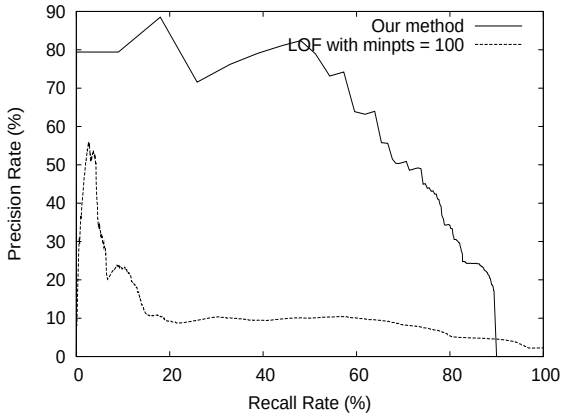


Figure 33: Feature-based method versus LOF

Table 17: Top outliers (10%) clustered by kmean

ID	**Rank**	**Size**	**Connections**	**Rate(%)**
1	7	159	neptune	95.0
			portsweep	1.3
2	12	114	satan	0.9
			teardrop	89.5
			ipsweep	0.9
3	13	110	nmap	100.0
4	16	107	smurf	73.8
5	18	101	satan	1.0
			ipsweep	4.0
			portsweep	87.1

Table 18: Top outliers (10%) clustered by SNN

ID	Rank	Size	Connections	Rate(%)
1	9	78	neptune	100.0
2	15	58	smurf	77.6
3	16	56	smurf	100.0

Table 19: The results of clustering using SNN on the entire dataset

ID	Rank	Size	Connections	Rate(%)
1	35	112	nmap	100.0
2	57	89	warezclient	73.0
			rootkit	1.1
3	74	79	neptune	100.0
4	90	65	ipsweep	100.0
5	112	59	smurf	78.0
6	118	58	smurf	100.0

Since the precision is low for a high recall rate, we apply different clustering algorithms on the outliers to group them into their corresponding attack types. First, we ran kmean on the top outliers which account for 10% of the dataset. There are 40 clusters. We then filter the output by removing clusters with the size greater than 250. There are 30 such clusters. Ten of them have at least 50% attacks. Table 17 shows the corresponding rank of the first five clusters that contain the attack connections. As we can see, 13 out of 18 top clusters do not contain any attack.

In the next experiment, we used a shared nearest neighbor (SNN) clustering algorithm to cluster the top outliers. SNN returns one cluster of size 3814 and 292 clusters of size less than 143. There are 19 clusters with the size from 50 to 143. Table 18 shows the top three clusters that contains the attack connections. The first attack cluster ranks 9^{th} and its size is 78. In those two experiments, the attack connections are not shown as strong unusual patterns.

The clustering of the top outliers could not detect all the top ten groups of attacks. Because of this, we want to cluster the entire dataset with the hope that those groups can be discovered. It is difficult to set the parameter k for kmean in this large

dataset. With large k, kmean will return many clusters whereas small values of k may group the normal connections with attack connections. Therefore, we used SNN for this experiment since it can discover small clusters with different densities in large datasets [52] without requiring the number of clusters as input. The algorithm returns 131 clusters with the size from 50 to 250. Among them, there are six clusters that contain attack connections (see Table 19). The first cluster that contains the attack connections is ranked 35^{th}. As we can see, the attack and normal connections are divided into smaller clusters. The attack patterns are not shown clearly in this experiment. The result shows that even though SNN can discover small clusters, they will return many of them.

6.12.1 Our Method

In the next experiment, we ran our method on the dataset to discover the unusual patterns. We set the parameters according to the parameter setting from the algorithm section ($N_u = 250$, $\alpha_c = 0.5$, $k = 4$, $\alpha_s = 0.5$, and $N_c = 1200$). The algorithm returns 20 unusual patterns with the size of at least 50. Table 20 shows the top 10 unusual patterns by size. The size of those patterns varies from 67 to 243. The first two patterns are of the attack type. Seventy percent of those patterns contain 100% attacks. The other 9 smaller patterns containing attacks are shown in Table 21. According to Table 20, we see that satan, neptune and portsweep follow strong patterns. In the table, we see that warezclient attacks also follow a pattern but its score is low (6.5) relative to those attack types, which means that its pattern is slightly different from normal patterns.

As mentioned above, the dataset contains 10 attack patterns with a size of at least 100. In our method, eight of ten are identified in the top unusual patterns (by size).

Table 23 shows the recall rate for each attack type found in Table 20. With the low false alarm rate, we still get a high recall rate.

Table 20: Top 10 unusual patterns ordered by size

Rank	Size	Score	Types	Rate(%)
1	243	24.1	smurf	79.4
			normal	20.6
2	192	43.6	nmap	100
3	170	12.9	normal	100
4	169	203.5	satan	100
5	150	104.1	neptune	100
6	129	26.1	ipsweep	100
7	114	14.2	back	100
8	84	6.5	warezclient	100
9	72	15.6	normal	100
10	67	107.7	portsweep	100

We then take all the unusual patterns with the size of at least 50 and order them by score. Table 22 shows the ranking, score and attack type of the patterns. According to the table, our method correctly identifies some of the attacks in the first six unusual patterns. All of these patterns have the detection rate of 100%. Among them, the attack type of Satan has the highest score which is 203.5. The score of the first normal connection pattern in the table is only 29.6 and its size is only 54.

In ranking either by size or by score (after the patterns with very low sizes are pruned), we can see that the attack types of satan, portsweep, neptune, and nmap are discovered by our approach. The results imply that these types of attack follow some patterns which are strongly different from normal connections.

Figure 33 shows the PR curve of our method versus LOF. The figure shows that

Table 21: Other unusual patterns ordered by size

Rank	Size	Score	Types	Rate(%)
13	66	11.8	teardrop	100
14	64	54.	pod	100
20	50	48.8	pod	100
22	48	83.1	guesspwd	100
23	44	156.6	neptune	100
33	31	20.0	teardrop	100
35	29	30.0	back	100

Table 22: Unusual patterns ordered by score

Rank	Type	Score	Size	Rank	Type	Score	Size
1	satan	203.5	169	11	smurf	24.1	243
2	portsweep	107.7	67	12	normal	15.6	72
3	neptune	104.1	150	13	normal	14.9	64
4	pod	54.6	64	14	back	14.2	114
5	pod	48.8	50	15	normal	12.87	170
6	nmap	43.6	192	16	teardrop	11.8	66
7	normal	29.6	54	17	normal	10.5	67
8	normal	26.2	63	18	normal	9.7	56
9	ipsweep	26.1	127	19	warezclient	6.6	84
10	normal	25.9	59	20	normal	4.2	67

Table 23: The recall rate of the attack types in the top 10 patterns

Type	Rate	Type	Rate
smurf	96.5	back	67.9
nmap	96	ipsweep	61.2
satan	85.8	warezclient	41.4
neptune	76.5	portsweep	33.5

our method yields a precision rate of 80% at very low recall rates. The method outperforms the LOF approach. Since we quickly prune the patterns with an unusual score below the cutoff threshold, the attacks with a very low unusual score are removed from the output. That is why the figure only shows the recall rate up to 90%. However, this does not effect the result much since the precision rate is usually low at this recall rate.

According to the experiments, the following interesting observations are found. Most unusual patterns with normal connections do not form highly supported patterns. Even though they may have high scores, they appeared as very low support patterns, whereas the attack connections form patterns with high support. The normal connections that can form patterns with high support tend to have a very low score. Few discovered unusual patterns have mixed results. Another important observation is that the number of unusual patterns with high support is very small.

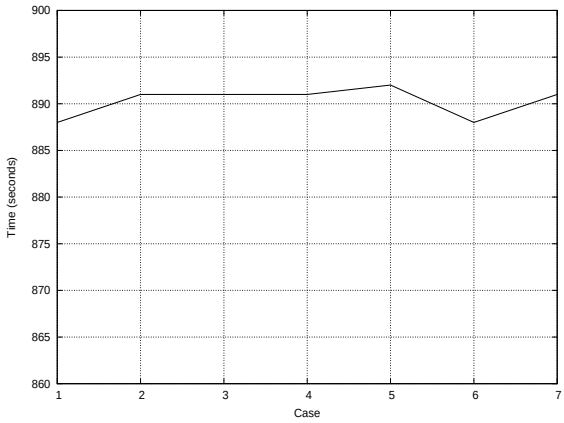


Figure 34: Running time with different orders of attributes

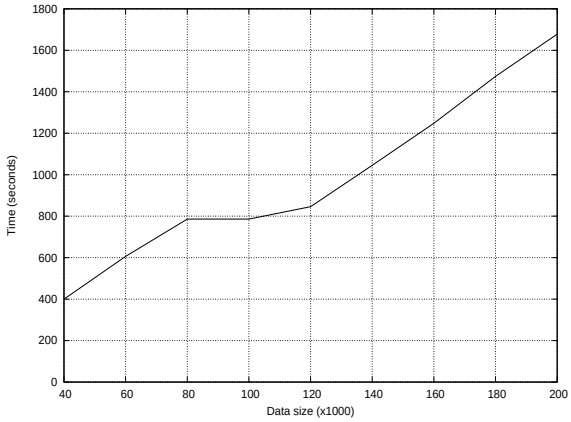


Figure 35: Running time of the feature-based method

In conclusion, we have performed a variety of experiments with different combinations of outlier detection and density-based clustering algorithms. The precision is low when the recall rate increases to 20%. For the clustering algorithms, the normal connections were also grouped into small clusters. However, our algorithm grouped the attack and normal connections almost correctly according to their connection type even though there were 22 types of attacks of small size. They are shown clearly as unusual patterns in terms of size and score.

6.13 Performance

We implement a memory-based version of the algorithm in Java. For each round, the dataset can be sorted in $O(NlogN)$ time. We use a hash table data structure to store the list of unusual candidate items so that the matching functions, matchNorm and matchUnusual, can be performed in constant time. At first, we ran the program with different initial attributes. The number of unusual candidates vary from 30K to 60K. During the first round, we do not combine the unusual patterns, therefore, the performance is not affected. In the next few rounds, the matching functions reduce the number of unusual items dramatically before the combining step.

Figure 34 shows the performance of the algorithm with different random orders of the attributes. According to the figure, we see that the order of the attributes does not affect the running time significantly. Also, the results are almost the same for different orders of the attributes. The attack connections are consistently in the top unusual patterns.

In the next experiment, we ran the program on the KDD dataset with the size varying from 40K to 200K. Figure 35 shows that the execution time of the program is linear with the growth of the data size. By replacing the memory-based hash table data structure and merge sort algorithm with the disk-based versions, the algorithm can be used for any large dataset.

6.14 Conclusion

We have introduced a fast method that can discover unusual observations by generating core rules from the feature-based chunks. A core rule is a set of strongly related observations. The rules are then combined into patterns by using the α function. When a pattern gains high support, it is considered normal. Otherwise, we will try to merge the pattern with other patterns. If they still can not gain enough support. It is considered unusual. The unusual patterns are ranked based on their support. In the experiments, we have compared our algorithm with other possible alternatives, namely outlier detection and clustering, to discover unusual patterns. According to the results, our approach yields the highest detection rate with the most unusual items grouped into the correct corresponding patterns. We have also introduced the score function to measure the degree of deviation of the unusual patterns from the normal patterns. The running time complexity of the method is $O(\varsigma \times n \times N \times logN)$. The experiments confirm that our method can run very fast in large datasets.

CHAPTER VII

CONCLUSION

This thesis shows that the efficiency and accuracy can be improved for both individual outlier and anomalous pattern detection. Among outlier detection methods using the distance approach, we focus on the local density-based outliers. The method requires the computation of k-nearest neighbors in order to compute the local outlier factors for the dataset. This is challenging in high dimensional data. In our thesis, we have shown that the full k-nearest neighbors are not necessary in order to identify local density-based outliers. Indeed, a local outlier is also dominant in any subset of the dataset. We present a randomization method to compute the local density-based outlier very fast without finding global k-nearest neighbors. We randomly select two representative points to partition a set into two subsets. It is performed recursively until the computation cost is small. The outliers in the subsets are outlier candidates. With multiple randomizations, the true outliers will remain. We also introduced a hybrid version of the randomization which performs global nearest neighbor recomputations to improve the accuracy of the method. The experiments on a variety of datasets have shown that the the randomized method and the original LOF have similar results. The experiments show that the randomization is scalable with high dimensional and large data sets.

In multiple dimensional data, an outlier may deviate only in a subset of the attributes. When the number of dimensions is large, the delta deviation in other attributes can show significant accumulation such that the deviation in the attributes where the outliers are strong are overlooked. The large number of dimensions suppresses these outliers. In our thesis, we have shown that the problem can be solved.

Previously, the outlier factors are computed using the full feature set. Instead, we perform the projections of the data onto each dimension. The sub-dimensional score is the projected deviation of an observation on the dimension. We filter the attributes with very low sub-dimensional score. The remaining scores are aggregated into the final score. This technique reduces the effect of the large number of dimensions on the outlier. Using normalization within the subset, we also solve the problem of a mixture of variances. As a result, the outliers can be ranked more accurately. The sub-dimensional approach allows us to visualize the outliers by drawing the graphs in the dimensions where the points deviate from others. It is helpful for explaining why the outliers are interesting. We also apply the clustering technique from [109] to cluster the outliers by observing that two points whose *oscore* between them is zero are close to each other and should be in the same cluster. The experiments have shown that the outliers can be detected more accurately.

We have shown the improvement of the accuracy of outlier detection. Practically, the outlier detection methods have very high false alarm rates. In many cases, the majority of outliers are not useful or not easy to understand. Therefore, in this thesis, we proposed the concept of anomalous patterns where the outliers follow certain patterns. The anomalous patterns imply that the deviation may not just be random. In our experiments, we have shown that top outliers are not always interesting. We have presented the adaptive dual-neighbor method to detect anomalous patterns based on the size and deviation. The granularity level can be adjusted. Our experiments show that our algorithm can discover interesting patterns which are undetected by clustering or outlier detection methods.

In some cases, an anomalous pattern can be defined according to the correlation between the attributes. A pattern is a set of observations that are strongly correlated. We have developed a feature-based method to compute the patterns efficiently. We partition the data into chunks of correlated observations and learn the patterns

within the chunks. During the matching and merging between patterns, we prune the patterns with high support. The remaining patterns after multiple merges are anomalous. In the experiments, we have shown that the approach yields the highest detection rate with the malicious observations grouped into the correct corresponding patterns. The experiments confirm that the feature-based method can run very fast in large datasets.

In conclusion, we have developed several methods to improve the accuracy and efficiency of outlier detection. We also proposed the concept of an anomalous pattern to reduce the false alarm rate. We have presented filtering and the adaptive method to improve the accuracy. In addition, it is shown that randomization and feature-based methods can detect outliers or anomalous patterns efficiently.

CHAPTER VIII

FUTURE WORK

In Chapters 3 and 6, we introduced efficient methods to detect outliers and anomalous patterns in large and high dimensional datasets. A natural extension of our methods is to implement parallel versions of those methods.

8.1 Randomized Algorithm

The randomization algorithm mentioned in Chapter 3 computes candidate outliers using local subspaces. This provides a possibility of parallelization to speed up the computation when a large computer cluster is available. Let us say we have a dataset that is partitioned and distributed in a computer cluster. A data node is a node that contains a partition of the data set. Let us recall that the randomized method has two main steps. The first step partitions the dataset to compute the candidate scores. The second step is to combine the candidate scores. In the first step, we randomly select k pivot points from the dataset, so as to to divide the dataset into k partitions where parameter k is the number of nodes in the cluster. Each pivot point is assigned to a node in the cluster. Each data node clusters its local data partition using the pivot points as the cluster centroids. The computed clusters are then sent to the computer nodes whose assigned pivot points are the same as their cluster centroids. As a result, a node receives all the points in the data set that is closest to its assigned pivot point. After that, each node will compute the candidate outlier scores for the points in the partition using the randomized algorithm. As we can see, this entire process corresponding to an iteration of the randomized algorithm. We can run that process multiple times to obtain different candidate scores. In the second step, a point is assigned to a cluster node. The candidate scores of a point are sent to the

assigned node. The node will combine the scores for the collected points and output the final scores.

As we see, the entire process can be performed in parallel. When the dataset is distributed equally, the performance should increase proportionally with the number of nodes in a cluster. However, there is a possibility of data skew where some nodes receive a majority of the dataset. In such cases, the performance will not increase proportionally. We need to develop a robust solution that handles this case.

In future work, we will design and implement a parallel algorithm and evaluate the performance and the robustness of the algorithm.

8.2 Anomalous Pattern Detection Using Correlated Attributes

In Chapter 6, we presented a framework to detect anomalous patterns. In our framework, we introduced the alpha function to combine the items into patterns. However, the function can not be used to combine categorical attributes. One future work is to modify the alpha function so that it can also combine the items based on both categorical and numerical data.

Another future work is to parallelize our method. The first step of our original method is to sort the dataset based on individual attributes. The dataset can be sorted using an existing parallel sort algorithm. The second step of the method is to learn the core rules in each chunk. For this step, each sorted dataset will be partitioned into chunks. The chunks will be sent to the cluster nodes. The cluster nodes will learn patterns from those chunks. As we can see, this framework works well for the first two steps of our existing method. However, the challenge lies in the combine/merge step of our algorithm. In the combine/merge step, we need to verify if an anomalous candidate can be combined with any other non-candidate patterns. This is equivalent with joining sets. In our method, we used a hash table for joining sets. This process is sequential for all the items. Therefore, the future work is to

revise the combine/merge set so that it can be computed in parallel.

8.3 Complexity in Data Model

Even though there are many different types of anomaly or outlier detection in the field, the existing methods are based on simple data models. The data models may not be comprehensible enough for accurate detection. The tabular model considers sets of records. However, it does not consider the complex relationship and the order between the records. The sequential model considers the order but it only considers sequences of simple values. The graph model considers the relationship but the graph nodes do not carry complex attributes. For an example, in a computer network, multiple connections to the same port are not the same as multiple connections to different ports. This information should be taken into account in analyzing network's activities.

Therefore, in future work, the data models in anomaly detection should handle more complex data types so that more sophisticated anomalies can be detected. The data models should consider the relationship and natural order between entities, the temporal information, and complex attributes. In addition, the anomaly detection methods based on these data models should be very efficient so as to be useful in practice.

REFERENCES

[1] Agarwal, D., "Detecting anomalies in cross-classified streams: a bayesian approach," *Knowl. Inf. Syst.*, vol. 11, no. 1, pp. 29–44, 2006.

[2] Aggarwal, C. C. and Yu, P. S., "Finding generalized projected clusters in high dimensional spaces," in *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 70–81, ACM, 2000.

[3] Aggarwal, C. C. and Yu, P. S., "Outlier detection for high dimensional data," in *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 37–46, ACM, 2001.

[4] Agrawal, R., Ailamaki, A., Bernstein, P. A., Brewer, E. A., Carey, M. J., Chaudhuri, S., Doan, A., Florescu, D., Franklin, M. J., Garcia-Molina, H., Gehrke, J., Gruenwald, L., Haas, L. M., Halevy, A. Y., Hellerstein, J. M., Ioannidis, Y. E., Korth, H. F., Kossmann, D., Madden, S., Magoulas, R., Ooi, B. C., O'Reilly, T., Ramakrishnan, R., Sarawagi, S., Stonebraker, M., Szalay, A. S., and Weikum, G., "The claremont report on database research," *SIGMOD Rec.*, vol. 37, no. 3, pp. 9–19, 2008.

[5] Agrawal, R., Imieliński, T., and Swami, A., "Mining association rules between sets of items in large databases," in *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 207–216, ACM, 1993.

[6] Aguilera, M. K., Golab, W., and Shah, M. A., "A practical scalable distributed b-tree," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 598–609, 2008.

[7] Akaike, H., "Information theory and an extension of the maximum likelihood principle," *2nd International Symposium on Information Theory*, pp. 267–281, 1973.

[8] Akdere, M., Çetintemel, U., and Tatbul, N., "Plan-based complex event detection across distributed sources," *PVLDB*, vol. 1, no. 1, pp. 66–77, 2008.

[9] Alexander Strehl and Joydeep Ghosh, "Relationship-based clustering and visualization for high-dimensional data mining," *INFORMS journal on computing*, pp. 208–239, 2003.

[10] Anagnostopoulos, A., Kumar, R., and Mahdian, M., "Influence and correlation in social networks," in *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 7–15, ACM, 2008.

[11] Arindam Banerjee, Sugato Basu, and Merugu Srujana, "Multi-way Clustering on Relation Graphs," in *Proceedings of the seventh SIAM International Conference on Data Mining*, Society for Industrial Mathematics, Jan. 2007.

[12] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y., "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[13] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y., "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[14] Au, W.-H., Chan, K. C. C., Wong, A. K. C., and Wang, Y., "Attribute clustering for grouping, selection, and classification of gene expression data," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 2, no. 2, pp. 83–101, 2005.

[15] Ayres, J., Flannick, J., Gehrke, J., and Yiu, T., "Sequential pattern mining using a bitmap representation," in *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 429–435, ACM, 2002.

[16] Bay, S. D. and Schwabacher, M., "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 29–38, ACM, 2003.

[17] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B., "The r*-tree: an efficient and robust access method for points and rectangles," *SIGMOD Rec.*, vol. 19, no. 2, pp. 322–331, 1990.

[18] Berchtold, S., Keim, D. A., and Kriegel, H.-P., "The x-tree: An index structure for high-dimensional data," in *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, (San Francisco, CA, USA), pp. 28–39, Morgan Kaufmann Publishers Inc., 1996.

[19] Bertino, E., Kamra, A., and Early, J. P., "Profiling database application to detect sql injection attacks," *Performance, Computing, and Communications Conference, 2002. 21st IEEE International*, vol. 0, pp. 449–458, 2007.

[20] Beyene, Y., Faloutsos, M., Duen Horng Chau, and Faloutsos, C., "The eBay graph: How do online auction users interact?," *INFOCOM Workshops 2008*, pp. 1–6, Apr. 2008.

[21] Beyer, K. S., Goldstein, J., Ramakrishnan, R., and Shaft, U., "When is "nearest neighbor" meaningful?," in *ICDT '99: Proceeding of the 7th International Conference on Database Theory*, (London, UK), pp. 217–235, Springer-Verlag, 1999.

[22] Böhm, C., Faloutsos, C., and Plant, C., "Outlier-robust clustering using independent components," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 185–198, ACM, 2008.

[23] Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., and Kriegel, H.-P., "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. 1, pp. 47–56, 2005.

[24] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J., "LOF: identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, 2000.

[25] Bu, Y., Chen, L., Fu, A. W.-C., and Liu, D., "Efficient anomaly monitoring over moving object trajectory streams," in *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 159–168, ACM, 2009.

[26] Chakrabarti, D., "Autopart: parameter-free graph partitioning and outlier detection," in *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, (New York, NY, USA), pp. 112–124, Springer-Verlag New York, Inc., 2004.

[27] Chan, P. K. and Mahoney, M. V., "Modeling multiple time series for anomaly detection," in *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, (Washington, DC, USA), pp. 90–97, IEEE Computer Society, 2005.

[28] Chandola, V., Banerjee, A., and Kumar, V., "Outlier detection: A survey," *ACM Computing Surveys*, pp. 1–72, Sept. 2009.

[29] Chandola, V., Mithal, V., and Kumar, V., "Comparative Evaluation of Anomaly Detection Techniques for Sequence Data," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), IEEE Computer Society, Dec. 2008.

[30] Charles Young et al., "KDD Cup 2004: Quantum physics dataset," 2004.

[31] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu, "On High Dimensional Projected Clustering of Data Streams," *Data Mining and Knowledge Discovery*, vol. 10, pp. 251–273, Mar. 2005.

[32] Chawla, N. V., Lazarevic, A., Hall, L. O., and Bowyer, K. W., *SMOTEBoost: Improving prediction of the minority class in boosting*, vol. 2838/2003 of *Lecture Notes in Computer Science*. Germany: Springer Berlin / Heidelberg, 2004.

[33] Chen, C., Yan, X., Yu, P. S., Han, J., Zhang, D.-Q., and Gu, X., "Towards graph containment search and indexing," in *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pp. 926–937, VLDB Endowment, 2007.

[34] Chen, C., Yan, X., Zhu, F., Han, J., and Yu, P. S., "Graph OLAP: Towards Online Analytical Processing on Graphs," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), pp. 103–112, IEEE Computer Society, Dec. 2008.

[35] Cheng, T., Yan, X., and Chang, K. C.-C., "Entityrank: searching entities directly and holistically," in *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pp. 387–398, VLDB Endowment, 2007.

[36] Christen, P., "Automatic record linkage using seeded nearest neighbour and support vector machine classification," in *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 151–159, ACM, 2008.

[37] Christopher Krugel, Thomas Toth, and Engin Kirda, "Service Specific Anomaly Detection for Network Intrusion Detection," in *ACM SAC*, 2002.

[38] Cook, D. J. and Holder, L. B., "Graph-based data mining," *IEEE Intelligent Systems*, vol. 15, no. 2, pp. 32–41, 2000.

[39] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H. A., Puz, N., Weaver, D., and Yerneni, R., "Pnuts: Yahoo!'s hosted data serving platform," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1277–1288, 2008.

[40] Dan Pelleg and Andrew Moore, "Active Learning for Anomaly and Rare-Category Detection," in *Proc. 18th Annual Conference on Neural Information Processing Systems*, Dec. 2004.

[41] Dana Ron, Yoram Singer, and Naftali Tishby, "The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length," *Machine Learning*, vol. 25, pp. 117–149, Nov. 1996.

[42] Das, K. and Schneider, J., "Detecting anomalous records in categorical datasets," in *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 220–229, ACM, 2007.

[43] Das, K., Schneider, J., and Neill, D. B., "Anomaly pattern detection in categorical datasets," in *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 169–176, ACM, 2008.

[44] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S., "Locality-sensitive hashing scheme based on p-stable distributions," in *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, (New York, NY, USA), pp. 253–262, ACM, 2004.

[45] Decandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Hastorun, D., Decandia, G., and Vogels, W., "Dynamo: Amazon's highly available key-value store," *SOSP*, 2007.

[46] Deng, H., King, I., and Lyu, "Formal Models for Expert Finding on DBLP Bibliography Data," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), pp. 163–172, IEEE Computer Society, Dec. 2008.

[47] Doan, A. and Halevy, A. Y., "Semantic integration research in the database community: A brief survey," *AI Magazine*, vol. 26, no. 1, pp. 83–94, 2005.

[48] Duen Hong Chau, Shashank Pandit, and Christos Faloutsos, "Detecting Fraudulent Personalities in Networks of Online Auctioneers," in *10th European Conference on Principles and Practice of Knowledge Discovery in Databases* (Fürnkranz, J., Scheffer, T., and Spiliopoulou, M., eds.), Lecture Notes in Computer Science, Springer, 2006.

[49] Eberle, W. and Holder, L., "Anomaly detection in data represented as graphs," *Intell. Data Anal.*, vol. 11, no. 6, pp. 663–689, 2007.

[50] Eberle, W. and Holder, L., "Discovering structural anomalies in graph-based data," in *ICDMW '07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, (Washington, DC, USA), pp. 393–398, IEEE Computer Society, 2007.

[51] Edwin M. Knorr and Raymond T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, (San Francisco, CA, USA), pp. 392–403, Morgan Kaufmann Publishers Inc., 1998.

[52] Ertöz, L., Steinbach, M., and Kumar, V., "Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data," in *Proceedings of the third SIAM international conference on data mining*, pp. 47–58, Society for Industrial and Applied, 2003.

[53] Ester, M., peter Kriegel, H., S, J., and Xu, X., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the second international conference on knowledge discovering and data mining*, pp. 226–231, AAAI Press, 1996.

[54] Fan, H., Zaïane, O. R., Foss, A., and Wu, J., "A nonparametric outlier detection for effectively discovering top-n outliers from engineering data," in *PAKDD*, pp. 557–566, 2006.

[55] Filippone, M., Camastra, F., Masulli, F., and Rovetta, S., "A survey of kernel and spectral methods for clustering," *Pattern Recogn.*, vol. 41, no. 1, pp. 176–190, 2008.

[56] Frank E. Grubbs, "Procedures for Detecting Outlying Observations in Samples," *Technometrics*, vol. 11, pp. 1–21, Feb. 1969.

[57] Freidman, J. H., Bentley, J. L., and Finkel, R. A., "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[58] Fujimaki, R., "Anomaly Detection Support Vector Machine and Its Application to Fault Diagnosis," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), pp. 797–802, IEEE Computer Society, Dec. 2008.

[59] G. S. Lingappaiah, "Effect of outliers on the estimation of parameters," *Metrika*, vol. 23, pp. 27–30, Dec. 1976.

[60] Genshiro Kitagawa, "On the Use of AIC for the Detection of Outliers," *Technometrics*, vol. 21, pp. 193–199, May 1979.

[61] Getoor, L. and Diehl, C. P., "Link mining: a survey," *SIGKDD Explor. Newsl.*, vol. 7, no. 2, pp. 3–12, 2005.

[62] Giudici, P. and Figini, S., *Applied Data Mining for Business and Industry*. Wiley Publishing, 2009.

[63] Gwadera, R. and Crestani, F., "Discovering Significant Patterns in Multistream Sequences," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), pp. 827–832, IEEE Computer Society, Dec. 2008.

[64] Haizheng Zhang, Marc Smith, C. Lee Giles, John Yen, and Henry Foley, "SNAKDD 2008 Social Network Mining and Analysis PostWorkshop Report," *SIGKDD Newsletter*, vol. 10, p. 74, Dec. 2008.

[65] Hawkins, D., *Identification of outliers*. London: Chapman and Hall, 1980.

[66] He, J., Liu, Y., and Lawrence, R., "Graph-Based Rare Category Detection," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), IEEE Computer Society, Dec. 2008.

[67] Helmer, G. G., Wong, J. S. K., Honavar, V., , Miller, L., and Miller, L., "Intelligent agents for intrusion detection," in *In Proceedings, IEEE Information Technology Conference*, pp. 121–124, 1998.

[68] Hido, S., Tsuboi, Y., and Kashima, "Inlier-Based Outlier Detection via Direct Density Ratio Estimation," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), IEEE Computer Society, Dec. 2008.

[69] Holder, L. B. and Yan, X., "Report on the first international workshop on mining graphs and complex structures (mgcs'07)," *SIGMOD Rec.*, vol. 37, no. 1, pp. 53–55, 2008.

[70] Höppner, F., "Discovery of temporal patterns. learning rules about the qualitative behaviour of time series," in *PKDD '01: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, (London, UK), pp. 192–203, Springer-Verlag, 2001.

[71] Huang, S.-M., Yen, D. C., Yang, L.-W., and Hua, J.-S., "An investigation of zipf's law for fraud detection (dss#06-10-1826r(2))," *Decis. Support Syst.*, vol. 46, no. 1, pp. 70–83, 2008.

[72] Huang, Y., Feamster, N., Lakhina, A., and Xu, J. J., "Diagnosing network disruptions with network-wide analysis," in *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 61–72, ACM, 2007.

[73] Huber, P. J., *Robust Statistics*. Wiley, 1981.

[74] Ide, T., Papadimitriou, S., and Vlachos, M., "Computing Correlation Anomaly Scores Using Stochastic Nearest Neighbors," in *Data Mining, 2007. ICDM '07. Seventh IEEE International Conference on* (Naren Ramakrishnan, Osmar R. Zaïane, Yong Shi, Christopher W. Clifton, and Xindong Wu, eds.), pp. 523–528, IEEE Computer Society, 2007.

[75] Jarvis, R. A. and Patrick, E. A., "Clustering using a similarity measure based on shared near neighbors," *IEEE Transactions on Computers*, vol. C-22, no. 11, pp. 1025– 1034, 1973.

[76] Jiong Yang and Wei Wang, "CLUSEQ: Efficient and Effective Sequence Clustering," in *Proceedings of the international conference on data engineering*, IEEE Computer Society Press, 1998.

[77] Jon M. Kleinberg, Ravi Kumar, and Prabhakar Raghavan, "The Web as a Graph: Measurements, Models, and Methods," in *Computing and Combinatorics: Proceedings of the 5th Annual International Conference COCOON'99* (Asano, T., Imai, H., Lee, D., Nakano, S.-i., and Tokuyama, T., eds.), vol. 1627 of *Lecture Notes in Computer Science*, pp. 1–17, Springer Berlin / Heidelberg, 1 Jan. 1999.

[78] Kam, P.-s. and Fu, A. W.-C., "Discovering temporal patterns for interval-based events," in *DaWaK 2000: Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, (London, UK), pp. 317–326, Springer-Verlag, 2000.

[79] Kamath, C., *Scientific Data Mining: A Practical Perspective*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009.

[80] Kamra, A., Terzi, E., and Bertino, E., "Detecting anomalous access patterns in relational databases," *The VLDB Journal*, vol. 17, no. 5, pp. 1063–1077, 2008.

[81] Kang, J. M., Shekhar, S., Wennen, C., and Novak, P., "Discovering Flow Anomalies: A SWEET Approach," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), pp. 851–856, IEEE Computer Society, Dec. 2008.

[82] Ke, Y., Cheng, J., and Ng, W., "Mining quantitative correlated patterns using an information-theoretic approach," in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 227–236, ACM, 2006.

[83] Ke, Y., Cheng, J., and Ng, W., "Correlation search in graph databases," in *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 390–399, ACM, 2007.

[84] Keogh, E., Lin, J., and Fu, A., "Hot sax: Efficiently finding the most unusual time series subsequence," in *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, (Washington, DC, USA), pp. 226–233, IEEE Computer Society, 2005.

[85] Khalid, S., "Motion-based behaviour learning, profiling and classification in the presence of anomalies," *Pattern Recogn.*, vol. 43, no. 1, pp. 173–186, 2010.

[86] Kim, H.-C. and Ghahramani, Z., "Outlier robust gaussian process classification," in *SSPR & SPR '08: Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, (Berlin, Heidelberg), pp. 896–905, Springer-Verlag, 2008.

[87] Koopman, S. J. and Ooms, M., "Forecasting daily time series using periodic unobserved components time series models," *Comput. Stat. Data Anal.*, vol. 51, no. 2, pp. 885–903, 2006.

[88] Korn, F., Pagel, B.-U., and Faloutsos, C., "On the 'dimensionality curse' and the 'self-similarity blessing'," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 96–111, 2001.

[89] Kriegel, H.-P., hubert, M. S., and Zimek, A., "Angle-based outlier detection in high-dimensional data," in *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 444–452, ACM, 2008.

[90] Kriegel, H.-P., Kröger, P., and Zimek, A., "Detecting clusters in moderate-to-high dimensional data: subspace clustering, pattern-based clustering, and correlation clustering," *PVLDB*, vol. 1, no. 2, pp. 1528–1529, 2008.

[91] Lachmann, A. and Riedewald, M., "Finding relevant patterns in bursty sequences," *PVLDB*, vol. 1, no. 1, pp. 78–89, 2008.

[92] Lakhina, A., Crovella, M., and Diot, C., "Characterization of network-wide anomalies in traffic flows," in *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, (New York, NY, USA), pp. 201–206, ACM, 2004.

[93] Lam, W. W. M. and Chan, K. C. C., "A graph mining algorithm for classifying chemical compounds," in *BIBM '08: Proceedings of the 2008 IEEE International Conference on Bioinformatics and Biomedicine*, (Washington, DC, USA), pp. 321–324, IEEE Computer Society, 2008.

[94] Laurikkala, J., Juhola, M., and Kentala, E., "Informal identification of outliers in medical data," in *The Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, 2000.

[95] Lazarevic, A. and Kumar, V., "Feature bagging for outlier detection," in *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, (New York, NY, USA), pp. 157–166, ACM, 2005.

[96] Lee, W. and Xiang, D., "Information-theoretic measures for anomaly detection," in *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, (Washington, DC, USA), p. 130, IEEE Computer Society, 2001.

[97] Li, X. and Han, J., "Mining approximate top-k subspace anomalies in multi-dimensional time-series data," in *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pp. 447–458, VLDB Endowment, 2007.

[98] Lin, J., Keogh, E., Lonardi, S., and Chiu, B., "A symbolic representation of time series, with implications for streaming algorithms," in *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, (New York, NY, USA), pp. 2–11, ACM, 2003.

[99] Lin, S.-d. and Chalupsky, H., "Unsupervised link discovery in multi-relational data via rarity analysis," in *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, (Washington, DC, USA), p. 171, IEEE Computer Society, 2003.

[100] Lu, F., "Uncovering fraud in direct marketing data with a fraud auditing case builder," in *PKDD 2007: Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, (Berlin, Heidelberg), pp. 540–547, Springer-Verlag, 2007.

[101] MacQueen, J. B., "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (Cam, L. M. L. and Neyman, J., eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[102] Mannila, H., Pavlov, D., and Smyth, P., "Prediction with local patterns using cross-entropy," in *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 357–361, ACM, 1999.

[103] Michael Giering, "Retail Sales Prediction and Item Recommendations Using Customer Demographics at Store Level," *SIGKDD Newsletter*, vol. 10, p. 84, Dec. 2008.

[104] Micheline Kamber and Jiawei Han, *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2 ed., Mar. 2006.

[105] Midi, H., Rana, M. S., and Imon, A. H. M. R., "The performance of robust weighted least squares in the presence of outliers and heteroscedastic errors," *WSEAS Trans. Math.*, vol. 8, no. 7, pp. 351–361, 2009.

[106] Minh Quoc Nguyen, Edward Omiecinski, and Leo Mark, "Accurately Ranking Outliers in Data with Mixture of Variances and Noise," *IADIS Multi Conference on Computer Science and Information Systems*, June 2009.

[107] Minh Quoc Nguyen, Edward Omiecinski, and Leo Mark, "A Fast Feature-based Method to Detect Unusual Patterns in Multidimensional Data," in *11th International Conference on Data Warehousing and Knowledge Discovery*, Aug. 2009.

[108] Minh Quoc Nguyen, Edward Omiecinski, and Leo Mark, "A Fast Randomized Method for Local Density-based Outlier Detection in Multidimensional Data," *(on submission)*, Mar. 2010.

[109] Minh Quoc Nguyen, Leo Mark, and Edward Omiecinski, "Unusual Pattern Detection in High Dimensions," in *The Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2008.

[110] Mörchen, F., "Unsupervised pattern mining from symbolic temporal data," *SIGKDD Explor. Newsl.*, vol. 9, no. 1, pp. 41–55, 2007.

[111] Nath, S. V., "Crime pattern detection using data mining," in *WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, (Washington, DC, USA), pp. 41–44, IEEE Computer Society, 2006.

[112] Newman, C. B. D. and Merz, C., "UCI repository of machine learning databases." http://archive.ics.uci.edu/ml/, 1998.

[113] Noble, C. C. and Cook, D. J., "Graph-based anomaly detection," in *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 631–636, ACM, 2003.

[114] Ogras, Y. and Ferhatosmanoglu, H., "Online summarization of dynamic time series data," *The VLDB Journal*, vol. 15, no. 1, pp. 84–98, 2006.

[115] Olfa Nasraoui, "Book Review: Web Data Mining - Exploring Hyperlinks, Contents, and Usage Data," *SIGKDD Newsletter*, vol. 10, p. 23, Dec. 2008.

[116] Olfa Nasraoui, Myra Spiliopoulou, Osmar R. Zaiane, Jaideep Srivastava, and Bamshad Mobasher, "WebKDD 2008: 10 years of Knowledge Discovery on the Web Post-Workshop report 78," *SIGKDD Newsletter*, vol. 10, p. 78, Dec. 2008.

[117] Oliner, A. J., Aiken, A., and Stearley, J., "Alert Detection in System Logs," in *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (Fosca Giannotti, Dimitrios Gunopulos, Franco Turini, Carlo Zaniolo, Naren Ramakrishnan, and Xindong Wu, eds.), pp. 959–964, IEEE Computer Society, Dec. 2008.

[118] Pandit, S., Chau, D. H., Wang, S., and Faloutsos, C., "Netprobe: a fast and scalable system for fraud detection in online auction networks," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (New York, NY, USA), pp. 201–210, ACM, 2007.

[119] Papadimitriou, P., Dasdan, A., and Garcia-Molina, H., "Web graph similarity for anomaly detection (poster)," in *WWW '08: Proceeding of the*

17th international conference on World Wide Web, (New York, NY, USA), pp. 1167–1168, ACM, 2008.

[120] Papdimitriou, P., Dasdan, A., and Garcia-Molina, H., "Web Graph Similarity for Anomaly Detection," technical report, Stanford, 22 Jan. 2008.

[121] Paul Barford and David Plonka, "Characteristics of Network Traffic Flow Anomalies," in *ACM SIGCOMM Internet Measurement Workshop*, 2001. http://www.imconf.net/imw-2001/imw2001-papers/47.pdf.

[122] Pei Sun, Sanjay Chawla, and Bavani Arunasalam, "Mining for Outliers in Sequential Databases," in *Proceedings of the 2006 SIAM International Conference on Data Mining*, Apr. 2006.

[123] R. B. Dean and W. J. Dixon, "Simplified Statistics for Small Numbers of Observations," *Anal. Chem.*, vol. 23, no. 4, pp. 636–638, 1951.

[124] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan, "Automatic subspace clustering of high dimensional data," *Data Mining and Knowledge Discovery*, 2005.

[125] Robin Sabhnani, Daniel Neill, A. M., "Detecting anomalous patterns in pharmacy retail data," in *Proceedings of the KDD 2005 Workshop on Data Mining Methods for Anomaly Detection*, August 2005.

[126] Salvador, S. and Chan, P., "Learning states and rules for detecting anomalies in time series," *Applied Intelligence*, vol. 23, no. 3, pp. 241–255, 2005.

[127] Schölkopf, B., Platt, J. C., Shawe-Taylor, J. C., Smola, A. J., and Williamson, R. C., "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, 2001.

[128] Serneels, S. and Verdonck, T., "Principal component regression for data containing outliers and missing elements," *Comput. Stat. Data Anal.*, vol. 53, no. 11, pp. 3855–3863, 2009.

[129] Shaft, U. and Ramakrishnan, R., "Theory of nearest neighbors indexability," *ACM Trans. Database Syst.*, vol. 31, no. 3, pp. 814–838, 2006.

[130] Shunsuke Hirose and Kenji Yamanishi, "Latent Variable Mining with Its Applications to Anomalous Behavior Detection," in *Proceedings of the 2008 SIAM International Conference on Data Mining*, Apr. 2008.

[131] Singh, S., Tu, H., Donat, W., Pattipati, K., and Willett, P., "Anomaly detection via feature-aided tracking and hidden markov models," *Trans. Sys. Man Cyber. Part A*, vol. 39, no. 1, pp. 144–159, 2009.

[132] Spiros Papadimitriou, Hiroyuki Kitagawa, Philip B. Gibbons, and Christos Faloutsos, "LOCI: Fast outlier detection using the local correlation integral," in *Proceedings of the 19th International Conference on Data Engineering: 2003*, pp. 315– 326, IEEE Computer Society Press, Mar. 2003.

[133] Srikant, R. and Agrawal, R., "Mining quantitative association rules in large relational tables," *SIGMOD Rec.*, vol. 25, no. 2, pp. 1–12, 1996.

[134] Steinwart, I., Hush, D., and Scovel, C., "A classification framework for anomaly detection," *J. Mach. Learn. Res.*, vol. 6, pp. 211–232, 2005.

[135] Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., and Gunopulos, D., "Online outlier detection in sensor data using non-parametric models," in *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pp. 187–198, VLDB Endowment, 2006.

[136] Sushmito Ghosh and Douglas L. Reilly, "Credit Card Fraud Detection with a Neural-Network," *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, 1994*, 1994.

[137] Tukey, J. W., *Exploratory data analysis.* Addison-Wesley, 1997.

[138] Vishwanathan, S. V. N., Borgwardt, K. M., and Schraudolph, N. N., "Fast computation of graph kernels," in *NIPS*, pp. 1449–1456, 2006.

[139] Walton College Teradata, "Walton College Teradata." http://spartan.walton.uark.edu/sqlassistantweb1.2/.

[140] Weigend, A. S., Mangeas, M., and Srivastava, A. N., "Nonlinear gated experts for time series: Discovering regimes and avoiding overfitting," *International Journal of Neural Systems*, 1995.

[141] Weiyun Huang, Edward Omiecinski, Leo Mark, and Minh Quoc Nguyen, "History Guided Low-Cost Change Detection in Streams," in *11th International Conference on Data Warehousing and Knowledge Discovery*, Aug. 2009.

[142] Weng-Keen Wong, Andrew Moore, Gregory Cooper, and Michael Wagner, "Bayesian network anomaly pattern detection for disease outbreaks," in *Proceedings of the Twentieth International Conference on Machine Learning* (Fawcett, T. and Mishra, N., eds.), (Menlo Park, California), pp. 808–815, AAAI Press, August 2003.

[143] Xiaoxin Yin, "Scalable Mining and Link Analysis Across Multiple Database Relations," *SIGKDD Newsletter*, vol. 10, p. 23, Dec. 2008.

[144] Xingquan Zhu and Xindong Wu, "Discovering Relational Patterns across Multiple Databases," in *IEEE 23rd International Conference on Data Engineering, 2007*, pp. 726–735, 2007.

[145] Yan, X. and Han, J., "gspan: Graph-based substructure pattern mining," in *ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining*, (Washington, DC, USA), p. 721, IEEE Computer Society, 2002.

[146] Ye, M., Li, X., and Orlowska, M. E., "Projected outlier detection in high-dimensional mixed-attributes data set," *Expert Systems with Applications*, vol. In Press, Corrected Proof, pp. –, 2008.

[147] Zhu, S., Li, T., Chen, Z., Wang, D., and Gong, Y., "Dynamic active probing of helpdesk databases," *PVLDB*, vol. 1, no. 1, pp. 748–760, 2008.

INDEX

α function, 97

accumulated sub-dimensional variations, 60
adaptive dual-neighbor
    adaptive dual-neighbor method, 5
Adaptive Nearest Neighbor (ANN), 46
adaptive neighbor, 70
adaptive neighbor distance, 48
aggregated variance ratio, 74
Akaike Information Content (AIC), 9
angle-based method, 24
anomaly detection in sequential data, 1

binary tree, 31
boundary point, 47

Chebyshev distance, 66
core rule, 97
curse of dimensionality, 23, 64
cutoff threshold, 74

decision boundary, 70
density-based clustering, 52
    SNN, 52
dimensional average local distance, 71
dimensional density, 69
dimensional ratio, 76
dimensional variance ratio, 72
distance-based method, 24
dual-neighbor, 44, 49
    dual-neighbor relationship, 48, 77

Euclidean distance, 66

feature bagging, 65
feature selection, 64
filtering, 68

global outlier, 3
granularity level, 47
graph-based anomaly, 19
group of outliers, 75
Grubb's test, 8

hyperplane, 22, 27
hypothesis test, 8

i-distance, 46
i-neighbor, 46
indirect dual-neighbor, 51
    indirect dual-neighbor relationship, 48
information entropy, 64

k-distance, 25, 92
k-distance neighborhood, 25
k-nearest neighbor, 4, 22, 23, 43, 44
KDD CUP 99 network connection dataset, 81
kmean, 12, 13, 52, 112

level of granularity, 70
local density, 24
local density based method, 24
local density-based outlier, xii, 3, 25
    LOF, 52
local dimensional density, 70
local outlier factor, xii, 24, 26, 92
local reachability density, 25
locally based, 24
LOCI, 24

M-estimation, 20
Markovian model, 15
Maximum Likelihood Estimation (MLE), 20
mixture of variances, 60
multi-dimensional index tree, 4
mutual information, 10, 64

neural network, 6, 10
nonrandomized version, 23
normalized measure, 16

odds measure, 16
one-class support vector machine, 11
outlier cluster score, 76
outlier detection, 22

parallelization, 122
partition, 28
point-based outlier detection, 19
Principal Component Analysis (PCA), 64
Probabilistic Suffix Automata, 15
Probabilistic Suffix Tree, 16

quantitative correlated pattern mining, 94
quantitative correlation analysis, 91

randomization method, 23
randomized method, 4
randomized version, 23
reachability distance, 25
robust statistics, 20

sequential data, 15
shared nearest neighbor, 13, 45
split function, 105
subspace clustering, 65
Support Vector Machine (SVM), 11
Symbolic Aggregate ApproXimation (SAX), 18

VITA

Minh received his B.S. degree in Computer Science in 2003 from Assumption University, Thailand. He was a Ph.D student in the College of Computing, School of Computer Science at Georgia Institute of Technology from 2004 to 2010. Minh belongs to the Database lab under the supervision of Professors Leo Mark and Edward Omiecinski. Minh's research area is databases system and data mining, in particular anomaly detection. He has been a teaching assistant for undergraduate and graduate courses in database design and implementation for more than three years. Minh received several awards before joining Georgia Tech, including the Vietnam Education Foundation Fellowship Award.

Minh married Ms. Tram Tran in 2005 and had a daughter named Laura Sofia Nguyen in 2009.