APPROACHES TO ABNORMALITY DETECTION WITH CONSTRAINTS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the
Graduate School of The Ohio State University

By

Matthew Eric Otey, B.S., M.S.

The Ohio State University

2006

Dissertation Committee:

Dr. Srinivasan Parthasarathy, Adviser
Dr. Gagan Agrawal
Dr. Dhabaleswar Panda

Approved by

Adviser
Graduate Program in
Computer Science and
Engineering
ABSTRACT

A common problem in data analysis is that of discriminating between modes of normal behavior and modes of abnormal behavior. Of particular interest are techniques that can automatically detect abnormal activity in data. This is important since abnormal data may be indicative of measurement error in scientific data, or malicious activity in security audit data. There are two basic approaches to the problem of automatically finding abnormalities. The first is known as signature detection, which involves finding known patterns of abnormality in a database. However, it has the drawback of not being able to detect abnormalities for which there is no prior information. The second approach is known as anomaly detection, which involves building a model of normal data and then searching for patterns that do not fit this model. Unlike the signature detection approach, it is able to detect abnormalities for which there is no prior information, but has the drawback that the anomalies it does detect may not be (significantly) abnormal. The most successful approaches will use both signature detection and anomaly detection techniques to utilize their combined strengths.

Much of the previous research in this area has focused on more general approaches to anomaly and signature detection. However, this work is focused on carrying out anomaly and signature detection under various constraints. For example, the data may contain heterogeneous attribute types, or have missing values. The data may also be distributed across several computers or streaming in at a high rate of speed, or there may be limitations on
the resources available to analyze the data. In this work, we develop novel solutions to the
abnormality detection problem with constraints, and empirically test them on various real
and synthetic data sets.
ACKNOWLEDGMENTS

I wish to thank Amol Ghoting, Gang Li, Wagner Meira jr., Sandeep Narravula, Ranjit Noronha, Donald C. Trost, Adriano Veloso, and Chao Wang for their assistance and their contributions to the work described in this dissertation.

I also wish to thank Gagan Agrawal and Dhabaleswar Panda for serving as core members of my dissertation committee, and Raghu Machiraju, Thomas Nygren, and Boris Pittel for serving as department and graduate school representatives for my candidacy and dissertation committees. All of them have raised important questions and provided valuable insight during the course of this work.

The members of the Data Mining Research Lab have also been supportive and helpful over the past five years, assisting with revisions, presentations, computer problems, and everything else. This group of people includes Sitaram Asur, Greg Buehrer, Keith Marsolo, Sameep Mehta, Weiwei Shi, Shirish Tatikonda, Duygu Ucar, Huai Wang, and Hui Yang. There is also a group of people outside the lab who have been equally helpful: Fatih Altiparmak, Hakan Ferhatosmanoglu, Ruoming Jin, and Steve Romig.

Outside the school, I wish to thank my friends and family for their support, especially Sherry and my parents.

Finally, I wish to thank Srinivasan Parthasarathy for being my adviser, and for giving me plentiful invaluable guidance over the past five years.
VITA

December 16, 1978  . . . . . . . . . . . . . . . . . . . . . . . . . . Born - Harrisonburg, Virginia, United States of America
2001  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . B.S. Computer Science, University of Virginia
2005  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . M.S. Computer Science and Engineering, The Ohio State University

PUBLICATIONS

Research Publications


**FIELDS OF STUDY**

Major Field: Computer Science and Engineering

Studies in Data Mining: Professor Srinivasan Parthasarathy
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
<td></td>
<td>ii</td>
</tr>
<tr>
<td><strong>Acknowledgments</strong></td>
<td></td>
<td>iv</td>
</tr>
<tr>
<td><strong>Vita</strong></td>
<td></td>
<td>v</td>
</tr>
<tr>
<td><strong>List of Figures</strong></td>
<td></td>
<td>xi</td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
<td></td>
<td>xiv</td>
</tr>
<tr>
<td><strong>Chapters:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Approaches to Abnormality Detection</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Limitations of Existing Abnormality Detection Techniques</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Thesis Statement</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>Contributions</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>Organization</td>
<td>7</td>
</tr>
<tr>
<td>2.</td>
<td>Related Work</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Distance-Based Outlier and Anomaly Detection Methods</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Clustering Approaches to Outlier and Anomaly Detection</td>
<td>10</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Measures of Distance and Similarity</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Frequent Itemset and Association Rule Methods</td>
<td>21</td>
</tr>
<tr>
<td>2.3</td>
<td>Supervised Machine Learning Methods</td>
<td>24</td>
</tr>
<tr>
<td>2.4</td>
<td>Statistical Outlier and Anomaly Detection Methods</td>
<td>27</td>
</tr>
<tr>
<td>2.5</td>
<td>Intrusion Detection Systems</td>
<td>28</td>
</tr>
<tr>
<td>2.6</td>
<td>Limitations of Past Approaches</td>
<td>30</td>
</tr>
</tbody>
</table>
### 3. Resource-Dependent Abnormality Detection on Network Interface Cards

- **3.1 Related Work in NIC-based Computing and Data Stream Processing**
- **3.2 Algorithms**
  - 3.2.1 Anomalous Client Detectors
  - 3.2.2 Hybrid Models For Anomaly Detection
- **3.3 Experimental Results**
  - 3.3.1 Setup
  - 3.3.2 Results
- **3.4 Summary**

### 4. Distributed Approaches for Anomaly Detection Using Frequent Itemsets

- **4.1 Distributed Incremental Frequent Itemset Mining**
- **4.2 Background and Related Work**
- **4.3 Algorithms**
  - 4.3.1 Problem Definition
  - 4.3.2 The ZtGZAG Incremental Algorithm
  - 4.3.3 Distributed Incremental Algorithm
  - 4.3.4 Interactivity Issues
  - 4.3.5 Discussions
- **4.4 Experimental Evaluation**
  - 4.4.1 Distributed Incremental Algorithm Evaluation
- **4.5 Summary**

### 5. Distributed Anomaly Detection in Mixed Attribute Data

- **5.1 Algorithms**
  - 5.1.1 Computing Anomaly Scores
  - 5.1.2 Distributed Model Construction
  - 5.1.3 Distributed Outlier Detection
  - 5.1.4 Complexity Analysis and Modifications for Increased Efficiency
  - 5.1.5 Single Pass Approach for Dynamic Data Sets
- **5.2 Experimental Results**
  - 5.2.1 Setup
  - 5.2.2 Qualitative evaluation
  - 5.2.3 Speedup in a Distributed Setting
  - 5.2.4 Benefits of the Approximation Scheme
  - 5.2.5 Execution time variation with number of attributes
  - 5.2.6 Evaluation of one pass approach
- **5.3 Conclusions**
6. Reduced-Memory Outlier Detection .......................... 112

   6.1 A Modified Outlier Detection Algorithm ............... 112
       6.1.1 Predicting Values of Categorical Attributes .......... 113
       6.1.2 Finding Anomalous Values of Continuous Attributes .... 114
       6.1.3 The RELOADED Algorithm .......................... 115
       6.1.4 Computational and Space Complexity .................. 117

   6.2 Experimental Results .................................. 119
       6.2.1 Setup .............................................. 119
       6.2.2 Detection Rate ...................................... 121
       6.2.3 Memory Usage ....................................... 124
       6.2.4 Execution Time ...................................... 125

   6.3 Conclusions and Future Work ........................... 127

7. A Dissimilarity Measure for Comparing Subsets of Data .......... 129

   7.1 Introduction ........................................... 129

   7.2 Algorithms ............................................ 132
       7.2.1 Dissimilarity Measure ............................... 132
       7.2.2 Example ............................................. 137
       7.2.3 Applications ........................................ 140

   7.3 Experimental Results ................................... 145
       7.3.1 Setup .............................................. 145
       7.3.2 Change Point Detection .............................. 145
       7.3.3 Anomaly Detection .................................... 146
       7.3.4 Data Set Clustering .................................. 150
       7.3.5 Missing Data ........................................ 152

   7.4 Conclusion .............................................. 152

8. Dissimilarity Measures for Detecting Hepatotoxicity in Clinical Trial Data . . 154

   8.1 Introduction ........................................... 154

   8.2 Algorithms ............................................ 156
       8.2.1 Dissimilarity Measures ............................... 157
       8.2.2 Applications ........................................ 159

   8.3 Experimental Results ................................... 161
       8.3.1 Setup .............................................. 161
       8.3.2 Anomaly Detection .................................... 162
       8.3.3 Data Set Clustering .................................. 165

   8.4 Conclusion .............................................. 165
9. Conclusions and Future Work ........................................ 167

9.1 Conclusions ...................................................... 167
9.2 Future Work ..................................................... 169

Bibliography .......................................................... 172
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The k-means algorithm.</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Motivation for adding NIC-based security</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>The $P(SrcIP \mid DstIP)$ anomalous client detector</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>The $P(SrcIP \mid DstPort, DstIP)$ anomalous client detector</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>Anomaly detection using frequent itemsets</td>
<td>41</td>
</tr>
<tr>
<td>3.5</td>
<td>The clustering-based anomaly detection algorithm</td>
<td>42</td>
</tr>
<tr>
<td>3.6</td>
<td>The heuristic function $H$ for determining similarity</td>
<td>43</td>
</tr>
<tr>
<td>3.7</td>
<td>Quantitative comparison of NIC-based and host-based intrusion detection</td>
<td>45</td>
</tr>
<tr>
<td>3.8</td>
<td>Results of different algorithms</td>
<td>47</td>
</tr>
<tr>
<td>3.9</td>
<td>Different intrusions are detected by different algorithms</td>
<td>48</td>
</tr>
<tr>
<td>4.1</td>
<td>Backtrack Trees for Items A and B on $\Delta$</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>Incremental Search for Maximal Frequent Itemsets</td>
<td>60</td>
</tr>
<tr>
<td>4.3</td>
<td>Overall Process of Distributed Mining</td>
<td>65</td>
</tr>
<tr>
<td>4.4</td>
<td>Number of transactions processed</td>
<td>69</td>
</tr>
<tr>
<td>4.5</td>
<td>Query Response Time using Equal Block Sizes</td>
<td>75</td>
</tr>
</tbody>
</table>
4.6 Query Response Time using Different Block Sizes

4.7 Communication Overhead

4.8 High-Contrast Frequent Itemsets

5.1 Lattice management

5.2 Constructing the local model

5.3 Finding outliers in the second pass of the data set

5.4 Finding outliers in a single pass of the data set

5.5 Distributed one-pass algorithm for outlier detection

5.6 Algorithm for hash table management and frequency estimation

5.7 (a) Speedup (b) Expected efficiency in a wide area network with 8 sites

5.8 Execution time variation with increasing lattice levels

5.9 (a) Detection rates (b) False positive rates

5.10 Execution time variation with increasing (a) Categorical attributes and (b) Continuous attributes

5.11 (a) Speedup for one pass approach (b) Expected efficiency in a wide area network

6.1 The RELOADED algorithm

6.2 Peak memory usage with increasing numbers of (a) categorical and (b) continuous attributes

6.3 Plot of execution time versus data set size

6.4 Plot of execution time versus number of categorical attributes

6.5 Plots of execution time versus number of continuous attributes
7.1 A plot of five synthetic data sets.  
7.2 The change point detection algorithm. 
7.3 Outliers in a data set discovered using different measures: (A) Mahalanobis; 
    (B) $D_{II}$; (C) $D_{Σ}$; (D) $D_{Σ}$ distance component only; (E) $D_{Σ}$ rotation com-
    ponent only; (F) $D_{Σ}$ variance component only. 
7.4 Plot of stock indices centered on change point. 
7.5 Scatter plot of stock indices. 
7.6 Outliers in 2004 Merck and Pfizer stock prices. 
7.7 The Pfizer/Merck March 11 outlier and preceding days. 
7.8 Pfizer/Merck 2004 stock prices. 
7.9 Dendrogram resulting from clustering of monthly data sets. 
7.10 Percentage of change points found for differing degrees of missing data. 
8.1 Top outlier rankings using (A) $D_e$ on Drug A in $D1$; (B) $D_e$ on Drug B in $D2$; (C) $D_e$ on Drug C in $D2$. 
8.2 Top Outlier rankings for Drug A as function of the number of components 
    used in $D'_e$: (A) 2 components, (B) 3 components, (C) 4 components.
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4.1</strong> Data set characteristics.</td>
<td>69</td>
</tr>
<tr>
<td><strong>5.1</strong> Detection rates - (a) Our approach and ORCA on 10% KDDCup1999 training data set (b) entire testing data set, (c) Our approach on entire KDDCup 1999 training data set</td>
<td>101</td>
</tr>
<tr>
<td><strong>5.2</strong> Detection rates for the single pass approach (Centralized and Distributed) for KDDCup 1999 data</td>
<td>108</td>
</tr>
<tr>
<td><strong>6.1</strong> Detection rates for the KDDCup data sets. The best detection rates are in bold. The detection rates for the single-pass version of RELOADED is given in parenthesis.</td>
<td>122</td>
</tr>
<tr>
<td><strong>6.2</strong> Peak heap usage in kilobytes.</td>
<td>123</td>
</tr>
<tr>
<td><strong>7.1</strong> Dissimilarity: distance component.</td>
<td>138</td>
</tr>
<tr>
<td><strong>7.2</strong> Rotation dissimilarity.</td>
<td>138</td>
</tr>
<tr>
<td><strong>7.3</strong> Variance dissimilarity.</td>
<td>139</td>
</tr>
<tr>
<td><strong>7.4</strong> Total dissimilarity ($D_{II}$).</td>
<td>139</td>
</tr>
<tr>
<td><strong>7.5</strong> Detection rates of notable outliers using the $D_{II}$ measure and a Mahalanobis metric-based $D_{Σ}$ measure (in parenthesis).</td>
<td>148</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

In recent decades, technology has allowed people and organizations to generate and collect large amounts of data. However, this data is worthless unless the valuable information (the knowledge) contained within it can be extracted. To solve this problem, data mining has emerged as a means of automatically analyzing large amounts of data to find pieces of interesting information. It combines elements from the fields of statistics, machine learning, and high-performance computing. Statistics and machine learning provide the techniques that can extract useful analyses from data, while high-performance computing provides the means of handling large amounts of data and the possibly computationally intense algorithms needed to perform the analysis. Data mining has many applications, from market basket data analysis to medical diagnosis, from stock price prediction to gene expression analysis, and more.

A common problem in data mining is that of automatically finding abnormal data in a database. This is a very important problem in today’s world, where abnormalities can be indicative of bad data or malicious behavior. Examples of bad data include skewed data values resulting from measurement error, or erroneous values resulting from data entry mistakes. An example of data indicating malicious behavior include anomalous transactions in a credit card database, which may be symptoms of someone using a stolen card or
engaging in other fraudulent behavior [23]. In the field of network traffic analysis, anomalous network packets may indicate either a possible intrusion or attack, or a failure in the network [142]. In the domain of patient medical data, abnormalities may arise when the patient is afflicted with some disease, or suffers some side-effects from a drug [127]. Efficient detection of such abnormalities reduces the risk of making poor decisions based on erroneous data, and aids in identifying, preventing, and repairing the effects of malicious or faulty behavior.

Additionally, many data mining algorithms and techniques for statistical analysis may not work well in the presence of these abnormalities. Abnormalities and other outliers may introduce skew or complexity into the model. Additionally, they make it difficult, if not impossible, to fit an accurate model to the data in a computationally feasible manner. For example, statistical measures of the data may be skewed because of erroneous values, or the noise of the abnormalities may obscure the truly valuable information residing in the data set. Accurate, efficient removal of these abnormalities may greatly enhance the performance of statistical and data mining algorithms and techniques [49]. Finding and eliminating such abnormalities as a pre-processing step for other techniques is known as data cleaning. As can be seen, different domains have different reasons for discovering abnormalities: Abnormalities may be noise that we may want to remove, since they obscure the true patterns we wish to discover, or the abnormalities may be the very things in the data we wish to discover. As has been said before, “One person’s noise is another person’s signal” [84].

2
1.1 Approaches to Abnormality Detection

There are two common approaches to the problem of finding abnormalities. The first is known as signature detection (supervised detection), which involves finding known patterns of abnormality in a database using classification and pattern recognition techniques. The other approach is that of anomaly detection (unsupervised detection), which involves building a model of normal data and then searching for data that does not fit this model, using outlier detection techniques. The signature detection approach can easily find abnormalities for which it has patterns, but it has the drawback that it is difficult to find abnormalities for which no patterns exist beforehand. The anomaly detection approach, however, is able to detect these novel abnormalities, but has the drawback that many of the anomalies it discovers may not be (significantly) abnormal. Another drawback of anomaly detection systems is that the model of normality must be constructed using data that contains no anomalies, which can be difficult to obtain in practice. A better approach to discovering abnormalities would be to use a combination of signature and anomaly detection approaches. The combined strengths of the approaches help to nullify their combined weaknesses.

1.2 Limitations of Existing Abnormality Detection Techniques

Over the years, a large number of techniques have been developed for abnormality detection. However, real-world data sets and environments present a range of difficulties that limit the effectiveness of these techniques. Some of these problems are: (a) there are (strict) limits on how much time can be spent by an algorithm to find anomalies; (b) the data set may be dynamic or contain a temporal component; (c) the data set may be distributed; (d) the data set may contain heterogeneous features (i.e. continuous and categorical features) and missing data.
The problem of time limits is very evident in the field of network intrusion detection, where abnormalities must be detected in real time. Such constraints are necessary so that a network administrator can act quickly to prevent or reduce damage to the network and the machines residing on it. Also, because of the real-time constraints, the abnormality detection techniques must be computationally inexpensive, so as not to adversely impact the performance of the network or the machines residing on it. Additionally, since the data set is streaming, abnormality detection schemes are limited to a single pass over the data. While this is not difficult for many signature detection schemes, most anomaly detection schemes, most notably outlier detection schemes, require multiple passes over the data set.

A related problem is that of evolving data sets. Evolving data sets are those where data is constantly being added, removed, or otherwise updated. This situation occurs quite often: network data streams are, by definition, evolving, and commercial transaction databases are constantly being augmented by new purchases. In many cases, the processes generating this new data may be non-stationary, meaning that a model of the data built at one point in time may become invalid in the future. In network intrusion detection, the models of network traffic and intrusion patterns are in a state of constant change, as new network protocols, virii, and intrusion techniques are constantly being developed and modified. The models representing the data in commercial transaction databases may also be in flux, since the purchases may be dependent upon the time of year (a phenomenon known as seasonality), and the demographics of the population of purchasers (since people may be moving in to or out of a region) This problem is known as concept drift. Any robust abnormality detection technique must be able to handle these types of evolving models.
Another difficult problem to deal with is the problem of distributed data sets. In the domain of network intrusion detection, there may be multiple edge routers collecting disjoint information on the status of the network. In the domain of commercial transaction analysis (market basket analysis), the branches of a department store chain may collect data on each purchase made at their respective stores. In order to build a global model of the items purchased, information from each store’s data set must be combined in some way. A naïve approach would be to transfer all the data to a single site and apply some abnormality detection technique there. However, this might not be possible if the resulting data set would be too large, or it is too expensive in terms of network resources to transfer the data to a single site. To discover abnormalities in distributed data, existing techniques must be extended to work in parallel or distributed environments, or novel techniques must be developed.

Finally, the features in a data set may be a heterogeneous mixture of categorical (nominal) and continuous types, and there may be missing values. Categorical or nominal features may take on a relatively small number of values, and these values have no ordering or only a partial ordering. Continuous features have values that are usually numeric and have a total order defined on them. Many techniques for abnormality detection assume that all features in a data set are either continuous or categorical. However, many real-life data sets contain a mixture of types. For example, a data point representing a network flow may contain continuous types (the number of bytes transferred between two hosts, the time the connection started, etc) and categorical types (the service accessed, the protocol used, etc). Having different feature types make it difficult to find relations between two features (for example, the correlation between the features) and to define distance or similarity metrics.
for such data points (for example, what is the distance between the TCP and UDP protocols?). When processing data sets with a mixture of feature types, many techniques convert the continuous features into categorical features by discretization (quantization), or convert categorical features into continuous features by applying some (arbitrary) ordering, which can lead to a loss in information and an increase in noise. A better abnormality detection system would need to develop meaningful and useful distance metrics in mixed-type feature spaces and measures relations between features of different types. Furthermore, missing values also present a problem for distance metrics (what is the distance between something and nothing?), and finding relations between features.

1.3 Thesis Statement

These problems lead to our thesis, which can be stated as abnormality detection techniques can be modified to operate under the constraints presented by real-world data and environments.

1.4 Contributions

In this dissertation we present our work that addresses the four limitations presented above in Section 1.2. Our work addresses these limitations in the following ways:

- We present a technique for detecting network intrusions at the level of the network interface card. This technique can detect intrusions in streaming data in real time, addressing limitations (a) and (b).

The use of “we” in this dissertation is a stylistic convention commonly used in the field of Computer Science. The work presented in this dissertation is mainly that of the author unless explicitly identified otherwise.
• We present a technique for building a global model of a dynamic, distributed data set that can facilitate abnormality detection in distributed domains, addressing limitations (b) and (c).

• We present a technique for finding outliers in distributed data streams which include a mixture of categorical and continuous attributes in real time, addressing limitations (a), (b), (c), and (d).

• Extend our previous work by reducing memory requirements and response times, addressing limitation (a).

• Discover abnormal time series in a collection with possible missing values and irregular sampling by developing dissimilarity measures for comparing two time series, addressing limitations (b) and (d).

1.5 Organization

The remainder of this dissertation is as follows. We first review the wealth of related work in signature and anomaly detection in Chapter 2. The related work draws on techniques from many diverse areas of data mining, including outlier detection, clustering, frequent itemset mining, classification, and statistical modeling. We also put special emphasis on the application domain of network intrusion detection, where much of this work has been done.

We next present our work in the domain of abnormality detection. Some of our work focuses on network intrusion detection [108, 109]. Specifically, we examined means of detecting intrusions at the level of the network interface card, where computational and
memory resources are extremely limited. The details of this work can be found in Chapter 3. Other work focuses on discovering anomalies in a distributed setting. For example, we use the idea of high-contrast frequent itemsets to characterize the differences between models of data stored at distributed sites and the global model of the data [114]. The details of this approach can be found in Chapter 4. We also use frequent itemsets in conjunction with correlations to detect anomalies in distributed streaming data sets containing a heterogeneous mixture of categorical and continuous attributes in a technique called LOADED [110]. We present the details of this work in Chapter 5. We present a variation on this anomaly detection scheme called RELOADED [113] that uses classifiers instead of frequent itemsets to model the categorical attributes in Chapter 6. Finally, in Chapters 7 and 8, we present dissimilarity measures that can be used to compare multivariate time series against each other for purposes of anomaly detection and clustering [111], and for detecting hepatotoxicity in pharmaceutical clinical trial data [112].
CHAPTER 2

RELATED WORK

There has been a large amount of previous work in the field of anomaly and signature detection. Bolton and Hand provide an overview of some of this work in their review of statistical fraud detection [23]. Much of this work can be grouped according to the general approaches used. These approaches include distance-based outlier detection methods, frequent itemset methods, machine learning methods, and statistical outlier and anomaly detection methods, each of which is treated in a separate section in this chapter. Furthermore, the domain of network intrusion detection is a central application of anomaly and signature detection, and so we will discuss the related work in that area separately. Finally, we examine some of the limitations of these approaches.

2.1 Distance-Based Outlier and Anomaly Detection Methods

One way of defining an anomaly is that it is a data point that lies far away from (most) other data points in the feature space. This definition gives rise to distance-based outlier detection methods. One of the first approaches for discovering outliers using distance metrics was first presented by Knorr et al. [82, 83, 85]. They define a point to be a distance outlier if at least some user-defined fraction of the points in the data set are further than some user-defined minimum distance away from that point. In other words, outliers are
those points that lie in regions of the data space that have relatively low density. Knorr et al. primarily focus on data sets containing only real valued attributes in their experiments.

Most methods for detecting outliers in this manner take time that is quadratic in the number of items in the data set, which can be unacceptable if the data set is very large. To address this issue, a method for discovering outliers in near linear time has been proposed [17]. The central idea is to perform pruning by keeping a monotonically decreasing score for each point in the data set. If the score falls below a certain threshold, then further processing on the data point is not necessary. In the worst case (when there are no outliers), the algorithm still takes quadratic time, but in practice the algorithm runs very close to linear time.

Recently, density-based approaches to outlier detection have been proposed [24]. In this approach, a local outlier factor (LOF) is computed for each point. The LOF of a point is based on the ratios of the local density of the area around the point to the local densities of its neighbors. The size of a neighborhood of a point is determined by the area containing a user-supplied minimum number of points (MinPts). A similar technique called LOCI (Local Correlation Integral) addresses the difficulty of choosing values for MinPts in the LOF technique by using statistical values derived from the data itself [117]. The paper also presents LOCI plots, which provide a way of presenting a single data point’s relation to the rest of the data.

2.1.1 Clustering Approaches to Outlier and Anomaly Detection

Clustering is a very similar problem to distance-based outlier detection. In fact, clustering is the converse problem of outlier detection: In clustering, one tries to find subsets
of data points where each point is very close or similar to other points in the same subset, but different from points in the other subsets. Since outliers are by definition different from most other points, when clustering is performed, they are those points that are either grouped together in relatively small subsets or not assigned any subset at all. Since clustering is a core task in data mining, machine learning, and statistics, there has been a tremendous amount of work done in this area over the years, and numerous clustering methods have been proposed. There are several different classes of clustering methods, but among the most popular are: partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods [65]. Jain et al provide a very extensive survey of clustering approaches [76].

Partitioning methods work by splitting the data into $k$ partitions and iteratively modifying the partitions. The goal is to minimize some function $M$ (usually the variance) that measures the similarity of the data in each partition. However, finding the global minimum of $M$ is computationally infeasible, as it requires the enumeration of all possible partitions [65]. Therefore, for non-trivial data sets, heuristic methods must be used. The most popular partitioning method is the k-means algorithm [95], which defines the centers of clusters in terms of their means, and assigns an object to the cluster with closest mean, according to some distance metric $D$. This process repeats itself until some stopping criteria is met. Examples of stopping criteria are if none of the clusters changes composition between two successive iterations, or if a maximum number of iterations has been met. If we use the former stopping criteria, then this operation will find a local minimum of the function $M$. The k-means algorithm can be seen in Figure 2.1. The primary benefits of the k-means algorithm are its simplicity and the fact that it runs in linear time with respect to the number of objects to be clustered. The k-means algorithm is not unlike the Expectation-Maximization
procedure k-means(Data \( D \), integer \( k \))
begin
    Let \( C \) be an array of \( k \) clusters
    Let \( \mu \) be an array of \( k \) means
    \( \forall i \leq k \) \( \mu_i = \text{some initial value} \)
    repeat
        \( \forall i \) clear cluster \( C_i \)
        \( \forall d_j \in D : \)
        \( l = \text{argmin}_{i \leq k} \{ \text{Distance}(d_j, \mu_i) \} \)
        \( C_l = C_l \cup \{ d_j \} \)
        \( \forall i \leq k \) \( \mu_i = \text{mean}(C_i) \)
    until the stopping criteria is met
end.

Figure 2.1: The k-means algorithm.

(EM) algorithm [39]; a derivation of the k-means algorithm from the EM algorithm can be found in [103]. There are several variations on the k-means algorithm. The k-medoids algorithm [81] uses the medoids (the most centrally located objects) of the clusters instead of the means. A similar medoids-based approach [45] uses the average deviation instead of the variance for the function \( M \). There has also been work that generalizes k-means style clustering to a larger set of data types and distance metrics using the idea of Bregman divergences [11]. When partitioning methods such as these are used, the outliers are the points which are most distant from all cluster centers.

One of the major drawbacks of partitioning methods such as k-means is specifying the number \( k \) of clusters to find. Most often, this value is not known a priori, and therefore must be estimated using trial-and-error, statistical, or analytical approaches [102, 148, 135]. However, specifying the number of clusters is not necessary for hierarchical clustering algorithms. Hierarchical approaches work by either repeatedly merging the closest pairs of
clusters (a bottom-up approach) or recursively dividing the clusters with the largest intra-cluster dissimilarities (a top-down approach). Some of the earliest hierarchical cluster algorithms, a bottom-up method called AGNES, and a top-down method called DIANA, were presented in [81]. Other clustering algorithms that use hierarchical methods are BIRCH [160], CURE [60], ROCK [62], and CHAMELEON [80]. An interesting result of hierarchical clustering is a dendrogram, a graphical representation of how the clusters were merged/divided during the course of the algorithm [67]. A Dendrogram is a binary tree, where the root node represents the entire data set, the leaves represent each data point, and each parent node is a cluster formed by merging the two child clusters. This graphical representation makes it easy to identify outliers. This stems from the fact that since outliers are dissimilar from most other points in the data set, they are more likely to be merged late in the process in a bottom-up approach, or divided out early in a top-down approach. Therefore, the outliers will be the leaves of the tree that have the shortest path to the root node.

Density-based methods grow clusters as long as the number of data points within the vicinity of a cluster is sufficiently large. An advantage of density-based methods over others is their ability to discover clusters of arbitrary shape. Examples of density-based methods are DBSCAN [44] and DENCLUE [71]. In density-based clustering, the outliers are those points not assigned to any cluster (since they occur in regions of low density). However, as Breunig et al point out, though these algorithms can find outliers, they are not capable of determining to what degree a data point is an outlier [24].

Grid-based methods quantize the attribute space into a finite number of grid cells in order to decrease processing time (as the algorithms are dependent not on the number of data points, but the number of cells in the grid). STING [152], WaveCluster [143], and CLIQUE [6] are examples of grid-based clustering methods. CLIQUE uses the Apriori
property, which in this case means that if a $d$-dimensional region in space has a minimum density $\delta$, then when that region is projected into smaller subspaces, the density of the region must be at least $\delta$. The idea is then to find regions of high density in low-dimensional space and then intersect different subspaces to find regions of high density in a higher-dimensional space. Outliers in this case are those data points that lie in low-density cells or subspaces. This is the approach used by Aggarwal and Yu [3]. However, finding low-density cells is difficult, since there are no upward or downward closed properties of such cells. Therefore, only an exhaustive search can find all such cells. However, as this is not practical in high-dimensional data, the authors make use of evolutionary search algorithms [103] to approximate the set of low-density cells. Further discussion on using subspace clustering to detect outliers can be found in Section 2.1.1.

Model-based methods attempt to fit the data to some mathematical model, most using either statistical or neural network approaches. Statistical approaches include COBWEB [46], CLASSIT [51], and AutoClass [27]. Neural network approaches include competitive learning [133] and self-organizing feature maps [86]. When model-based approaches are used, outliers are those data points that do not fit the model well. However, such models may suffer from over-fitting or are too costly to compute, leading to reduced outlier detection accuracy.

One could use clustering techniques to find outliers, which may indicate an anomaly. For example, the ADMIT system presented by Sequeira and Zaki [142] uses such an approach. In this approach, sequences of command-lines are clustered using a sequence similarity metric. A set of clusters is built from training data using a intra-cluster similarity threshold. In the testing phase, any sequence which violates the intra-cluster similarity threshold for all clusters is considered an anomaly.
The Packet Header Anomaly Detection (PHAD) scheme developed by Mahoney and Chan also uses clustering techniques to detect anomalies [96, 99]. PHAD detects network traffic anomalies by learning the normal ranges of values in various fields of network packet headers. It does this by using up to \( C \) clusters to represent the range normal values of each field. If there are less than \( C \) clusters, then each new observed value generates a new cluster. If there are more than \( C \), then when a new value is observed, two other clusters must be merged. Each data point is assigned an anomaly score inversely proportional to the number of times the set of clusters has been updated. The intuition behind this anomaly score is that if a field takes on large number of values, then a novel value for that field is less anomalous than for a field that takes on only a small number of values.

Clustering has also been used in an alternative manner to construct an anomaly detector for time series data [135]. In this work, anomalies are detected by comparing the actual behavior of a given time series to its expected behavior as defined in terms of a finite state automaton (FSA). The states of the FSA correspond to the clusters found by a novel clustering algorithm called Gecko. The Gecko algorithm forces the clusters to be non-overlapping along the time axis to insure that there are no loops in the FSA. Next, RIPPER [35] is used to generate a set of rules that describe each state. These descriptions are then used to build the state transitions. For each state, there are three possible transitions when a given input is received: a transition to the current state if the input matches the description of the current state, a transition to the next state if the input matches the description of the next state, and a transition to an anomaly state if the input does not match the descriptions current or next states. One limitation of this approach is its simplicity. It only allows for a single mode of normality in the time series, and it does not allow for the FSA to re-enter a state after it has left it. In some cases, there may be several different valid states at a given
time point, and it may be the case that the FSA should iterate between two or more states, especially if the mode of normality is characterized by a cyclical behavior over time. The Gecko algorithm does not allow for this since the clusters it finds must be non-overlapping along the time dimension.

**Clustering in Subspaces**

Many existing clustering techniques (see Section 2.1.1) attempt to find clusters that span the entire feature space. However, it may be the case that only a subset of the features are relevant to the structure of the clusters (that is to say, the dimensions in the cluster which have the smallest variances). When we perform anomaly or outlier detection, we assume that anomalies are those points that are most distant from all cluster centers when all dimensions are considered. However, we may miss anomalies that are relatively close to some cluster center only because the distance in the irrelevant dimensions (those with high variance) is small. The distance in the relevant dimensions may still be very large, but this is masked by the effect of the irrelevant dimensions. To find these anomalies, we must be able to determine which dimensions (features) are relevant to each cluster in the data set. This is the problem of clustering in subspaces, or co-clustering.

There has been some work in clustering data in subspaces. Some of the earliest work is that of Hartigan [66]. This approach views the data set as a matrix and recursively partitions that matrix into smaller matrices using either horizontal or vertical splits. The goal is to minimize the sum of squares (variance) of the partitions. This approach is similar to top-down hierarchical clustering. However, the nature of Hartigan’s algorithm limits where splits can occur in a matrix. An approach by Dhillon et al [40] for co-clustering contingency tables. Since the contingency table can be viewed as a joint distribution between two random variables (one variable takes on the column values and the other takes on the
row values), techniques from information theory can be applied to find a co-clustering that leads to the largest mutual information between the two random variables. However, their approach is limited to clustering contingency tables. Grid-based clustering methods (see Section 2.1.1) are also able to find clusters in subspaces. Parsons et al also provide a review of subspace clustering [120].

2.1.2 Measures of Distance and Similarity

Not only are there many different methods for distance-based outlier and Anomaly detection to chose from, there are also many different distance or similarity metrics that can be used. For continuous data, metrics such as the Euclidean, Manhattan, or Canberra distances can be used. Defining a notion of similarity or distance for categorical attributes is more difficult. One measure for categorical attributes is the Jaccard coefficient [77], which measures the similarity of two sets of items (or data points). However, this measure ignores other data points in the neighborhood. The rock hierarchical clustering algorithm addresses these shortcomings [62]. ROCK is based on the idea of links, that is to say, two data points are similar if they have common neighbors. Another method for clustering categorical data is the k-modes algorithm [73]. It is a partitioning approach similar to k-means or k-medoids, but uses the mode of a cluster to define the center. The distances between two data points are calculated using the Hamming distance metric. Other approaches to clustering categorical data attempt to find an optimal clustering by minimizing an entropy criterion [92]. The approach attempts to minimize the entropy of data set when it has been partitioned into a set of clusters. The criterion measures the difference between the entropy of a single cluster containing all data points and the average entropy of a set of $k$ clusters. This approach uses a Monte-Carlo method that iteratively moves a random point to another
cluster if this results in an increase in the difference in entropy. This repeats until there is
no change in cluster membership, which indicates that an optimal partition has been found.
Another approach to clustering using ideas from information theory is that of Gondek and
Hofmann [57]. This approach is capable of constraining the clustering operation, such that
the clusters found are not redundant with respect to classes or clusters already known to
exist in the data.

Another difficult problem is clustering data that contains both continuous and categor-
ical data. A naïve approach would be to discretize continuous attributes and then apply a
categorical clustering algorithm, but the discretization removes information from the con-
tinuous attributes, which may lead to sub-optimal clustering. A more optimal approach
would be to retain the attributes in their original form and develop new distance metrics
that take into account both categorical and continuous distances. One such approach is
that of k-prototypes [72]. The k-prototypes algorithm is a variation on the k-means and
k-modes algorithms for clustering data. Instead of using the mean or mode to represent the
cluster center, the k-prototypes algorithm uses a prototype. A prototype is a representative
vector for the cluster, where the continuous attributes are represented by their means in the
cluster, and the categorical attributes are represented by their modes. A distance measure
$D$ is then defined between two objects $x$ and $y$ as follows:

$$D(x, y) = E(x_r, y_r) + \gamma \times \Delta(x_c, y_c)$$  \hspace{1cm} (2.1)$$

where $x_r$ and $y_r$ are the vectors of the continuous attributes in $x$ and $y$, and $x_c$ and $y_c$ are
the vectors of the categorical attributes in $x$ and $y$. $E$ is a similarity or distance metric that
is defined on continuous attributes, and $\Delta$ is a similarity or distance metric that is defined
on categorical attributes. In [72], $E$ is the Euclidean distance metric,

$$E(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

(2.2)

and $\Delta$ is the Hamming distance metric,

$$\Delta(x, y) = \Sigma_i \delta(x_i, y_i)$$

(2.3)

where

$$\delta(x_i, y_i) = \begin{cases} 0 & \text{if } x_i = y_i \\ 1 & \text{otherwise} \end{cases}$$

(2.4)

Finally, $\gamma$ is a user-supplied weight that effects how much the algorithm is biased toward the continuous or the categorical attributes. In the case where $\gamma = 0$, the k-prototypes algorithm is identical to the k-means algorithm, since the categorical attributes are ignored.

As mentioned above, there have been many metrics proposed that find the distance or similarity between the records of a data set [5, 75, 56], or the between the attributes of a data set [37, 146]. However, these metrics are defined only between a pair of records or a pair of attributes. Similarity metrics for comparing two data sets have been used in image recognition [74], and hierarchical clustering [76]. The Hausdorff distance [74] between two sets $A$ and $B$ is the minimum distance $r$ such that all points in $A$ are within distance $r$ of some point in $B$, and vice-versa. Agglomerative hierarchical clustering frequently makes use of the single-link and complete-link distances between two clusters [76] to decide which pair of clusters can be merged. The single-link distance between two clusters is the minimum pairwise distance between points in cluster $A$, and points in cluster $B$, while the complete-link distance is the maximum pairwise distance between points in cluster $A$, and points in cluster $B$. There is also an average-link distance [65], which is the average of all pairwise distances between points in cluster $A$, and points in cluster $B$. However, these metrics do not explicitly take into account the correlations between attributes in the
data sets (or clusters). Parthasarathy and Ogihara [123] propose a similarity metric for clustering data sets based on frequent itemsets. By this metric, two data sets are considered similar if they share many frequent itemsets, and these itemsets have similar supports. This metric takes into account correlations between the attributes, but it is only applicable for data sets with categorical or discrete attributes.

There has also been work for defining distance metrics that take into account the correlations present in continuous data. The most popular metric is the Mahalanobis distance [128], which accounts for the covariances of the attributes of the data. However this can only be used to calculate the distance between two points in the same data set. Yang et al propose an algorithm for subspace clustering (i.e. subsets of both points and attributes in a data set) that finds clusters whose attributes are positively correlated with each other. However, this algorithm is concerned with searching for subsets of a given data set, as opposed to comparing two data sets explicitly, and it is also limited by the fact that it can only find positive correlations. Böhm et al [20] modify the DBSCAN algorithm [44] by using PCA to find clusters of points that are not only density-connected, but correlation-connected as well. That is to say, they find subsets of a data set that have similar correlations. To determine if two points of the data set should be merged into a single cluster, they must be in each other’s “correlation” neighborhood which is determined by a PCA-based approximation to the Mahalanobis distance. This approach is more flexible than Yang et al’s in that it can find clusters with negative correlations between the attributes. However, its basic density-based approach merges points incrementally so that the resulting clusters have similar correlations, which is not appropriate for merging whole data sets into clusters with similar correlations. Furthermore, their measure is unable to find subsets of data with similar correlations that are not density-connected.
Recently, Aggarwal has argued for user interaction when designing distance functions [2] between points. He presents a parametrized Minkowski distance metric and a parametrized cosine similarity metric that can be tuned for different domains. He also proposes a framework for automatically tuning the metric to work appropriately in a given domain. Based on these ideas in the next section we present a tunable metric for computing a measure of dissimilarity across data (sub)sets.

2.2 Frequent Itemset and Association Rule Methods

Frequent itemsets are sets of items or feature values that frequently occur together in a transaction record or a feature vector. They provide a means of describing the positive correlations in market basket data. They also provide the basis for generating association rules that describe the data set. The support of an itemset is a measure of this positive correlation. The support of an itemset \( X \) is defined as the percentage of transactions or feature vectors in which the itemset appears, or in other words, \( P(X) \). An itemset is called frequent if its support exceeds some user-defined threshold. The problem of mining the frequent itemsets from a data set is a hard one, since, in the worst case, the number of frequent itemsets is exponential in the total number of items or features, as the problem of finding all subsets of set reduces to the problem of finding frequent itemsets.

While frequent itemset mining has exponential complexity in the worst case, many efficient ways have proposed to mine frequent itemsets. The most famous algorithm for mining frequent itemsets is the Apriori algorithm proposed in [7]. The Apriori algorithm exploits the Apriori property that all subsets of a frequent itemset must also be frequent. This allows an algorithm to remove a potentially large number of itemsets from consideration: if an itemset is not frequent, then none of its supersets can be frequent. Since the
introduction of the Apriori algorithm, others have developed alternate methods of generating frequent itemsets. Some of these alternate methods use backtracking techniques, like GENMAX [58] and ZIGZAG [151]. Another method, called ECLAT, partitions the search space into subspaces based on equivalence classes for more efficient computation [159]. After all frequent itemsets have been found, association rules can be generated. Association rules have the form $X \rightarrow Y$, where $X$, $Y$, and $X \cup Y$ are all frequent itemsets. The two basic measures of the usefulness of an association rule are its support ($P(X \cup Y)$), and its confidence ($P(Y|X)$). Another measure of the interestingness of a rule is its J-Measure, defined as:

$$J(X \rightarrow Y) = P(X) \left( P(Y|X) \log \frac{P(Y|X)}{P(Y)} + P(\neg Y|X) \log \frac{P(\neg Y|X)}{P(\neg Y)} \right)$$  \hspace{1cm} (2.5)$$

This measure, proposed by Smyth and Goodman [145], characterizes the amount of information contained in the rule.

One of the major drawbacks of normal association rules and frequent itemsets is that they only describe positive correlations. Researchers have presented methods for finding negative associations and, more generally, correlated item sets. In [137], a method for finding negative association rules was presented. The basic idea is that item sets whose support is much smaller than the expected support form the basis of negative association rules. In [144], a method of generating rules that identify correlations is proposed. The method takes into account both the presence and absence of items in an itemset as a basis for generating rules. The significance or interestingness of the rules is measured using the chi-squared test.

Chakrabarti et al. [26] present a method for discovering interesting patterns (anomalies) in market basket data. The interesting patterns they are searching for are changes
in inter-item correlations over time. They use an information-theoretic approach to discover interesting patterns using the fact that more interesting (rarer) patterns have a longer description length.

Padmanabhan and Tuzhilin [116] propose three algorithms (FilterMinZoomUR, MinZoominUR, and MinZoomUR, and to directly find the minimal set of unexpected rules in a data set (as opposed to performing post-processing on the results of another rule-mining algorithm). A rule is unexpected if it violates a prior belief about the database. More formally, a rule \( A \rightarrow B \) is unexpected with respect to a belief \( X \rightarrow Y \) if \( B \land Y \) is false, the support of \( X \cup A \) meets a minimum threshold, and the rule \( (A \cup X) \rightarrow B \) holds (inferring \( (A \cup X) \rightarrow Y \) does not hold). Since most rules should hold with respect to the belief, the rule \( A \rightarrow B \) is unexpected. By finding the minimal set of unexpected rules, redundant rules and rules that can be inferred from the minimal set can be ignored. As the authors note, the quality of the unexpected rules found depends on having a comprehensive set of beliefs, which can be generated using a method proposed earlier by the authors [115].

Yairi et al. [154] investigate fault detection using spacecraft house-keeping data. Existing approaches involve monitoring the limits of various sensors, simulating system behavior to validate the sensor values, or building expert systems using human knowledge. However, these approaches are insufficient, since there may be combinations of sensor values, all within valid limits, that are anomalous, the design and execution of simulations may be prohibitively expensive, or the expert system’s rules may be incomplete, since humans cannot predict all possible conditions. Their proposed method involves discovering representative sequences and patterns from the data, and using association rule mining to
build a collection of rules describing the normal states of the spacecraft. They select appropriate rules by picking those with the highest J-measure (see Equation 2.5). Anomalies are detected if there is a large change in the confidence or support of a rule over time.

The STUCCO algorithm provides a means of mining contrast sets [16]. A contrast set is an itemset whose support varies significantly across different data sets. These itemsets are useful for characterizing the difference between two groups and are therefore useful for classification. In STUCCO, the contrast sets are found using a hypothesis test, where the null hypothesis is that the supports of a potential contrast set are equal across the different data sets. A chi-squared test is performed, and for suitably small p-values, the null hypothesis can be rejected, and the itemset can be confirmed as a contrast set.

2.3 Supervised Machine Learning Methods

Machine learning is a branch of artificial intelligence concerned with constructing computer programs that automatically learn from examples. More precisely, it involves using example data points to learn a target function that maps the points to class values [103, 65]. Once such a function has been learned, it can used to perform classification or prediction. There are two basic types of machine learning: unsupervised and supervised learning. Unsupervised learning involves learning the target function from examples that do not have class labels and it is usually performed using clustering techniques, such as those discussed previously in Section 2.1.1. In unsupervised approaches, it is difficult to determine how closely the learned function approximates the true (optimal) classification function, as there is no ground truth with which to compare. In this section we focus on supervised learning techniques, which involves learning the target function from examples that are labeled with the classes to which they belong. There has been extensive research in the field of
supervised learning. A wide variety of classification techniques have been examined over the years [93]. Such techniques include decision trees [104], neural networks [128], and Bayesian classifiers [134], among others.

Lee and Xiang examine anomaly detection from the standpoint of information theory in [91]. The authors examine several different data sets using the information theoretic measures entropy, conditional entropy, relative conditional entropy, and information gain. The authors note that as the entropy of the data increases, the complexity of the model needed to describe it also increases, and that more complex models are less useful when detecting intrusions. They use conditional entropy and information gain as guides for selecting the best attributes to include in the model of the data.

A method using trees to detect anomalies of a temporal nature is presented in [140]. In this method, a node represents an action, an edge represents the probability that one action will be followed by another, and a path represents a pattern. If a pattern cannot be matched to one in the tree, then this is indicative of an anomaly.

Bayesian classifiers use prior knowledge of the data and classes to perform classification. The classifier is derived from Bayes’ rule and has the form:

\[
Class(X) = \arg \max_{c_i \in C} \left( \frac{P(X|c_i)P(c_i)}{P(X)} \right)
\]

where \(C\) is the set of all classes and \(X\) is the instance to be classified. A full Bayes classifier is an optimal classifier, meaning that no classifier trained on the same data can do better on average [103]. There is a spectrum of Bayesian classifiers, ranging from naïve classifiers (where all attributes are assumed to be independent) to full classifiers (where all attributes are assumed to have dependencies between them) [134]. Fuller classifiers, though more accurate, are relatively more difficult to construct than their more naive counterparts, since they require the estimation of a relatively large number of joint probabilities. Naïve
classifiers are more commonly used because they are computationally more efficient. However, it has been shown that naïve classifiers can perform very well even when the attribute independence assumption is violated [42].

A method for detecting disease outbreaks via anomalous patterns using Bayesian networks is presented in [153]. The central problem is trying to detect an anomalous disease outbreak against a background of other non-anomalous or regular disease outbreaks (for example, trying to detect an outbreak of SARS during the onset of flu season). To address this problem, a baseline model is constructed by sampling data from a Bayesian network constructed from all data prior to the past 24 hours. Rules are created from this sampled data and rule scores are set based on how well the rules apply to the data from the past 24 hours.

One simple method of detecting anomalies is to use probabilistic models. Such models were used by Mahoney and Chan for detecting network intrusions [99]. Their approach, called ALAD (“application layer anomaly detection”) estimates the probability distribution of a given attribute in network traffic data, and assumes that values of the attribute with relatively low probabilities are indicative of an anomaly. They later expanded this idea into a more general approach for generating rules that describe normal data called LERAD (“learning rules for anomaly detection”) [97, 98]. LERAD first generates candidate rules from a sample of the data by finding common attributes in pairs of instances in the sample. Then the rule set is pruned such that only a minimal set of rules that covers all instances in the sample.

In [107], a technique using discriminant analysis is presented. The technique uses the Fisher linear discriminant to project all data points onto a single vector. Then the vector is divided into two segments that divide the normal data from the abnormal data.
2.4 Statistical Outlier and Anomaly Detection Methods

Statistical methods for abnormality detection assumes that the data fits a certain distribution or probability model [65]. Signature-detection based techniques that build models for known abnormalities (e.g. Bayesian Classification) are discussed above in Section 2.3. In this section we focus on statistical anomaly and outlier detection methods.

Many statistical approaches assume that the majority of the data fits some distribution \( D \) and then flags those points that are outliers with respect to the model using discordancy tests [65, 14, 68, 132]. However, it has been noted that these approaches are typically univariate, and do not scale well for high-dimensional data [65, 85, 117]. Furthermore, it may be difficult to characterize the distribution \( D \) to use in the tests. However, in some domains and approaches statistical approaches can still be useful. For example, Hill examines Benford’s Law in [70], which states that in a list of numbers, the mantissas of their logarithms are all each equally likely. This is useful in detecting fraud in accounting data in [106], since it is likely that fabricated data does not conform to Benford’s Law.

Bolton and Hand use two statistical approaches for fraud detection [22]. The first approach is peer group analysis [21], which examines the behavior over time of a target object relative to peers that are similar to it. The standardized distance of a target to the centroid of its peer group. This distance is a t-statistic, and targets that act anomalously with respect to their peer groups will have a large value for this statistic. The benefits of this approach is that it examines the targets locally: while a target’s behavior may not be globally anomalous, it may locally be so since the target is acting abnormally with respect to its peers. The second approach they use is called break point analysis [141]. This examines the behavior of a target object over time using a fixed-width sliding window. Anomalies are detected by comparing statistics of the newer data in the window against the statistics of the older
An anomaly is flagged if there is a significant difference in the value of the “old” and “new” statistics. Unlike peer group analysis, no comparisons are made to other objects in the data set, and so the data for the object does not need to be collected at fixed time points. However, since break point analysis ignores the information provided by other objects, the power of the tests used to compare the “old” and “new” statistics is much lower.

2.5 Intrusion Detection Systems

Abnormality detection is very important in network traffic data. In this domain, abnormalities can be indicative of network failure or malicious behavior. Since almost all organizations are dependent on the Internet to survive, it is extremely important to track down the cause of these abnormalities to minimize the threats to the security and integrity of an organization’s data. The domain of network traffic data presents several constraints on the nature of the abnormality detection approaches that one can use. Foremost among these are the fact that the amount of data to be analyzed is large and streaming, and the fact that the characteristics of both the normal and abnormal data changes over time.

A comparison of various anomaly detection schemes is presented in [89]. Its focus is on how well different schemes perform with respect to detecting network intrusions. The authors use the 1998 DARPA network connection data set to perform their evaluation, which is the basis of the KDDCup 1999 data set we use to test many of our approaches intrusion detection [69]. They found detection rates ranging from a low of 52.63% for an approach based on the Mahalanobis distance to a high of 84.2% for an approach using support vector machines.

Signature detection methods are better understood and widely applied in the domain of network intrusion detection. They are used in both host based systems, such as virus
detectors, and in network based systems such as SNORT [131] and BRO [125]. These systems use a set of rules encoding knowledge gleaned from security experts to test files or network traffic for patterns known to occur in attacks. A limitation of these systems is that as new vulnerabilities or attacks are discovered, the rule set must be manually updated. Another disadvantage is that minor variations in attack methods can often defeat such systems. These limitations can only be surpassed through the use of anomaly detection techniques.

Since the signatures of network attacks can change too quickly to build an effective signature detector, anomaly detection approaches are more useful and interesting. However, what is considered normal is more abstract and ambiguous. Rather than finding rules that characterize attacks, we wish to find rules that characterize normal behavior [47]. Since what is considered normal could vary across different environments, a distinct model of normality can be learned according to the typical behavior in each environment. Much of the research in anomaly detection uses the approach of modeling normal behavior from a (presumably) attack-free training set. Because we cannot predict all possible non-hostile behavior, false alarms are inevitable. Forrest et al. [48], making the connection between anomaly detection systems and biological immunology, found that when a vulnerable Unix system program or server is attacked (for example, using a buffer overflow to open a root shell), that the program makes sequences of system calls that differ from the sequences generated during normal operation [47]. Forrest uses \( n \)-gram models (sequences of \( n = 3 \) to 6 calls), and matches them to those sequences observed during training. A score is generated when a sequence observed during detection is different from those observed in the training phase. Other models of normal system call sequences have been used to detect anomalies, such as finite state automata [139] and neural networks [52, 53]. Lane and
Brodley use instance-based methods [88] and on sequence similarity [87], and Sequeira and Zaki [142] use clustering methods for detecting anomalous user commands.

Current network anomaly detection systems such as NIDES [8], ADAM [13], and SPADE [38] model only features of the network and transport layer, such as port numbers, IP addresses, and TCP flags. Models built with these features could detect probes (such as port scans) and some denial of service (DOS) attacks on the TCP/IP stack, but would not detect attacks of the type detected by Forrest, since the exploit code is transmitted in the packet payload, which is ignored by such systems. Most current anomaly detectors use a stationary model, where the probability of an event depends on its average rate during training, and does not vary with time. However, using the average rate could be incorrect for many processes. Paxson and Floyd [126] found that many network processes, such as the rate of a particular type of packet, have self-similar (fractal) behavior. Events do not occur at uniform rates on any time scale. Instead, they tend to occur in bursts. Hence, it is not possible to predict the average rate of an event over a time window by measuring the rate in another window, regardless of how short or long the windows are. An example of how a stationary model fails in an anomaly detector would be any attack with a large number of events, such as a port scan or a flooding attack. While most research in intrusion detection has focused on either signature detection or anomaly detection, most researchers have realized that the two models must work hand-in-hand to be most effective [13, 12].

2.6 Limitations of Past Approaches

One limitation of these approaches to detecting abnormalities is that they are more concerned with detecting abnormalities than in avoiding them. In domains such as network intrusion detection, avoidance of or protection from intrusions is much more valuable than
detecting the intrusions after the fact, since the host or network has by then been compromised or damaged. In order to actively protect a host or network, an intrusion detection system must be able to detect the intrusions early and quickly. This requires the underlying abnormality detection schemes to be sensitive and to not be computationally intense. To answer these limitations, we have developed techniques for detecting intrusions at the level of the network interface card, which has the effect of protecting the host machine from the detected intrusions. This work is presented in Chapter 3.

The above approaches are also limited by the fact that many of them are designed to operate in a centralized setting. These approaches implicitly assume that distributed data sets can be combined at some central location for processing. However there are many reasons why this assumption is poor: The size of the combined data set may be too large to store, the cost of transmitting data sets may be too high, or legal concerns (e.g. customer privacy) may prevent two organizations from sharing their data. Only recently have researchers begun to examine distributed techniques for abnormality detection. For example, Locasto et al [94] examine collaborative techniques for network intrusion detection. If organizations can collaborate, then each can build a better model of global network activity, and construct more precise models of attacks (since they have more data from which to estimate the model parameters). This allows for better characterization and prediction of attacks. In Chapter 4 we present our own techniques for building global models of distributed data, in this case find global frequent itemsets in distributed data sets that are constantly being modified. In Chapter 5 we present a technique called LOADED for finding outliers in distributed streaming data sets.

In many applications, there are many resource constraints. Examples of such constraints in network intrusion detection include time constraints (examination of packets should not
adversely affect the network throughput), and memory and processor constraints (a detection system running on a host machine should leave enough memory and processor time for the user to run his or her applications). Embedded network intrusion detection systems not running on the host machine may have even more serious memory and processor constraints. Distributed abnormality detection techniques can be constrained by the bandwidth and latency of the networks they utilize. Some of the techniques mentioned in this section (especially the clustering techniques presented in Section 2.1.1) have memory and time requirements that do not scale well with the size of the data set. Abnormality detection in resource-constrained environments must use light-weight, approximate, or sampling techniques. The intrusion detectors operating on the network interface card that we present in Chapter 3 are constrained by the relatively small memory and slow processor on the card. Our distributed approach for mining frequent itemsets presented in Chapter 4 is constrained by the bandwidth and latency of the underlying network. The LOADED approach to anomaly detection presented in Chapter 5 must identify outliers in a single pass of the data, and has parameters for adjusting its performance to meet the resource constraints of the environment in which it runs.
CHAPTER 3

RESOURCE-DEPENDENT ABNORMALITY DETECTION ON NETWORK INTERFACE CARDS

As mentioned previously, the goal of an intrusion detection system is to detect inappropriate, incorrect, and unusual activity on a network or on the hosts belonging to a local network by monitoring network activity. Most work on signature and anomaly detection has relied on detecting intrusions at the host processor level, even those that occur only at the network layer and below. A problem with these approaches is that even if an anomalous activity or intrusion is detected, one is often unable to prevent the anomalous packets from causing havoc by disrupting the system and over utilizing the CPU (for example, by denial-of-service attacks). This work targets the emerging application of network intrusion detection using Network Interface Cards (NICs). Specifically, we study a novel architecture for network intrusion detection using NICs, and empirically evaluate its feasibility.

The primary role of NICs in computer systems is to move data between the system’s components and the network. A natural extension to this role would be to actually police the packets forwarded in each direction by examining packet headers and simply not forwarding suspicious packets. The rationale for NIC-based intrusion detection coupled with conventional host-based intrusion detection can be stated as follows: First, functions such as signature-based and anomaly-based intrusion detection can be performed on the
NIC, which has its own processor and memory. This makes the system virtually impossi-
ble to bypass or compromise as can be the case with software-based systems that rely on
the host operating system to function. Second, if the host is loaded (with other programs
running simultaneously), an intrusion detection system that relies on the host’s processing
capabilities may be slowed down, thereby adversely affecting the bandwidth available for
acceptable network transmissions. A NIC-based strategy will not be affected by the load
on the host and therefore will not suffer the same slowdown. Third, there is a potential
to naturally handle the scalability problem associated with centralized intrusion detection
systems, since each individual NIC can handle the in-bound and out-bound traffic of the
particular processor/local area network it is concerned with, thus effectively distributing the
work concerned. Fourth, NIC-based strategies provide better coverage (one-to-one map-
ing between hosts and intrusion detection systems) and functional separation (e.g. internal
NICs can detect port sweeps while NICs at the firewall can detect IP address sweeps). Fi-
nally, the NIC-based scheme is inherently flexible, dynamically adaptive, and can work in
conjunction with a host-based intrusion detection system. A host-based intrusion detection
system can download new rules/signatures into the NIC on the fly, making the detection
process adaptive. Figure 3.1 represents the overall architecture for NIC-based security.

The above advantages notwithstanding, the current disadvantage to NIC-based intru-
sion detection is that the processing capabilities on the NIC are severely limited and the
memory sub-system is much smaller when compared to those available on the host ma-
chine. Therefore, the task of implementing algorithms on the NIC presents several new
challenges. For example, the NICs we use are not capable of performing floating point
operations. As a result, we are forced to eliminate those operations or resort to estimates
Figure 3.1: Motivation for adding NIC-based security

based on fixed-point operations. We also need to limit the impact on bandwidth and latency for non-intrusion packets. Given this fact, the question now becomes how best to use the NIC’s processing capabilities for intrusion detection? This is the key question that this work seeks to answer.

3.1 Related Work in NIC-based Computing and Data Stream Processing

Recently there has been a fair amount of activity in the area of NIC-based computing. Most closely related to our work is the use of NICs for firewall security [105]. The idea is to embed firewall-like security at the NIC level. Firewall functionality such as packet filtering, packet auditing, and support for multi-tiered security levels has been proposed. While most of these ideas are in their infancy, some simple ideas have been commercialized (e.g.
3Com’s embedded firewall). The past few years have also witnessed the emergence of application domains wherein data elements arrive in the form of a continuous stream. Examples of such streaming data sets include stock tickers and click streams. Often, these data streams can be characterized as infinite streams that have no pre-defined size. This requirement has motivated online processing of data streams as and when they arrive and by developing algorithms that bound memory usage. Existing algorithms have been redesigned to process the stream in one pass using a summary structure, which stores an approximate representation of the data stream in memory [41, 100, 61, 101]. When processing network data for network intrusion detection, we essentially process each incoming packet one at a time, and never get a second look. This makes data stream related research highly relevant to NIC-based network intrusion detection. However, the key difference is that NIC-based stream processing is even more constrained when it comes to processing capabilities and memory usage. As a result, several data stream processing algorithms are unsuitable for network intrusion detection under real-time processing requirements.

3.2 Algorithms

In this section we describe the two basic types of algorithms (anomalous client detectors and hybrid models) we evaluated. Each of these algorithms were implemented both on the NIC as well as on the host processor. However, they have all been designed keeping the memory and processing limitations of the NICs in mind. NICs typically have a relatively small amount of memory, and a slow, limited processor. Any algorithm designed to run efficiently on a NIC must take these facts into account. For example, the NICs on which the following algorithms were implemented have no floating-point capabilities, and so any
floating-point operations must either be eliminated, or estimated using fixed-point implementa-
tions. Likewise, array sizes must be constrained so that they fit within the limited memory available on the NIC.

3.2.1 Anomalous Client Detectors

**P(SrcIP | DstIP) Anomalous Client Detector**

```plaintext
function DetectAnomalousClient(SrcIP, DstIP)
begin
SDTable: A two-dimensional hash table with one axis indexed by DstIP and the other by SrcIP, which holds the number of packets each DstIP receives from each SrcIP.
DstTable: A one dimensional table holding the number of packets received by each DstIP
H: Hash function for the SrcIP axis in the SDTable
DstTable[DstIP]++;
SDTable[DstIP][H(SrcIP)]++; /*conflict resolution is handled using standard approaches*/
if ((SDTable[DstIP][H(SrcIP)] / DstTable[DstIP]) > threshold(DstIP))
    return “Normal Client”;
else
    return “Possible Anomalous Client”;
end
```

Figure 3.2: The $P(SrcIP | DstIP)$ anomalous client detector

The $P(SrcIP | DstIP)$ anomalous client detector algorithm (see Figure 3.2) is loosely based on one of the models used in the non-stationary application layer anomaly detection (ALAD) algorithm proposed by Chan and Mahoney [99] as described in Section 2.3. The objective of this model is to determine the anomaly score of a given packet based on the quantity of the previous interactions between a particular client and the particular destination host in question. The anomaly score is based on the value of $P(SrcIP | DstIP)$, the probability that a client machine $SrcIP$ connects to a given host $DstIP$. The set of normal clients for a host are those for which $P(SrcIP | DstIP)$ is greater than some
threshold A value of $P(SrcIP \mid DstIP)$ that is lower than the threshold may be indicative of suspicious behavior, since there is a low probability of $SrcIP$ connecting to $DstIP$.

The model is capable of detecting two types of anomalous behavior. One behavior is when a new or existing client (someone the system has not seen before) attempts to connect to a host that it has not connected to before. The second behavior is when the amount of a client’s interaction with a particular host radically changes over time. Note that the system can only detect anomalies in the quantity of the interactions between the host and the client; it cannot detect anomalies in the character of the interactions.

To model the first scenario we need to model an anomaly score or surprise factor, for such a scenario unfolding. Basically, a new client accessing a well known world wide web portal is less surprising than a new client accessing a particular internal machine that is typically accessed only by a handful of trusted client machines. To model this surprise factor, we rely on incrementally keeping track of a threshold function that is inversely proportional to the entropy of accesses to a particular destination host. The entropy for the web based example (low threshold value) would be much higher than the entropy for the trusted client example (high threshold value). This threshold function is therefore dependent on the number of different client connections for a given host as well as the frequency of client connections. We incrementally maintain the threshold by adopting ideas from our previous research [121]. Note that a destination host which receives connections from a large number of sources (clients), is more likely to be accessed by a new source (client), than by a host which typically receives connections from just a few sources (clients). To model the second scenario, we monitor the rate of change of $P(SrcIP \mid DstIP)$ over temporal windows. This would allow us to detect large fluctuations in the conditional probabilities which would trigger a possible anomaly alarm.
The basic algorithm stores information in a set of hash tables. Again we use hash tables so that one can save on memory utilization, at a (hopefully small) cost to accuracy of the model. Upon receiving a new packet, it simply computes the conditional probability \( P(SrcIP \mid DstIP) \) and if this conditional probability is less than the threshold determined from the training data, then the algorithm considers the source host to be an anomalous client. In the current implementation of the algorithm the thresholds are adjusted according to the Gini index instead of entropy, as the computation of a Gini index is better suited to the capabilities of the NIC than the computation of entropy is; computing a Gini index requires squaring probabilities, whereas computing an entropy value requires taking logarithms of probabilities, a function that is not efficient on the NICs.

**P(SrcIP | DstPort, DstIP) Anomalous Client Detector**

```plaintext
function DetectAnomalousClient(SrcIP, DstPort)
begin
SDDTable: A three-dimensional hash table with one axis indexed by DstPort, one by DstIP, and the third by SrcIP, which holds the number of packets each (DstPort, DstIP) pair receives from each SrcIP.
DstTable: A two dimensional table holding the number of packets received by each (DstPort, DstIP) pair
H: Hash function for the SrcIP axis in the SDDTable
DstIP = Localhost.IPAddress;
DstTable[DstPort][DstIP]++;
SDDTable[DstPort][DstIP][H(SrcIP)]++; /*conflict resolution is handled using standard approaches*/
if ((SDDTable[DstPort][DstIP][H(SrcIP)] / DstTable[DstPort][DstIP]) > threshold(DstPort, DstIP))
return “Normal Client”;
else
return “Possible Anomalous Client”;
end
```

Figure 3.3: The \( P(SrcIP \mid DstPort, DstIP) \) anomalous client detector
The $P(SrcIP \mid DstPort, DstIP)$ anomalous client detector algorithm (see Figure 3.3) is also loosely based on one of the models used by Mahoney and Chan [99]. The objective of this model is to determine the anomaly score of a given packet based on previous interactions between a particular client and the particular service (port) it is accessing on a given host. This model is designed to be used in a distributed manner: this algorithm runs on the NIC of each host in the network, instead of running on the central firewall NIC. As such, the value of $DstIP$ is constant (it is the IP address of the host), and so the hash table is only two-dimensional. Running this algorithm on the host NICs leaves the firewall NIC free to run other detectors. This algorithm can operate on the firewall NIC, but its memory requirements are much higher, as the hash table must be three-dimensional since it must take into account the value of $DstIP$, which it takes directly from the packet header.

### 3.2.2 Hybrid Models For Anomaly Detection

We have also developed hybrid models that use resources at both the host and NIC level. Operations that cannot be performed at the NIC level due to memory or speed constraints can be performed at the host level, and their results can be sent to the NIC to aid in detection. We discuss two hybrid approaches to anomaly detection: one that uses frequent itemsets, and one that uses clustering.

#### Anomaly Detection Using Frequent Itemsets

This algorithm uses the concept of frequent itemsets to discover anomalies (see Figure 3.4). This is a hybrid system: The frequent itemsets are computed at the host level and periodically passed down to the NIC, since the NIC itself does not have sufficient resources to compute frequent itemsets. The frequent itemsets are computed using features found in the headers of packets sampled by the host machine.
The intuition behind this algorithm is that since intrusions occur only rarely, by sampling the packets to find the frequent itemsets, few if any intrusion packets will be incorporated into the model. Also, many intrusion packets have little in common with normal packets, and so the number of frequent itemsets found in an intrusion packet will be relatively small. Finally, the size of those itemsets will be relatively small since small itemsets usually have high supports and so these itemsets are shared by all packets, whether they are intrusions or not. Hence, in the algorithm, two thresholds (\(\text{CountThreshold}\) and \(\text{SizeThreshold}\)) are used to determine which are intrusion packets and which are not. \(\text{CountThreshold}\) separates those packets in which only a few itemsets are found from those in which many are found. \(\text{SizeThreshold}\) separates those packets with itemsets that are, on average, smaller from those with larger itemsets.
function ClusterPacket(Clusters C, Packet P)
begin
    Max: The maximum goodness value for clustering
    Threshold: The threshold value for declaring a packet as part of a cluster
    Max := max{H(c, P)|c ∈ C}
    if (Max < Threshold)
        declare anomaly;
end

Figure 3.5: The clustering-based anomaly detection algorithm

Anomaly Detection using Clustering

The packet clustering algorithm (see Figure 3.5) also takes a hybrid approach to NIC-based intrusion detection. The host computes a set of initial clusters based on a random sample of packets from the flow. Then these initial clusters are sent to the NIC. The algorithm running on the NIC will then either assign each packet to one of these clusters or if the packet is not similar enough to any current cluster, the algorithm will declare this packet as a anomaly and assign an anomaly score to it. We use a heuristic function $H$ (see Figure 3.6) which gives the anomaly score based on the similarity and dissimilarity measures of the cluster and the packet. Typically the number of clusters formed is very small, so this part of the algorithm can easily be implemented on the NIC. To be up-to-date, at regular intervals the host takes new random samples and recomputes the clusters if needed. Since the network data used has both categorical and continuous attributes, we use a hybrid version of ROCK [62] to cluster the packets. Our preliminary experiments show good results in this direction.
3.3 Experimental Results

In this section we detail the experimental results we obtained. We first start by describing the experimental setup (the machine configurations, data sets used and evaluations performed) for each of the three algorithms. We then examine both the qualitative and quantitative aspects of the algorithms. The quantitative study focuses on the comparing the host-based and NIC-based schemes while the quality study focuses on the quality of the algorithms as measured by the accuracy of detection.

3.3.1 Setup

All host-based experiments, unless otherwise noted, were evaluated on dual-processor 300 MHz Pentium II machines with 128 MB of memory. All NIC-based evaluations were done on Myrinet NICs, each of which had a 66Mhz LANai 4 processor and 1 MB of memory.
To test each algorithm, we used both synthetic and real data sets. The synthetic data sets were generated using a program available at http://www.cis.ohio-state.edu/~otey/NIC/. The real data set is the 1999 DARPA Intrusion Detection Evaluation data set. The size of the packets, unless otherwise noted, is 2048 bytes. Each packet is composed of four protocol bits, four flag bits, four source IP bytes, four destination IP bytes, two source port bytes, two destination port bytes, one TTL byte, two packet size bytes and 2032 bytes for the data. We set it up so that the network packet generator sends roughly 1 packet every 0.001 seconds for a net theoretical throughput of 2MB per second. However, the actual measured throughput is 1796 KB per second.

The synthetic data set we use to test the anomalous client detectors focuses solely on the source and destination IP addresses. All other fields are filled with random numbers. The data set we used consisted of 256 local hosts, the set of valid clients per host ranged from 0-64, and the total set of clients was the remaining valid IP addresses on the Internet (approximately 4.3 billion hosts). The first hundred thousand packets were messages from valid clients (the training data) and the next 1 million packets are testing packets, about 90% of which come from the set of valid clients. For other experiments, we use the first three weeks of tcpdump data from the DARPA data sets. Weeks one and three contain normal network traffic. Week two contains four days of data containing 34 intrusions (Tuesday’s tcpdump file is unreadable). For the anomalous client detectors we used the data from weeks one and two to do testing, and for the frequent itemset-based detector we used weeks one and three to build the itemsets and week two to test it.
3.3.2 Results

Before detailing the experimental results on performance, we wanted to quickly get a sense for the individual resource requirements (memory and computational) of the different algorithms. We first compared the performance of the host-based and NIC-based strategies using the synthetic data set. The results are documented in Figure 3.7. The X-axis documents variation in host load. The Y-axis is the net throughput of valid packets (allowed to filter through). Recall that the maximum throughput is limited by the incoming data rate which is 1796 KB per second. For the $P(SrcIP \mid DstIP)$ anomalous client detector algorithm we notice that the NIC-based and Host-based strategies perform comparably until the host becomes overloaded. The NIC-based strategy is unaffected by the load and performs at a relatively constant rate allowing for a 95% throughput efficiency.

Figure 3.7: Quantitative comparison of NIC-based and host-based intrusion detection
Figures 3.8 and 3.9 show how different algorithms fared with the DARPA data set. In all, 18 of the 34 intrusions present in week two were detected. Figure 3.8 show the results for different versions of the anomaly detectors as well as the frequent itemset-based detector. The $P(SrcIP \mid DstIP)$ Window algorithm is a modification of the the $P(SrcIP \mid DstIP)$ algorithm that employs a sliding window technique that periodically removes older packets from the hash table. The $P(SrcIP \mid DstIP)$ Sample algorithm is another modification that only examines a sample of the packets. In this case we sampled 25% of the packets. The $P(SrcIP \mid DstPort, DstIP)$ algorithm is implemented here in a non-distributed fashion: We use a three-dimensional hash table instead of a separate table for each internal host. For all of these detectors, the threshold used was a constant value of 0.000015. The frequent itemsets were generated from a 5% sample of the weeks one and three data with a minimum support of 5% and a minimum itemset size of 4. CountThreshold was set to 5 and SizeThreshold was set to 4.05.

In Figure 3.8 one can see that the first four algorithms perform fairly well as far as false positives are concerned. The Sample algorithm does the best in this respect, though it only detects 9 intrusions. In the Venn diagram in Figure 3.9 one can see which intrusions were detected by which algorithms. It is worth noting that the frequent itemset algorithm can only detect denial-of-service (DOS) and probing attacks, since these attacks produce anomalies in the packet headers. Generally, user-to-root (U2R) and remote-to-local (R2L) attacks cannot be detected by this algorithm since their signatures are concealed within the payload portion of the packet, which is ignored by the algorithm. The other algorithms can detect such intrusions, however, because many intrusions originate from a machine that has had little previous contact with the host. Therefore the probabilities will be smaller and the packets will be marked as intrusions.
### 3.4 Summary

We present and evaluate a NIC-based network intrusion detection system. We consider embedding both signature detection algorithms as well anomaly detection algorithms in our evaluation.

The quantitative improvements we achieve with this approach relies on the fact that the operating system of the host does not have to be interrupted with the detection process. Thus on heavily loaded hosts admissible network traffic proceeds at a consistent rate provided the computational and memory resources of the NIC is not stretched. Our preliminary empirical results bear this out. A key element in understanding the trade-offs involved is the amount of computation and memory utilization involved in the programs. The larger the computation cost, the better the performance of a purely host-based approach.

From the qualitative angle, the downside of using simplified algorithms is that the quality and detection rate are hampered. However, the benefit of having the NIC do the policing is that it can actually prevent network-based intrusions from wrecking havoc on host systems. Since the intrusive packet, if caught, never reaches the host operating system, this approach can detect and prevent, unlike host-based systems. In effect, the NIC acts as a
Figure 3.9: Different intrusions are detected by different algorithms

basic shield for the host. If the NIC cannot catch up with the rate the packets are arriving, it can begin dropping the packets (this is the default behavior in the Myrinet NICs), as this may be indicative of a denial-of-service attack. If the NIC were to become overwhelmed by a such an attack, the host would be spared from it. We would prefer to sacrifice only the NIC to the attack rather than the entire host machine.

However, from a technology perspective we are not very far away from 1GHz NIC processors (with appropriately larger memory). With those projected systems we anticipate that NIC-based intrusion detection will do better both from a quantitative standpoint and from a a qualitative standpoint (as we will be able to use less-restrictive algorithms).
CHAPTER 4

DISTRIBUTED APPROACHES FOR ANOMALY DETECTION
USING FREQUENT ITEMSETS

In the previous chapter we examined abnormality detection as it applied to detecting network intrusions at the NIC level. In that case our detection techniques faced two constraints: the resources available on the NIC and the streaming nature of the data. In this chapter we examine another approach for anomaly detection. However, in this instance we are faced with a different abnormality detection problem and a different set of constraints. Instead of NIC-based abnormality detection, in this chapter we concern ourselves with large distributed databases of transactional data. As such, the first of these constraints is the fact that the data must be processed in a distributed manner. The second constraint is that the data is being updated at the different rates at different sites, which could lead to problems of concept drift. These constraints require us to develop anomaly detection techniques to not only build a global model of normality for the distributed data, but to also proceed to detect anomalies of a distributed nature.

Why should we concern ourselves with distributed processing of data? As mentioned previously, data mining in general is a means of automatically analyzing large amounts of data to find pieces of interesting information. In the domain of abnormality detection, these pieces of interesting information are the abnormalities themselves. In many cases,
however, the data we wish to analyze may be so large that standard sequential approaches
to abnormality detection may not be able to efficiently analyze the data. Efficient pro-
cessing then requires parallel or distributed techniques. Furthermore, because of societal
factors, market factors, or the inherent distributed nature of data collection, the data to be
analyzed is often spread across a wide geographical area. For example, at The Ohio State
University, network traffic data is collected at seven distributed end points (routers) for
subsequent analysis. To perform abnormality detection in such an environment, one has
two choices: either the data can be shipped to one centralized location to be processed,
or the data must be processed in a distributed manner. The former approach can be very
inefficient, especially if the data collection points are very far apart and the connections
between them are relatively slow, while the latter approach could possibly lose out some
important global information. Distributed data mining is also desirable when one tries to
build independent models of separate data sets. In anomaly detection, one might want to
model different subsets of the data (e.g. data residing at different sites) and compare how
these models differ from that of the complete data set in order to find unexpected patterns
and differences that characterize how the distribution of the data across the different sets
or sites is skewed. The best characterizations use high-contrast patterns to differentiate the
different data sets.

The second problem is that of processing dynamic data sets where the data set is being
modified over time by both additions and deletions. Examples of such data sets are market-
basket and network traffic data sets. In the previous chapter we dealt with dynamic data
sets that were streaming, but in this chapter we generalize to dynamic data sets where the
order of deletions from the data set may not be the same as the order of the additions.
Since the data is dynamic, the statistical properties of the data set are most likely dynamic
as well, which leads to concept drift. An accurate model of such data must be dynamic itself. Suppose we have some model $M_t$ of the data built using an algorithm $A$ at time $t$. A naïve approach to compute a new model at time $t + 1$ would be to re-execute $A$ on the updated data to build model $M_{t+1}$. This approach is not efficient, though, because it ignores the previous information stored in $M_t$, and essentially replicates work that has been done before. An incremental approach that uses $M_t$ and the updated data to build $M_{t+1}$ would be much more efficient.

An important issue when finding abnormalities or data mining in general in distributed databases is to understand the differences between the databases held at each site. An effective way to understand such differences is to find the high-contrast patterns. Since these patterns characterize the differences between data sets, they can also be useful for building classification rules when the data sets belong to separate classes. Furthermore, high-contrast patterns can be useful in detecting anomalies by comparing subsets of a data set (the databases at each site) against the full data set (the global, unified database). Since high-contrast patterns are a characterization of how the data is skewed over the sites, they can indicate abnormal sites, as those sites will either contain patterns not frequently found at the other sites, or have few patterns that are frequently found at the other sites. In the domain of network intrusion detection, an abnormal site may be a machine infected with a virus or a router forwarding network streams that may contain intrusions. In the domain of commercial transaction analysis, an abnormal site may be a store where the customers have different buying patterns or where fraud occurs. High-contrast patterns are key to finding anomalies in distributed databases.
In this chapter we examine an approach to mine frequent itemsets from distributed, dynamic data sets. These frequent itemsets will allow us to build a model of global normality for a distributed data set. Furthermore, this global model can be used in conjunction with the local models to discover high-contrast frequent itemsets that will allow us to find anomalies in a distributed setting. Our work here is based on the ZIGZAG algorithm for incrementally mining frequent itemsets [151].

4.1 Distributed Incremental Frequent Itemset Mining

Frequent itemset mining (see Section 2.2) is a core data mining task, and has many applications to abnormality detection. As frequent itemsets are those sets of items that co-occur in a large number of database records or transactions, they inherently model what is “normal” for a database. Since abnormal records or transactions are unlikely to have much in common with other records or transactions, they are therefore unlikely to contain many frequent itemsets. We utilized this idea in Section 3.2.2 in Chapter 3 for one of our hybrid NIC-based intrusion detectors, and we will utilize the idea again in Chapter 5.

To address the problem of mining frequent itemsets from dynamic data sets, several researchers have proposed incremental techniques [29, 30, 50, 90, 147, 151]. Incremental algorithms essentially re-use previously mined information by combining it with the fresh data to efficiently compute the new set of frequent itemsets. However, the size of the data set could be so large and the rate at which it is being updated could be so high that existing incremental algorithms are ineffective by themselves. Therefore we must also rely on high-performance computing techniques. Furthermore, the database may be distributed over multiple sites, being updated at different rates at each site, which requires the use of distributed asynchronous data mining techniques.
Ideally, one would like to mine frequent itemsets interactively (for example, so that the most recent anomalies can be discovered on demand). To do so, one’s query must be answered in as little time as possible. To achieve this, an algorithm for distributed, incremental mining must take into account the rate at which data arrives and the computational resources available at each site, since these two variables can vary from one site to the next. Additionally, one might like to analyze the skew present in the distributed database, specifically how the supports of the frequent itemsets vary from site to site. For example, in the context of network intrusion detection, one might like to know how models of network traffic vary at different points in a network (e.g. routers). Large differences in the amount of network traffic may indicate suspicious behavior in a particular part of the network. Therefore, the algorithm must be able to determine the high-contrast frequent itemsets.

4.2 Background and Related Work

Often, the size of a data set or the rate at which data is inserted or removed is so large that existing sequential algorithms are ineffective. In these cases, parallel or distributed algorithms are necessary. In [118], Park and Kargupta give an overview of a wide variety of distributed data mining algorithms for association rule mining, classifier learning, collective data mining, and clustering, among others. In particular, there has been much research into parallel and distributed algorithms for mining association rules [156, 64, 119, 124, 158, 155]. Zaki provides an overview of several of these methods and others in [157].

A common approach for mining distributed databases is the centralized one, in which all data is moved to a single central location and then mined. However, such an approach requires a large amount of communication to send all the data to single location, and also
increases the work load of the central site. Another common approach is the *local* one, where models are built locally at each site, and then moved to a common location where they are combined. The latter approach is the quickest but often the least accurate, while the former approach is more accurate but generally quite expensive in terms of time required. In the search for accurate and efficient solutions, some intermediate approaches have been proposed [4, 28, 31, 156]. In [4] three approaches for mining distributed data were proposed. The **COUNT DISTRIBUTION** algorithm is a simple distributed implementation of **APRIORI** [7]. All sites generate the entire set of candidates, and each site can thus independently get local support counts from its partition. At each iteration the algorithm does a sum reduction operation to obtain the *global support counts* by exchanging *local support counts* with all other sites. Since only the support counts are exchanged among the sites, the communication overhead is reduced. However, it performs one round of communication per iteration (note that synchronization is implicit in communication). The **DATA DISTRIBUTION** algorithm generates disjoint candidate sets on each site. However, to generate the *global support counts*, each site has to scan the entire database (its local partition and all remote ones) in all iterations of the algorithm. Hence this approach suffers from high I/O overhead. The **CANDIDATE DISTRIBUTION** algorithm partitions the candidates during each iteration, so that each site can generate disjoint candidates independently of the other sites, but it still requires one round of communication per iteration.

In [156] two distributed algorithms were presented, **PARECLAT** and **PARMAXECLAT**. Both algorithms are based on the concept of equivalence classes. Each equivalence class corresponds to a sub-tree in the search space for frequent itemsets, and they can be processed asynchronously on each site. **PARECLAT** outperforms **DATA, COUNT, and CANDIDATE DISTRIBUTION** algorithms by more than one order of magnitude. **PARMAXECLAT**
outperforms PARECLAT, but it searches only the *maximal* frequent itemsets, instead of all frequent itemsets.

These techniques are devised to scale up a given algorithm (e.g., APRIORI, ECLAT, etc.). Data is distributed (or in some cases, replicated) among different sites and a data mining algorithm is executed in parallel on each site. These approaches do not take into account the possible distributed nature of the data. Some assume a high-speed network environment and perform excessive communication operations. These approaches are not efficient when the databases are distributed over a geographically wide area. The FDM (Fast Distributed Mining) algorithm presented in [28] attempts to cut down on communication between sites. It does this by first having each site mine its local frequent itemsets and then exchanging these so that support counts can be taken across all sites to find the global frequent itemsets. In [138], Schuster and Wolff note that FDM does not scale well as the number of sites increases, and so propose the DISTRIBUTED DECISION MINER algorithm and several variations, which do not assume that each local frequent itemset is potentially a global frequent itemset. Additionally, the DISTRIBUTED DECISION MINER is efficient even in the presence of skewness in the data. Not only is it desirable to reduce the amount of communication involved in distributed mining, it is also desirable to adapt the algorithm to differences in the amount of computational and communication resources at each site.

In [59], three strategies of distributed data mining are examined, based on what information is exchanged between sites. The three strategies are those that move results (MR), move models (MM), and move data (MD). The MD strategy is generally avoided, since it can be costly in terms of communication. The authors propose the Papyrus system, which makes use of all three strategies to different degrees, based on the values of a cost function and
an error function that take into account the cost of transmitting data between nodes and the distribution of data over the nodes.

There has also been work on the problem of incrementally mining frequent itemsets [90, 147, 151, 29, 30, 50] in dynamic databases. Some of these algorithms cope with the problem of determining when to update the current model of frequent itemsets, while others update the model after an arbitrary number of updates [151]. To decide when to update, Lee and Cheung [90] propose the DELI algorithm, which uses statistical sampling methods to determine when the current model is outdated. A similar approach proposed by Ganti et al [50] monitors changes in the data stream. An efficient incremental algorithm, called ULI, was proposed by Thomas et al [147]. ULI strives to reduce the I/O requirements for updating the set of frequent itemsets by maintaining the previous frequent itemsets and the negative border [149] along with their support counts. The whole database is scanned just once, but the incremental database must be scanned as many times as the size of the longest frequent itemset.

4.3 Algorithms

Below we present our approach to mining frequent itemsets from dynamic, distributed databases. We first give a formal definition of the dynamic, distributed frequent itemset mining problem in Section 4.3.1. In Section 4.3.2 we review the sequential and parallel versions of the incremental algorithm ZIGZAG upon which our approach is based. In Section 4.3.3 we describe our distributed and incremental algorithm. We prove that this algorithm generates an accurate global model of frequent itemsets, and also present an upper bound for the amount of communication necessary in the distributed and incremental
mining operation. Finally, we describe approaches for interactively mining itemsets in a distributed setting.

4.3.1 Problem Definition

Let $\mathcal{I}$ be a set of distinct items. Let $\mathcal{D}$ be a database of transactions, where each transaction has a unique identifier ($\text{tid}$) and contains a set of items. A set of exactly $k$ items is called a $k$-itemset. The tidset of an itemset $C$ corresponds to the set of all transaction identifiers ($\text{tids}$) where the itemset $C$ occurs. The support of $C$ is the number of transactions of $\mathcal{D}$ in which it occurs as a subset. The itemsets that meet a user-specified minimum support are referred to as frequent itemsets. A frequent itemset is maximal if it is not subset of any other frequent itemset. Using $\mathcal{D}$ as a starting point, a set of new transactions $d^+$ is added forming $\Delta = (\mathcal{D} \cup d^+)$. Let $s_\mathcal{D}$ be the minimum support used when mining $\mathcal{D}$, and $F_\mathcal{D}$ be the set of frequent itemsets obtained. Let $\Pi$ be the information kept from the current mining that will be used in the next operation (in our case, $\Pi$ consists of $F_\mathcal{D}$). An itemset $C$ is frequent in $\Delta$ if its support is no less than $s_\Delta$. If a frequent itemset in $\mathcal{D}$ remains frequent in $\Delta$ it is called a retained itemset.

In the distributed case, the database $\Delta$ can be divided into $n$ partitions, $\delta_1, ..., \delta_n$. Each partition $\delta_i$ is assigned to a site $S_i$. Let $C.\text{sup}$ and $C.\text{sup}_i$ be the support of $C$ in $\Delta$ (global support) and $\delta_i$ (local support) respectively. Given $s_\Delta$, $C$ is global frequent if $C.\text{sup} \geq s_\Delta \times |\Delta|$; likewise, $C$ is local frequent in $\delta_i$ if $C.\text{sup}_i \geq s_\Delta \times |\delta_i|$. The set of all maximal global frequent itemsets is denoted as $\text{MFI}_\Delta$, and the set of maximal local frequent itemsets at $\delta_i$ is denoted as $\text{MFI}_{\delta_i}$. The task of mining frequent itemsets in distributed and dynamic databases is to find $F_\Delta$. 

57
4.3.2 The ZIGZAG Incremental Algorithm

Many algorithms for mining frequent itemsets are apriori-based and use a common procedure. In this procedure, a set of candidates is generated, then the infrequent ones are pruned, and only the frequent ones are used to generate the next set of candidates. Clearly, an important issue in this task is to reduce the number of candidates that are generated. An interesting approach to reduce the number of candidates is to first find $\text{MFI}_\Delta$. Once $\text{MFI}_\Delta$ is found, it is straightforward to obtain all frequent itemsets (and their support counts) in a single data set scan, without generating infrequent (and unnecessary) candidates. This approach works because of the well-known downward closure property (all subsets of a frequent itemset must be frequent). The number of candidates generated to find $\text{MFI}_\Delta$ is much smaller than the number of candidates generated to directly find all frequent itemsets. The idea of maximal frequent itemsets has been successfully used in several data mining tasks, including incremental mining of dynamic databases [151, 150]. ZIGZAG incrementally computes $\text{MFI}_\Delta$ using the previous knowledge $\Pi$. This avoids the generation and testing of many unnecessary candidates. Having $\text{MFI}_\Delta$ is sufficient to know which itemsets are frequent; their exact support are then obtained by examining $d^+$, $d^-$ and using $\Pi$, or, where this is not possible, by examining $\Delta$.

ZIGZAG employs a backtracking search to find $\text{MFI}_\Delta$. Backtracking algorithms are useful for many combinatorial problems where the solution can be represented as a set $C = \{c_0, c_1, \ldots\}$, where each $c_j$ is chosen from a finite possible set, $\mathcal{P}_j$. Initially $C$ is empty; it is extended one item at a time, as the search space is traversed. The length of $C$ is the same as the depth of the corresponding node in the search tree. Given a $k$-candidate itemset (or a partial solution), $C_k = \{c_0, c_1, \ldots, c_{k-1}\}$, the possible values for the next item $c_k$ comes from a subset $\mathcal{R}_k \subseteq \mathcal{P}_k$ called the combine set. If $y \in \mathcal{P}_k - \mathcal{R}_k$, then nodes in
the subtree with root node \( C_k = \{ c_0, c_1, \ldots, c_{k-1}, y \} \) will not be considered by the backtrack algorithm. Each iteration of the algorithm tries to extend \( C_k \) with every item \( e \) in the combine set \( R_k \). An extension is valid if the resulting itemset \( C_{k+1} \) is frequent and is not a subset of any already known maximal frequent itemset. The next step is to extract the new possible set of extensions, \( P_{k+1} \), which consists only of items in \( R_k \) that follow \( x \). The new combine set, \( R_{k+1} \), consists of those items in the possible set that produce a frequent itemset when used to extend \( C_{k+1} \). Any item not in the combine set refers to a pruned subtree. ZigZag performs a depth-first traversal of the search space, as depicted in Figure 4.1. In this example the minimum support is 30%. The framed itemsets are the maximal frequent ones, while the cut itemsets are the infrequent ones.

![Figure 4.1: Backtrack Trees for Items A and B on \( \Delta \)](image)

The support computation employed by ZigZag is based on the associativity of itemsets, which is defined as follows. Let \( C \) be a \( k \)-itemset of items \( C_1 \ldots C_k \), where \( C_i \in I \). Let \( \mathcal{L}(C) \) be its tidset and \( | \mathcal{L}(C) | \) is the length of \( \mathcal{L}(C) \) and thus the support count of \( C \). According to [58], any itemset is obtained by joining its atoms (individual items) and its
support count is obtained by intersecting the tidsets of its subsets. In the first step, \textsc{ZigZag} creates a tidset for each item in \( d^+ \), \( d^- \), and \( \Delta \). The main goal of incrementally computing the support is to maximize the number of itemsets that have their support computed based just on \( d^+ \) and \( d^- \) (i.e., retained itemsets), since their support counts in \( D \) are already stored in \( \Pi \). To avoid replicating work already done before, \textsc{ZigZag} first verifies if the extension \( I_{L+1} \cup \{y\} \) is a retained itemset. If so, its support can be computed by just using \( d^+ \), \( d^- \), and \( \Pi \), thereby enhancing the support computation process. All these procedures are described in Figure 4.2. Finally, \textsc{ZigZag} employs a top-down enumeration to determine \( F_\Delta \). Each itemset \( C \in MFI_\Delta \) is broken into \( k \) subsets of size \((k - 1)\). This process iterates generating smaller frequent subsets and incrementally computing their support counts until there are no more subsets to be checked, and \( F_\Delta \) was found.

\begin{figure}[h]
\begin{align*}
\textbf{ZigZag}(I_L, R_L, L) \\
&\text{for each } x \in R_L \text{ do} \\
&\quad I_{L+1} = I \cup \{x\} \\
&\quad P_{L+1} = \{y : y \in R_L \text{ and } y > x\} \\
&\quad \text{if } I_{L+1} \cup P_{L+1} \text{ has a superset in } MFI \text{ return} \\
&\quad R_{L+1} = \text{Extend}(I_{L+1}, P_{L+1}) \\
&\quad \text{if } R_{L+1} == \emptyset \\
&\quad \text{if } I_{L+1} \text{ has no superset in } MFI \\
&\quad \quad MFI = MFI \cup I_{L+1} \\
&\quad \text{else} \\
&\quad \quad \text{ZigZag}(I_{L+1}, R_{L+1}, L + 1) \\
\end{align*}
\end{figure}

\begin{figure}[h]
\begin{align*}
\textbf{Extend}(I_{L+1}, P_{L+1}) \\
&\quad R = \emptyset \\
&\quad \text{for each } y \in P_{L+1} \text{ do} \\
&\quad \quad y' = y \\
&\quad \quad \text{if } I_{L+1} \cup \{y\} \text{ is a retained itemset} \\
&\quad \quad \quad \sigma_\Delta(y') = \sigma_D(I_{L+1} \cup \{y\}) + \sigma_{d^+}(I_{L+1} \cup \{y\}) \\
&\quad \quad \quad \quad - \sigma_{d^-}(I_{L+1} \cup \{y\}) \\
&\quad \quad \text{else} \\
&\quad \quad \quad \sigma_\Delta(y') = \sigma_\Delta(I_{L+1} \cup \{y\}) \\
&\quad \quad \text{if } \sigma_\Delta(y') == s_\Delta \\
&\quad \quad \quad R = R \cup \{y'\} \\
&\quad \text{return}
\end{align*}
\end{figure}

Figure 4.2: Incremental Search for Maximal Frequent Itemsets
Parallel Incremental Algorithm

We now consider the problem of parallelizing the search for maximal frequent itemsets in a shared memory model (i.e. a model in which each processor has direct and equal access to the system’s entire memory). An efficient parallel search in this model has two main issues: first, minimizing synchronization (i.e. locks and barriers), and second, improving data locality (i.e. maximizing access to the local cache by minimizing the number of cache misses).

The main idea of our parallel approach is to assign distinct backtrack trees to distinct processors. Note from Figure 4.1 that the two issues mentioned above can be addressed by this approach. First, there is no dependence among the processors, because each backtrack tree corresponds to a disjoint set of candidates. Since each processor can proceed independently there is no synchronization while searching for maximal frequent itemsets. Second, this approach is very efficient in achieving good data locality, since the support computation of an itemset is based on the intersection of the tidsets of the last two generated subsets. To achieve a suitable load-balancing, we first sort all of the workloads \( w_i \) (the work load \( w \) due to tree \( i \)). The workload \( w_i \) is calculated using a correlation-based estimation of the workload involved with processing each backtrack tree. Correlations can be used to generate statistical dependencies for both the presence and absence of items (i.e., elements of the combine set) in an itemset (i.e., a partial solution), and its value can be computed by: \( \frac{\sigma_D(\alpha \cup x)}{\sigma_D(\alpha) \cdot \sigma_D(x)} \), where \( \alpha \) is a partial solution and \( x \) is an item in the combine set. The correlation is proportional to the size of the backtrack tree, which corresponds to the computational workload. This is a key optimization in our algorithm. The \( T \) backtrack trees are assigned to the \( N \) processors by using the scheme of bitonic partitioning [34]. This scheme is based on the observation that the sum of the workload due to tree \( i \) and
\[ w_i + w_{2N-i-1} = T - i - 1 + (T - (2N - i - 1) - 1) = 2T - 2N - 1 \]

We can therefore assign itemsets \( i \) and \( (2N - i - 1) \) as one unit with uniform work \( (2T - 2N - 1) \). If \( T \mod 2N = 0 \) then perfect balancing results. The case \( T \mod 2N \neq 0 \) is handled as described in [34]. In short, the tree with the highest workload is assigned to processor 0, and each tree thereafter is assigned to the processor with the smallest load. When all backtrack trees have been processed, all maximal frequent itemsets have been found.

4.3.3 Distributed Incremental Algorithm

The MFI search employed by ZigZag is very efficient, but it can only be applied when the evolving data set is centralized. Now we will explain how we can extend ZigZag for mining distributed data sets. We first present Lemma 1 [136], which is the basic theoretical foundation of our approach.

**Lemma 1** A global frequent itemset must be local frequent in at least one partition.

**Proof:** Let \( C \) be an itemset. If \( C.sup_i < s_\Delta \times |\delta_i| \) for all \( i = 1, \ldots, n \), then \( C.sup < s_\Delta \times |\Delta| \) (since \( C.sup = \sum_{i=1}^{n} C.sup_i \) and \( |\Delta| = \sum_{i=1}^{n} |\delta_i| \)), and \( C \) cannot be globally frequent. Therefore, if \( C \) is a global frequent itemset, it must be local frequent in some partition \( \delta_i \). □

In the first step each site \( S_i \) independently performs a parallel and incremental search for \( MFI_{\delta_i} \), using ZigZag on its data set \( \delta_i \). After all sites finish searching, the result will be the set of all local MFIs, \( \{MFI_{\delta_1}, MFI_{\delta_2}, \ldots, MFI_{\delta_i}\} \). This information is sufficient for determining all local frequent itemsets, and from Lemma 1, it is also sufficient for
determining all global frequent itemsets. The second step starts after all local MFIs have been found. Each site sends its local MFI to the other sites, and then they join all local MFIs. Now each site knows the set $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$, which is an upper bound for MFI$_{\Delta}$.

In the third step each site independently performs a top down incremental enumeration of the potentially global frequent itemsets, as follows. Each itemset present in the upper bound $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$ is broken into $k$ subsets of size $(k - 1)$. This process iterates, generating smaller subsets and incrementally computing their support counts until there are no more subsets to be checked. At the end of this step, each site will have the same set of potentially global frequent itemsets (and the support associated with each of these itemsets).

**Lemma 2** $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$ determines all global frequent itemsets.

**Proof:** We know from Lemma 1 that if $C$ is a global frequent itemset, so it must be local frequent in at least one partition. If $C$ is local frequent in some partition $\delta_1$, so it must be determined by $\text{MFI}_{\delta_1}$, and consequently by $\bigcup_{i=1}^{n} \text{MFI}_{\delta_i}$. □

By Lemma 1 all potential global frequent itemsets were found, but not all itemsets generated in the third step are global frequent (some of them are just local frequent). The fourth and final step makes a reduction operation on the local support counts of each itemset, to verify which of them are globally frequent in $\Delta$. The process starts with site $S_1$, which sends the support counts of its itemsets (generated in the third step) to site $S_2$. Site $S_2$ adds the support count of each itemset (generated in the third step) to the value of the same itemset obtained from site $S_1$, and sends the result to site $S_3$. This procedure continues until site $S_n$ has the global support counts of all potentially global frequent itemsets. Then site $S_n$ finds all itemsets that have support greater than or equal to $s_\Delta$, which constitutes the set of all global frequent itemsets, i.e., MFI$_{\Delta}$.  

63
We illustrate all steps of the algorithm execution in Figure 4.3. The transactions of \( \Delta \) (used in the example of Figure 4.1) were distributed in two data sets \( \delta_1 \) and \( \delta_2 \). The value of the minimum support is 50%. In the first step each site mines its local MFI. The result is \( \text{MFI}_{\delta_1} = \{ \text{ABDE, BCE} \} \), and \( \text{MFI}_{\delta_2} = \{ \text{ACDE, BCD} \} \). In the next step, all sites exchange their local MFIs, so that each one can compute the upper bound \( \bigcup_{i=1}^{n} \text{MFI}_{\delta_i} \), which is \( \{ \text{ABDE, ACDE, BCD, BCE} \} \). Now, each site computes the support count of each subset of each itemset in \( \bigcup_{i=1}^{n} \text{MFI}_{\delta_i} \). Some of the generated subsets at site \( S_i \) are not local frequents in \( d_i \), but their support count must be computed because they must be local frequent at the other site, and therefore they can still be global frequent itemsets (i.e., ABE). In the last step, the global frequent itemsets are found by aggregating (sum reduction operation) the local counts of each local frequent itemset.

The overall approach is as follows. At each site the local MFI is found using the appropriate version of the ZIGZAG algorithm (the sequential version if the site is a single processor machine, or the parallel version if the site is a SMP machine or a cluster). Once the local MFIs have been found, the distributed procedure is then used across the different sites to find the global MFI. An upper bound on the amount of communication performed during the distributed mining operation can be found in [114].

### 4.3.4 Interactivity Issues

#### Query Response Time

One of the goals of the distributed mining algorithm is to minimize response time to a query for the global frequent itemsets in a dynamic, distributed database. Since the database is dynamic, each site is incrementally updating its local frequent itemsets. The time it takes to update the local frequent itemsets is proportional to \( B = | d^+ | + | d^- | \), that is to say, the size of a block of differences. We can view the updates to the database as a queue.
containing zero or more such blocks. If a query arrives while a block is being processed, there cannot be a response until the calculation of the local frequent itemsets is completed and used to find the global frequent itemsets. An obvious approach to reducing response time is to decrease the size of $B$. However, because of overhead, the time it takes to do two increments of size $B$ is longer than the time it takes to do a single increment of size $2 \times B$. So there is a trade-off: If the size of $B$ is increased, then the global frequent model $MFI_{\Delta}$ will be more up-to-date, since it incorporates a greater number of changes to the database, but the amount of time it takes to respond to the query will increase.

High-Contrast Frequent Itemsets

An effective way of modeling a distributed database is to characterize the differences between the different sites. One way to characterize these differences is to use high-contrast frequent itemsets. As mentioned above, these itemsets can help identify sites that
are anomalous (as they contain data that is different from the data contained at the other sites, and hence many itemsets that do not appear elsewhere), or rules that are themselves anomalous (for example, itemsets that are only frequent in a small subset of the sites).

The support counts of high-contrast frequent itemsets vary significantly across the different sites. We use the well-established notion of entropy to detect how the support count of a given frequent itemset is distributed across the databases [32]. For a random variable, the entropy is a measure of the non-uniformity of its probability distribution. Let $X$ be a global frequent itemset. The value $p_X(i) = \frac{X.sup_i}{X.sup}$ is the probability that $X$ occurs in $\delta_i$. $\sum_{i=1}^{n} p_X(i) = 1$, and $H(X) = -\sum_{i=1}^{n} (p_X(i) \times \log(p_X(i)))$ is a measure of how the local support counts of $X$ is distributed across the different databases. Now let $E(X) = \frac{\log(n) - H(X)}{\log(n)}$. Then high values of $E(X)$ infers more skew is present in the database. Note that since $0 \leq H(X) \leq \log(n)$, then $0 \leq E(X) \leq 1$. If $E(X)$ is greater than or equal to a given minimum entropy threshold, then $X$ is classified as high-contrast frequent itemset. The high-contrast frequent itemsets that one finds is depended upon both the minimum entropy threshold and the minimum support used to mine the frequent itemsets, since this determines exactly what itemsets are frequent.

4.3.5 Discussions

Mining the MFI to find the frequent subsets has several advantages, both in incremental and distributed mining. In this section we discuss some of these advantages.

All extant incremental mining algorithms make use of the negative border [90] to perform the incremental operation. The basic idea is to maintain the negative border as the data set is updated. As shown in [151], the size of the negative border is typically much
larger than the size of $MFI_\Delta$. So, updating the negative border requires many more candidates to be processed, incurring computational and I/O overhead. By updating $MFI_\Delta$, we process fewer candidates than other approaches.

Almost all distributed algorithms for frequent itemset mining (COUNT DISTRIBUTION [4], FDM [28], and DMA [31]) require a round of communication in every iteration of the algorithm. However, synchronization is implicit in communication, and therefore these algorithms suffer from excessive communication overhead. Our approach overcomes this problem by making use of maximal frequent itemsets. Each site independently searches for its local MFI, so no communication is needed during this search. After all local MFIs are found, only one round of communication is performed to build the upper bound. Again, each site independently enumerates the local frequent itemsets, and after all local frequent itemsets are found, only one reduction operation is needed to find the global frequent itemsets. Furthermore, the global frequent itemsets need only be found when the system receives a query. Until then, each site works independently to update its local MFI continuously. Therefore, by making use of maximal frequent itemsets, our distributed algorithm asynchronously mines the frequent itemsets.

### 4.4 Experimental Evaluation

Our experimental evaluation of the algorithms presented in section 4.3.3 was carried out on two clusters. The first cluster consists of nodes with dual PENTIUM III 1GHz processors and 1GB of memory running Red Hat Linux 7.1. The second cluster consists of nodes with single PENTIUM III 933 MHz processors and 512 MB of memory running Red Hat Linux 7.3. We further partitioned both clusters into two virtual clusters for a total of four clusters for some experiments. We assume that each database is distributed between
the clusters, and that each node in the cluster has complete access to its cluster’s portion of the database. Within each cluster, we have implemented the parallel program using the MPI message-passing library (MPICH over GM²), and for communication between clusters we use sockets. On dual-processor nodes, we use POSIX PTHREADS to carry out parallel computation.

We used several real and synthetic data sets for testing the performance of our algorithms. The WCup data set is derived from click-stream data from the official site of the 1998 World Soccer Cup. The WPortal data set is derived from the click-stream data of a large Brazilian web portal. We scanned the logs and produced transaction files, where each transaction is a session of accesses to the site by a client. Each item in the transaction is a web request. Not all web requests were turned into items; to become an item, the request must have three properties: (1) the request method is GET; (2) the request status is OK; and (3) the file type is HTML. A session starts with a request that satisfies the above properties, and ends when there has been no click from the client for 30 minutes. We also used a synthetic data set (also available from IBM Almaden), which has been used for benchmarking previous data mining algorithms. This data set mimics the transactions in a retailing environment [7]. Table 4.1 shows the characteristics of the real and synthetic data sets used in our evaluation. It shows the number of items, the average transaction length, the number of transactions, and the size of each data set.

### 4.4.1 Distributed Incremental Algorithm Evaluation

We performed several sets of experiments in a broader scenario involving several clusters. The first set involved finding the number of transactions that were processed and incorporated into the global model MFIΔ when we varied certain parameters. The second

²www.myricom.com
<table>
<thead>
<tr>
<th>Data set</th>
<th>#Items</th>
<th>Avg. Length</th>
<th>#Transactions</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCup</td>
<td>5,271</td>
<td>8</td>
<td>7,618,927</td>
<td>645MB</td>
</tr>
<tr>
<td>WPortal</td>
<td>3,183</td>
<td>4</td>
<td>7,786,137</td>
<td>428MB</td>
</tr>
<tr>
<td>T512D8000K</td>
<td>2,000</td>
<td>5</td>
<td>8,000,000</td>
<td>351MB</td>
</tr>
</tbody>
</table>

Table 4.1: Data set characteristics.

The first experiment we conducted was to examine how the query response time was affected by the block size and the query arrival time.

**Transactions Processed**

The first experiment we conducted was to examine how the size of a block, the number of nodes used in each cluster, and the time at which a query arrives affect the amount of data used to build the global model $MFI_\Delta$. For the *WPortal* database we used a minimum support of 0.01% and for the *WCup* database we used a minimum support of 1.0%. The
results all have similar trends, and two example cases can be seen in Figure 4.4. The X-axis represents the time elapsed from the beginning of the mining operation until the moment when the query arrived (the deadline), and the Y-axis represents the number of transactions that are incorporated into the global model $\text{MFI}_\Delta$. The lines on the graph represent different values of the block size $B$, which is given here as percentages of the database on each cluster. The graphs show that if more time that elapses before the query arrives, the more data that can be incorporated into the model, which is to be expected. It also shows that as the block size decreases, fewer transactions can be processed before the next query arrives. This is due to the fact there is more overhead involved in processing a large number of small blocks than there is in processing a small number of large blocks.

**Query Response Time**

In the next set of experiments we focused on the query response time, that is to say, the amount of time a user must wait before the global model is computed. For this experiment we varied the block size $B$ (in these experiments, we assume that each cluster uses the same block size) and the time at which the query arrives. The results can be seen in Figure 4.5. The X-axis again represents the time at which the query arrives, and the Y-axis represents the time spent waiting for the global model to be computed. These graphs show that as $B$ decreases, the time to wait for a response also decreases. However, the time at which a query arrives affects the waiting time in a seemingly random manner. This is because a query arrives at some random point during the processing of a block. The time remaining to compute the local frequent itemsets is therefore a random number. The graphs above show the query response time averaged over five runs, and the vertical bars represent the variance in the runs. This set of experiments, in conjunction with the previous set, clearly show the trade-off between block size and query response time: As the block size increases,
the number of transactions processed also increases, but the response time increases as well. Figure 4.6 shows the same experiment when different block sizes across clusters are allowed. The basic idea is that smaller response times can be obtained by assigning larger blocks to the less powerful cluster. The local model in this cluster will be updated less frequently, but in response, smaller query response times can be obtained by this approach. We can prove this by comparing the results in Figure 4.6 against the respective result in Figure 4.5.

Communication

We also performed a set of experiments to analyze the communication overhead imposed by our algorithm. In particular we examined the number of bytes transferred between clusters when we varied the minimum support, the block size $B$, and the number of clusters involved in the computation. The results can be seen in Figure 4.7. As is expected, as the minimum support decreases, the number of candidates will increase, and therefore the number of bytes that must be transferred between the clusters will increase, since our algorithm must exchange the support counts of every candidate processed. Also, as the block size increases, the amount of communication decreases. The reason is that for smaller block sizes the number of candidates processed tends to be greater (assuming the same minimum support). Finally, the amount of communication required increases when more clusters are involved in the process. However, the increasing factor is not linear because the skewness of the data also increases when more clusters are involved, and so the number of candidates processed is increased as well.
High-Contrast Itemsets

The last set of experiments examine high-contrast frequent itemsets. We utilized three databases: \textit{WPortal, WCup}, and a highly-skewed synthetic database. The highly-skewed synthetic database was created by generating four different synthetic databases: \textit{T10I2D200K}, \textit{T10I4D2000K}, \textit{T10I6D2000K} and \textit{T10I8D2000K}, and then assigning them each to a different cluster.

We varied three parameters: the minimum support, the number of clusters involved in the process, and the minimum entropy. Figure 4.8 shows the results obtained in each database. As we can observe, very different results were obtained from each database. The percentage of high-contrast frequent itemsets is interesting because to some extent it reveals the skewness of the database. From the experimental results, we know that the \textit{WCup} database is more skewed than the \textit{WPortal} database. Given the same support thresholds for these two databases, \textit{WCup} will give a much higher percentage of high-contrast frequent itemsets. The percentage of high-contrast frequent itemsets usually decreases as the minimum support threshold increases. This is expected, given that when the support threshold is low, there will be a large number of global frequent itemsets generated. Many of these itemsets become global frequent only because they have high local support at some site.

In contrast, when the support threshold increases, it becomes harder for a local frequent itemset to become global frequent, which results in a decrease of high-contrast frequent itemsets. Accordingly, there is a higher proportion of high-contrast frequent itemsets in the former scenario. Furthermore, the more clusters over which the data distributed, the greater the possibility of skewness in the data. This is verified by the experimental data. It is interesting to notice that for the highly-skewed synthetic data, when the support threshold is incremented from 0.05 to 0.1, the percentage of high-contrast frequent itemsets did not
increase as expected. We guess this can be attributed to the high skewness of the data. We surmise that for such data there exists some threshold for our claim to take effect. Taking our synthetic data as an example, the threshold value is around 0.1. Below this threshold, when we raised the support value, both the number of high-contrast frequent itemsets and global frequent itemsets decreased, but the loss of the high-contrast frequent itemsets was dominated by the loss of the global frequent itemsets, leading to an increased percentage of high-contrast frequent itemsets.

4.5 Summary

In this chapter we have considered the problem of mining frequent itemsets on dynamic and distributed databases, the solution of which has many applications for anomaly detection. We presented an efficient distributed incremental algorithm to deal with this problem. In particular, we presented techniques to minimize the response time to a query for the global set of frequent itemsets, as well as to find high-contrast frequent itemsets. The efficiency of our algorithm stems from the fact that it makes use of the MFI, reducing both the number of candidates processed and the amount of communication necessary. Our experiments in the distributed setting examined the trade-offs involved in minimizing the query response time (whether to sacrifice query response time in order to incorporate more transactions in the model), the amount of data transferred between clusters, and how the distribution of the data affected the number of high-contrast frequent itemsets. This work can facilitate anomaly detection in several different ways. ZIGZAG finds both local and global frequent itemsets, which allow us to build local and global models of normality. This allows for the identification of anomalies at both the local level that may not be visible globally, and vice versa. The high-contrast frequent itemsets allow for the identification of
anomalous sites, that is to say sites whose local model of normality is significantly different from that of other local sites or the global model. High-contrast frequent itemsets also allow for the identification of anomalous frequent itemsets, that is to say, itemsets that are only frequent in a small subset of all the sites.
Figure 4.5: Query Response Time using Equal Block Sizes.
Figure 4.6: Query Response Time using Different Block Sizes.

Figure 4.7: Communication Overhead.

Figure 4.8: High-Contrast Frequent Itemsets.
CHAPTER 5

DISTRIBUTED ANOMALY DETECTION IN MIXED ATTRIBUTE DATA

In many cases, outlier detection schemes need to be sensitive to response time constraints imposed by the domain. For instance, an application such as network intrusion detection requires on-the-fly outlier detection. Most existing outlier detection approaches require multiple passes over the data set which, if not impossible, are prohibitive in such a scenario. Also, all non-parametric approaches rely on some well-defined notion of distance to measure the proximity between two data points. However, many real-world data sets contain not only continuous attributes, but categorical attributes as well, which have a serious implication for a well-defined notion of distance. Simply disregarding categorical attributes in such a scenario may result in the loss of information important for effective outlier detection. Finally, the data in question may be distributed across different sites. Researchers in the network intrusion detection community have looked at distributed anomaly detection [89]. However, their algorithms are not general, as they are tailored to capturing specific network intrusions. In this work we address the above challenges by making the following contributions:

- We define an anomaly score wherein one can effectively identify outliers in a mixed attribute space by considering the dependencies between attributes of different types.
• We present a two-pass distributed algorithm for outlier detection based on this anomaly score. We also present an approximation scheme by which this algorithm allows for more efficient outlier detection when memory and processor resources are an issue.

• We extend our technique to handle dynamic data sets, where only a single pass can be made, and concept drift is a concern.

• We extensively evaluate our algorithms on several real and synthetic data sets and confirm the utility of finding outliers in mixed attribute data. We also present results on the scalability of our approach.

5.1 Algorithms

Our work is motivated by the following questions: First, how does one find outliers in mixed attribute data sets? Existing outlier detection algorithms do not handle mixed attribute data sets effectively. In our solution, we develop an anomaly score function that allows us to capture a) dependencies between categorical attributes, b) dependencies between continuous attributes, and c) dependencies between categorical and continuous attributes. As we will see, these dependencies capture information that is crucial for effective outlier detection.

Second, how does one find outliers in a distributed setting? Existing distance-based outlier detection techniques flag a data point as an outlier based on its distance from the other points in the data set. In a distributed setting, the sites must either replicate the entire data set locally, or engage in voluminous queries to calculate the distances between data points residing on different sites. However, this is not feasible due to the excessive storage and communication requirements. Ideally, each site should find outliers locally while minimizing communication with other sites. This is the basis of our approach and
is facilitated by an anomaly score function that is based on a global model of the data that is easily constructed by combining local models built independently at each site. The only communication that need take place between the sites is a single round of communication to build the global model from the local models.

Third, how does one find outliers in dynamic data sets? Most existing approaches for outlier detection must make multiple passes over the data set since they need to capture the distances between every pair of points in the data set. To facilitate online discovery, one needs to be able to summarize essential parts of data set incrementally. We show that we can also process the data set in one pass, using an anomaly score function that facilitates single-pass processing in a distributed setting.

Our distributed approach to outlier detection is a direct result of our method for computing anomaly scores for each point in a data set. It builds on our previous work with the LOADED algorithm [54] by adding support for distributed processing and altering the anomaly score function to better handle continuous attributes. We first present our anomaly score that takes into account the dependencies between the categorical and continuous attributes. We then present our distributed approach, as well as techniques for reducing memory and computing requirements. Finally, we show how to extend our approach to handle dynamic data sets.

5.1.1 Computing Anomaly Scores

Our anomaly score function is computed from a data model containing two basic parts that take into account the categorical and continuous attribute spaces respectively. Our approach is based on the idea of links, which take into account the dependencies between the attributes in the data set. Outliers are then the points that violate these dependencies.
We first consider links in categorical attribute space, and then consider them in a mixed attribute space.

**Links in a Categorical Attribute Space**

Informally, two data points $P_i$ and $P_j$ are linked if they are considerably similar to each other. Moreover, each link has a strength which captures the degree of similarity, and is determined using a similarity metric defined on the two points. Formally, we consider a data set $D$ in which each data point (record) $P_i$ has $N_c$ categorical attributes. We represent $P_i$ as an ordered set

$$P_i = < (attribute_1, value_1), \ldots, (attribute_{N_c}, value_{N_c}) >$$

consisting of $N_c$ attribute-value pairs. Specifically, we define two data points $P_i$ and $P_j$ to be linked in a categorical attribute space if they have at least one attribute-value pair in common. The associated link strength is equal to the number of attribute-value pairs shared between the two points.

Using these definitions, we define an outlier in a categorical attribute space as a data point that has either (a) very few links to other points or (b) very weak links to other points. A score function that generates high scores for outliers would assign scores to a point that are inversely proportional to the sum of the strengths of all its links. To estimate this score efficiently, we rely on ideas from frequent itemset mining [7]. Let $I$ be the set of all possible attribute-value pairs in the data set $M$. Let

$$D = \{ d \mid d \in PowerSet(I) \land \forall i,j:i \neq j d_i.attribute \neq d_j.attribute \}$$

be the set of all itemsets (or groups of attribute-value pairs), where an attribute only occurs once per itemset. We define $sup(d)$ as the number of points $P_i$ in the data set where $d \subseteq P_i$, 

80
otherwise known as the *support* of itemset \(d\). We also define \(|d|\) as the number of attribute-value pairs in \(d\).

We can then define an anomaly score function as:

\[
Score_1 (P) = \sum_{d \subseteq P} \left( \frac{1}{|d|} \mid sup(d) \leq s \right)
\]  

(5.1)

where the user-defined threshold \(s\) is the minimum support or, using our link terminology, the minimum number of links. The score is proportional to the number of infrequent itemsets, and inversely proportional to the size \(|d|\) of those infrequent itemsets (i.e. the link strength). The score function has the following characteristics:

- A point with no links to other points will have the highest possible score.
- A point that has only a few links, each with a low link strength, will have a high score.
- A point that has only a few links, some with a high link strength, will have a moderately high score.
- A point that has several links, but each with a low link strength, will have a moderately high score.
- Every other point will have a low to moderate score.

We note that the function \(sup(d)\) has the downward closure property, otherwise known as the *apriori* property:

**Lemma 3** If \(sup(d) > s\) then \(sup(d') \geq s \forall d' \subseteq d\).
Example

The score estimation process is illustrated as follows: Consider the two points $P_1$ and $P_2$ as shown in Figure 5.1 together with their itemset lattice. Note that Figure 5.1 only shows the subset of the overall itemset lattice corresponding to the points $P_1$ and $P_2$. For point $P_1$, all of its subset itemsets are frequent, and thus using the function $Score_1$, its score would be zero (so it is not an outlier). On the other hand, for point $P_2$, the subset itemsets $abd$ and $bd$ are not frequent. Since $abd$ has size 3 and $bd$ has size 2, the resulting score is $\frac{1}{3} + \frac{1}{2} = \frac{5}{6}$.

Links in a Mixed Attribute Space

Thus far, we have presented an approach to estimate a score for a point in a categorical attribute space. As motivated previously, we would like to capture the dependencies between the continuous and categorical attributes in a domain-independent manner. If a point violates the expected dependencies between the categorical and continuous attributes, then it is most likely an outlier. We choose a unified approach to capture this dependence. We incrementally maintain covariance matrices [130] to capture the dependencies between the

---

Figure 5.1: Lattice management
continuous attributes. For each itemset in the data set $D$, we maintain a covariance matrix, implicitly capturing the dependence between the values in the mixed categorical and continuous attribute space.

A point is defined to be linked to another point in the mixed attribute space if a) they are linked in the categorical data space, and b) if their continuous attributes adhere to the joint distribution as indicated by the covariance matrix. To determine whether a point’s continuous attributes adhere to the distribution, we check to what degree the point violates the covariance matrix. Consider the vector $P$ containing the continuous attributes of a data point which also contains the itemset $d$ (in a slight abuse of notation, we express this as $d \subset P$). As stated above, for each itemset $d$ in the lattice, there is a covariance matrix $C$, so let $C^d$ be the covariance matrix for itemset $d$, and $C^d_{ij}$ is then the covariance between continuous attributes $i$ and $j$. To determine the degree to which $P$ violates $C^d_{ij}$, we calculate the covariance score $c_{ij}^P$ for each pair of continuous attributes $i$ and $j$ of $P$:

$$c_{ij}^P = (P_i - \mu_i^d) \times (P_j - \mu_j^d)$$

where $\mu_i^d$ and $\mu_j^d$ are the means of the attributes $i$ and $j$ for all points in the data set containing the itemset $d$. We can then compute the violation score $V$ of a point $P$ as:

$$V_\tau(P) = \sum_i \sum_j v_\tau(P_{ij})$$

where

$$v_\tau(P_{ij}) = \begin{cases} 0 & \text{if } \frac{c_{ij}^P - C^d_{ij}}{\sigma^2_{C^d_{ij}}} \leq \tau \\ 1 & \text{otherwise.} \end{cases}$$

and

$$\sigma^2_{C^d_{ij}} = \frac{1}{\text{sup}(d) - 1} \sum_{\{P \mid d \subseteq P\}} (c_{ij}^P - C^d_{ij})^2.$$
Under the assumption that $\frac{c_{ij}^P - c_{ij}^d}{\sigma_{c_{ij}}}$ is normally distributed, the value of $\tau$ represents the width of a confidence interval for $c_{ij}^P$ (or a measure of the degree of a violation we are willing to tolerate), and can be chosen from a table of the cumulative normal distribution.

We can now modify our anomaly score function for a point $P$ to account for mixed attribute data as follows:

$$Score_2(P) = \sum_{d \subseteq P} \left( \frac{1}{|d|} \mid (C1 \lor C2) \text{ is true} \right)$$

(5.6)

where $C1: \text{sup}(d) \leq s$, and $C2: V(P) > \delta$. $\delta$ is then the maximum violation score allowed for each covariance matrix. Condition $C1$ is the same condition used to find outliers in a categorical data space using $Score_1$, and condition $C2$ adds continuous attribute checks to $Score_1$. This score function therefore captures:

- Dependencies between categorical attributes (C1).
- Dependencies between continuous attributes (C2).
- Dependencies between categorical and continuous attributes, since we maintain a covariance matrix for each itemset in the lattice.

We note that when using the $Score_1$ function in Equation 5.1, one need not examine all possible subsets of the categorical attributes. This is because some of the subsets may be frequent, and due to the apriori property in Lemma 3, the subsets of frequent itemsets are also frequent, and therefore add nothing to the score. Therefore, one can prune the search space by ignoring the subsets of frequent itemsets. However, for the score function in Equation 5.6, one may potentially have to consider all subsets of the categorical attributes, even if some subsets are frequent. This is due to the covariances; even if an itemset is
frequent and the continuous attributes agree with the covariances for that itemset, they may disagree with the covariance matrices corresponding to the subsets of that itemset.

We state a heuristic, however, that if the continuous attributes of a point $P$ do not violate the covariance matrix $C_d$ for some itemset $d \subseteq P$, then $\forall e \subset d$, the continuous attributes of $P$ will not violate $C_e$, since $C_d$ is computed from a subset of the points used to compute $C_e$, by the apriori property in Lemma 3. Therefore there is a subset or cluster of points associated with itemset $e$ that have a covariance with which the continuous attributes of $P$ agree, even if $V_\tau(P) > \delta$ for itemset $e$. This heuristic allows us to modify our score function as follows:

$$Score_3(P) = \sum_{d \subseteq P} \left( \frac{1}{|d|} : (C1 \lor C2) \land C3 \text{ is true} \right)$$

(5.7)

The conditions $C1$ and $C2$ are the same as in $Score_2$ (see Equation 5.6). The new condition, $C3$ is defined as follows: $C1 \lor C2$ holds true for every superset of $d$ in $P$. Condition $C3$ is our heuristic that allows for more efficient processing since if an itemset does not satisfy conditions $C1$ and $C2$, then none of its subsets will be considered. In other words, for point $P1 = (A, B, C)$ in Figure 5.1, if the itemset $ABC$ is frequent and if covariances for itemset $ABC$ agree with the continuous attribute values in point $P1$ (within a certain threshold $\delta$), none of its subset itemsets $AB, BC, AC, A, B, C$ need be checked.

**Example**

We again go through our example, this time using $Score_3$ in Equation 5.7. Note that in Figure 5.1, frequent itemsets are labeled using uppercase characters, while infrequent itemsets are labeled using lowercase characters. Consider the points $P1 = (A, B, C)$ and $P2 = (A, B, D)$. For simplicity, we will assume that the continuous attributes of $P1$ and $P2$ are always in agreement with the covariance matrices, and so are not shown here.
The three-itemset corresponding to $P_1$ is frequent, so we only need to calculate the score for itemset ABC. As for $P_2$, ABD is not frequent. Therefore, we increment the score and examine its subsets AB, AD, and BD. AB and AD are frequent, so we only have to increment the score for both itemsets and disregard their subsets. However, itemset BD is not frequent. As before, we increment the score and examine its subsets, B and D. B and D are both frequent, so we just increment the score for both and move on to the next point. Note that in our algorithm, we do not have to examine all of the subsets of itemset ABD.

### 5.1.2 Distributed Model Construction

As mentioned earlier, our basic approach is for each site to build a local model independently, and then combine these models to build a global model, which each site then uses to detect outliers in its subset of the complete data set. Furthermore, the local models can be constructed in a single pass of each site’s subset of the data, and outliers can be detected in a second pass using the global model.

#### Local Model Construction

We must estimate all necessary model parameters in a single pass of the local data set. This is trivial for the support counts of each itemset in the lattice (i.e. $sup(d)$). For the covariance matrices, we note that after applying some basic algebra, we have:

$$C_{ij}^d = \frac{S_{ij}^d}{sup(d) - 1} + \frac{L_i^d \times L_j^d}{sup(d) \times (sup(d) - 1)}$$

(5.8)

where

$$S_{ij}^d = \sum_{\{P | d \subseteq P\}} P_i \times P_j$$

(5.9)

and

$$L_i^d = \sum_{\{P | d \subseteq P\}} P_i$$

(5.10)
Similarly, for computing the violation score (see Equation 5.5), we note that

\[
\sigma^2_{C_{ij}} = \frac{S_{c_{ij}} + 2C_{ij}L_{c_{ij}} + sup(d) \times (C_{ij})^2}{sup(d) - 1}
\]  

(5.11)

where

\[
S_{c_{ij}} = \sum_{\{P|d \subset P\}} (c_{P,d}^{ij})^2 \]  

(5.12)

and

\[
L_{c_{ij}} = \sum_{\{P|d \subset P\}} c_{P,d}^{ij} \]  

(5.13)

Note that the means \(\mu_i\) and \(\mu_j\) used in Equation 5.2 can be computed from \(L_{d,i}^d\), \(L_{j}^d\), and \(sup(d)\). We also note that all of these values are exact, except for that of \(\sigma^2_{C_{ij}}\), which must be calculated using only the partial estimates of \(\mu_i\) and \(\mu_j\) available at that point in the pass (see Equation 5.2). However, as more points are processed, the values of \(\mu_i\) and \(\mu_j\) will stabilize. At the end of the first pass, each site will have an itemset lattice, and for each itemset \(d\) in the lattice, there will be a support count \(sup(d)\), matrices \(S^d\), \(S_{c_{ij}}\), and vectors \(L^d\) and \(L_{c_{ij}}\). The psuedocode for the local model building algorithm is shown in Figure 5.2.

**Procedure:** BuildLocalModel  
**Input:** DataSet, MAXLEVEL  
**Output:** Local Model  
For each point \(p \in DataSet\):
  - \(G = \) Enumeration of all itemsets of size \(MAXLEVEL\) for point \(p\)
  - For each \(g \in G\):
    - Update \(sup(g)\), \(S^g\), \(L^g\), \(S_{c_{gs}}\), and \(L_{c_{gs}}\) for \(g\) in the hash table
    - Add all subsets of \(g\) of size \(|g| - 1\) into \(G\)
  - End For
End For

![Figure 5.2: Constructing the local model.](image-url)
Global Model Construction

After the first pass of the data, the sites participate in a reduction operation to construct the global model. The reduction operation to produce the global model is a simple summation of the local models, since it follows from Equations 5.8 to 5.13 that for each itemset \( d \) in the lattice:

\[
\text{Global}(\sup(d)) = \sum_{i=1}^{N} \text{Local}_i(\sup(d)) \tag{5.14}
\]

\[
\text{Global}(S^d) = \sum_{i=1}^{N} \text{Local}_i(S^d) \tag{5.15}
\]

\[
\text{Global}(L^d) = \sum_{i=1}^{N} \text{Local}_i(L^d) \tag{5.16}
\]

\[
\text{Global}(S_{cd}) = \sum_{i=1}^{N} \text{Local}_i(S_{cd}) \tag{5.17}
\]

\[
\text{Global}(L_{cd}) = \sum_{i=1}^{N} \text{Local}_i(L_{cd}) \tag{5.18}
\]

where \( N \) is the number of sites participating. From these values we can compute the global values of all model parameters. The result is that the global model is the same as a local model constructed by a site that has access to the complete data set.

5.1.3 Distributed Outlier Detection

In this section we detail the process of detecting outliers in the second pass of the algorithm. Once the reduction has been performed, each site will have a copy of the global model. In the second pass, each site, now armed with the global model, will calculate an anomaly score (see \( \text{Score}_3 \) in Equation 5.7) for each point in the local data set. Any point exceeding a threshold will be flagged as an outlier. The psuedocode for the outlier detection algorithm is shown in Figure 5.3. An anomaly score for a point is calculated.
Procedure: FindOutliers

Input: DataSet, s, τ, δ, ∆Score, ScoreWindowSize, MAXLEVEL

Output: Outliers discovered in one pass

For each point \( p \in \text{DataSet} \):
   \( G = \) All itemsets of size MAXLEVEL in point \( p \)
   For each \( g \in G \):
      Get \( \text{sup}(g) \) from the hash table
      If \( \text{sup}(g) < s \):
         \( \text{score}(p) \leftarrow \text{score}(p) + \frac{1}{|g|} \)
      Else
         If \( \text{V}_\tau(p) > \delta \):
            \( \text{score}(p) \leftarrow \text{score}(p) + \frac{1}{|g|} \)
            Add all subsets of \( g \) of size \(|g| - 1\) into \( G \)
      End For
   If \( \text{score}(p) > \) (average score in ScoreWindow \( \times \) ∆Score):
      Flag as Outlier
   Else
      Normal
      Update the Score window of size ScoreWindowSize
   End For

end

Figure 5.3: Finding outliers in the second pass of the data set.

Based on the number of itemsets that it contains (as per function \( \text{Score}_3 \) in Equation 5.7) that are infrequent or whose corresponding covariances are violated. Our algorithm works as follows:

1. For each point, we enumerate all possible itemsets with a size of \( \text{MAXLEVEL} = N_c \) into a set \( G \).
2. For each of the itemsets \( g \in G \), we increment its support count in the lattice.
3. We check if \( g \)'s support is less than \( s \). If so, we increase the score by a value inversely proportional to the size of the itemset, as dictated by the score function.
4. If the itemset $g$ is frequent, we check that the continuous attributes of the point are in agreement with the distribution determined by the covariance matrix using the covariance violation threshold $\delta$. If so, we will not increase the score. Otherwise, we increase the score, again by a value inversely proportional to the size of the itemset.

5. If $g$ is not frequent, or if the continuous attributes of the point violate the covariance matrix, find all subsets of $g$ of size $|g| - 1$ and insert these into $G$.

6. If $G$ is not empty, go to step 2, otherwise continue.

7. We maintain all reported scores over a recent window of size $\text{ScoreWindowSize}$ and calculate the average score in the window. If the total score for the point is greater than $\Delta\text{Score} \times \text{average score}$, then we flag this point as a local outlier.

### 5.1.4 Complexity Analysis and Modifications for Increased Efficiency

We first examine the space and computational complexity of our outlier detection algorithm as presented above. For a feature space with $N_c$ categorical attributes and $N_q$ continuous attributes, let $M$ be the maximum number of distinct values taken by any categorical attribute. For a data set of size $n$, we have

$$\text{Total space} \leq N_q^2 \times \sum_{i=0}^{N_c} M^i \binom{N_c}{i} = O \left( N_q^2 \times (M + 1)^{N_c} \right)$$

Thus, the space used scales quadratically with $N_q$ and exponentially with $M$ and $N_c$, but it is independent of the number of points in the data set. When we look at the computational complexity, for a data set of size $n$, we have

$$\text{Total execution time} \leq n \times N_q^2 \times \sum_{i=0}^{N_c} \binom{N_c}{i} = O \left( n \times N_q^2 \times 2^{N_c} \right) \, .$$
Thus, execution times scales linearly with \( n \), quadratically with \( N_q \) and exponentially with \( N_c \). However, there are several modifications we can make so that our algorithm is more efficient in terms of both memory usage and execution time, especially in regards to the exponential complexity.

**Using a Partial Lattice**

Notice that in our data structure we keep a matrix \( S^d \) and a vector \( L^d \) for each itemset \( d \in D \). This is clearly unrealistic in terms of memory requirements for data sets with high dimensionality. One modification is to reduce the number of itemsets we have to examine. Instead of storing all itemsets (with their respective matrices and vectors), we only store itemsets and corresponding matrices and vectors if those itemsets have a size less than or equal to a value \( MAXLEVEL \leq N_c \). This also aids in reducing the amount of computation performed. For some applications, the amount of memory used by our algorithm or the time taken to determine if a point is an outlier may be too costly. In these cases, it would be better to use a smaller value for \( MAXLEVEL \). Note that if we use a value of \( MAXLEVEL < N_c \), the accuracy of the results will be affected. However, given an \( k \)-itemset \( I \), the frequency of \( I \) is bounded by the smallest frequency of its subsets of size \((k - 1)\). As a result, the frequencies of the lower order itemsets serve as good approximations for the frequencies of the higher order itemsets. In the general case, however, accuracy improves as \( MAXLEVEL \) increases. We empirically validate this in Section 5.2.

The time complexity of our algorithm grows linearly in the size of the data set and quadratically in the number of continuous attributes. Furthermore, our algorithm maintains and examines the itemset lattice beginning at \( MAXLEVEL \leq N \) down to the positive border. Therefore, the time it takes to check if a point is an outlier is proportional to
the number of lattice levels between $MAXLEVEL$ and the positive border. Moreover, a decade of research [7, 151] has shown that under practical considerations, an itemset lattice can be managed in an efficient and scalable fashion, which is also evident in our experimental results.

**Discarding the Lattice**

It is also possible to drastically save on memory usage (and communication bandwidth) by discarding the lattice structure entirely. This is due to the fact the entire lattice can be summarized by its topmost level. Recall that there are several values associated with each itemset $d$ in the lattice. These values are $sup(d)$, $S^d$, $L^d$, $S_{cd}$, and $L_{cd}$ (see Equations 5.8 to 5.13). Consider an itemset $d$ not in the top level of the lattice, having size $k$. We note that:

$$\text{sup}(d) = \sum_{\{e \supset d \land |e| = k+1\}} \text{sup}(e)$$

(5.19)

Similar equalities hold for $S^d$, $L^d$, $S_{cd}$, and $L_{cd}$, following from equations 5.8 to 5.13. Therefore, instead of storing the entire lattice, we can compute the relevant nodes on the fly. For an itemset $d$ with $|d| < N_c$, to compute the anomaly score, we need to access all itemsets in the $N_c$ that are a superset of $d$. This approach saves on memory usage, requiring only $O(N_q^2 M N_c)$ space, compared to the $O(N_q^2 (M + 1)^{N_c})$ space needed by the original algorithm. However, this savings in memory usage has the trade-off that in it increases the computational complexity. In the original version, one can maintain a large hash table containing all itemsets and relevant matrices, but with this modification, the relevant portions of the lattice must be recomputed for each data point. Each itemset $d \subset P$ must have the relevant statistics ($sup(d)$, $S^d$, etc.) computed from each of its potential $M^{N_c - |d|}$ supersets in the top level of the lattice, adding another exponential factor to the computational complexity. Nevertheless, the fact that the top row summarizes the
entire itemset lattice allows us to more efficiently use the communication bandwidth when constructing a global model, as entire lattices need not be exchanged.

5.1.5 Single Pass Approach for Dynamic Data Sets

Many applications, such as network intrusion detection, demand that outlier detection be performed on dynamic or streaming data. In such an environment, it is impossible to make more than a single pass over the data. In this section we present a variation on our outlier detection algorithm to discover outliers in distributed dynamic data sets.

Since the model parameters are computed incrementally, it is straightforward to use them immediately to detect outliers. Our basic single-pass approach exploits this fact and uses the values of the model parameters computed from all previous points to compute the anomaly score of the current point. The drawback of this approach is that it requires an anomaly score be computed for a point using incomplete knowledge of the full data set. This is particularly a problem for the first several points of the data set, when the model parameters have been estimated from only a few points. However, this approach has the advantages that it is the only appropriate approach available for dynamic or streaming data, and with some simple extensions, it can cope with the problem of concept drift. Our algorithm for single-pass distributed outlier detection is given in Figure 5.4. It is very similar to the second-pass outlier detection algorithm given in Figure 5.3, except that each new point to be examined is incorporated into the local model.

The major concern for the single-pass approach is how to build a global model. Because of the dynamic nature of the data, each site will always have only a partial local model. However, the sites can still construct a global model from their partial local models in much the same way as in Section 5.1.2. The difference is that the global model must be...
**Procedure:** OnePassFindOutliers  
**Input:** $DataSet, s, \tau, \delta, \Delta Score, ScoreWindowSize, MAXLEVEL$  
**Output:** Outliers discovered in one pass

For each point $p \in DataSet$:  
- $G =$ All itemsets of size $MAXLEVEL$ in point $p$  
- For each $g \in G$  
  - Get $sup(g)$ from the hash table  
  - Update $sup(g), L^g, S_{c^g},$ and $L_{c^g}$ for $g$ in the hash table  
  - If $sup(g) < s$  
    - $score(p) \leftarrow score(p) + \frac{1}{|g|}$  
  - Else  
    - If $V_{\tau}(p) > \delta$  
      - $score(p) \leftarrow score(p) + \frac{1}{|g|}$  
    - Add all subsets of $g$ of size $|g| - 1$ into $G$  
- End For  
- If $score(p) > (\text{average score in ScoreWindow } \times \Delta Score)$  
  - Flag as Outlier  
  - Else  
    - Normal  
    - Update the Score window of size $ScoreWindowSize$  
- End For  

Figure 5.4: Finding outliers in a single pass of the data set.

reconstructed regularly, since the local models are continually being updated. However regularly updating the models is an expensive operation considering the size of the lattice and the available network bandwidth. Even if we use both modifications in Section 5.1.4, the size of the data structure to be communicated is $O(N_q^2 M^{MAXLEVEL} \binom{N_c}{MAXLEVEL})$. A more efficient alternative would be to exchange local outliers between sites, since each data point is much smaller than the model, and the number of outlier points should be very small, since outliers are, by definition, rare. If all sites agree that a point is an outlier, then we can assume that the point is a global outlier.
Our distributed single pass algorithm is presented in Figure 5.5. Each site runs the *OnePassFindOutliers* algorithm and the parameters are set uniformly across all sites. The sites only communicate when some user-specified event occurs. Examples of such events include a user’s query for the global outliers, when a site finishes processing a fixed number of points, or when a site finds a fixed number of outliers. When such an event occurs, each site broadcasts all outliers it has found since the last event to all other sites. Upon receiving a broadcast, a site will check all the points it has received in the broadcast to see if they are flagged as outliers locally, and will return that information to the requesting site. If, for a given point, all sites agree that it is a local outlier, then it is flagged as a global outlier.

There are trade-offs to consider when choosing which type of event to use. For example, since outliers are rare by definition, the simplest event to use is that a site requests validation each time it receives a new local outlier. However, this approach requires a synchronization between sites for any outlier found at a local site. As an alternative, we can choose an event where a site accumulates $k$ potential local outliers before broadcasting them for validation. In this case, there is less synchronization between the sites, but there is a greater communication cost. This increase in communication cost stems from the fact that the sites are receiving information about global outliers less often, which increases the number of local outliers that need to be broadcast at each event.

** Modifications for Increased Accuracy **

As motivated previously, several applications (e.g. network intrusion detection) demand outlier detection over dynamic data. In order to capture concept drift and other dynamics of the data, we need to update our model to capture the most recent trends in the data. We achieve this by introducing a bias towards the more recent data. We maintain the itemsets together with their frequency counts in a hash table and the bias is introduced in
Distributed One-Pass Outlier Detection Algorithm

For each incoming point
   Process the point using OnePassFindOutliers()
   If it is found to be a local outlier,
      Add it to the set of outliers
         to be verified at next event
   If an event occurs,
      Broadcast the set of potential outliers
         to all other sites
      Receive normal/outlier labels from all the sites
      If all other sites also flag a point as an outlier,
         Flag the point to be a global outlier
   If another site needs outlier verification,
     Receive the points, and process them
     Return outlier/normal labels to that site
End For

Figure 5.5: Distributed one-pass algorithm for outlier detection

the following two ways: First, we maintain an itemset together with its frequency count in the hash table only if it appears at least once every $W$ points, and second, the frequency for every itemset in the hash table will be decremented by a value $\Delta f$ every $W$ points.

We apply these biases using smart hash-table management coupled with a down-counting approach described in Figure 5.6.

Our hash table management works as follows. For every $W$ points, we create a new hash table with index $i \div W$ and delete the oldest hash table with index $(i \div W - h)$. Thus, at every instant in time, we maintain at most $h$ hash tables. We estimate the frequency for an itemset based on its frequency in the $h$ hash tables. Moreover, for every $W$ points, relevant itemsets will have their frequencies biased with a value of $-\Delta f$. The oldest hash table is deleted every $W$ points and the $h$ most recent hash tables will contain the relevant itemsets with their biased frequencies.
Hash Table Management:
Let $i$ be the index of the point that just arrived
If $i \mod W = 0$,
    Then create a new hash table with index $i \div W$
delete the hash table with index $i \div W - h$

Frequency Estimation:
When estimating the frequency for an itemset from a point with index $i$:
If an itemset for the $i^{th}$ point is found in hash table with index $i \div W$
    Increment and use the freq. from this hash table
Else
    If itemset is found in hash table with index $i \div W - j$, $j > 0$
        insert it into the hash table with index $i \div W$,
        set freq = freq in table $(i \div W - j) + 1 - j \times \Delta f$
        and then use this freq.
    Else
        insert it into the hash table with index $i \div W$
        with freq. = 1

Figure 5.6: Algorithm for hash table management and frequency estimation

5.2 Experimental Results

5.2.1 Setup

We evaluate our algorithm’s performance in an eight-node cluster, where each node has dual 1 GHz Pentium III processors and 1 GB of memory, running Red Hat Linux 7.2. This set-up also gives us the ability to vary network bandwidth between the nodes and allows us to evaluate performance with varying network bandwidth and latency. Our implementations are in C++ and we use MPI for message passing. We evaluate our algorithm\(^3\) on the following data.

\(^3\)For all experiments unless otherwise noted, we use the following parameter settings: $s = 10$, $\tau = 1.96$, $\delta = 30\%$, $\Delta Score = 10$, $ScoreWindowSize = 40$, $h = 2$. 

97
**KDDCup 1999 Intrusion Detection Data**

The 1999 KDDCup data set [69] contains a set of records that represent connections to a military computer network where there have been multiple intrusions and attacks by unauthorized users. The raw binary TCP data from the network has been processed into features such as *connection duration, protocol type, number of failed logins*, and so forth. This data set was obtained from the UCI KDD archive [19]. The training data set has 4,898,430 data instances with 32 continuous attributes and 9 categorical attributes. The testing data set is smaller and contains several new intrusions that are not present in the training data set. Since these data sets have an unrealistic number of attacks, we preprocess them such that intrusions constitute 2% of the data set, and the proportions of different attacks is maintained. In network traffic, packets tend to occur in bursts for some intrusions. Therefore, intrusion instances are not randomly inserted into the data, but occur in bursts that are randomly distributed in the data set. The processed training data set contains 983,561 instances with 10,710 attack instances, while the processed testing data set contains 61,917 instances with 1,314 attack instances.

**Adult Data**

The Adult data set [19], containing 48,842 data instances with 6 continuous and 8 categorical attributes, was extracted from the US Census Bureau’s Income data set. Each record has features that characterize an individual’s yearly income together with a class label indicating whether the person made more or less than 50,000 dollars per year.
US Congressional Voting Data

The Congressional Votes data set consists of a representative’s votes on 16 issues together with a label indicating if the representative is a Republican or Democrat. The data set has 435 data instances with 16 categorical attributes.

Synthetic Data

Since there are very few publicly available mixed-attribute data sets with labeled outliers, we wrote a synthetic data set generator to produce data to compare performance with existing algorithms, and with varying data set characteristics. The generator can produce data sets with a user-supplied number of continuous attributes ($N_q$) and categorical attributes ($N_c$). The data points are generated according to a multi-modal distribution, and the user supplies the number of modes $M$. Basically, each mode has associated with it an itemset of size $N_c$ and a cluster of points in $N_q$-dimensional space. Therefore, the number of modes determines the number of distinct values for each of the categorical attributes, and so has an effect on the size of the itemset lattice. The cluster of points associated with each itemset results from randomly generating points uniformly in a sphere, and then applying random transformations (scaling, shearing, translating, etc.) to ensure that the clusters have different means and covariances. To create actual data sets for our experiments, we first generate a set of normal points from one distribution, and then separately generate a much smaller set of outliers from another distribution. The two sets are then randomly mixed to produce the final data set. In our experiments, we consider a synthetic data set with 100,000 normal points and 1,000 outliers. We expect the synthetic data set we have generated to be more challenging than the real data sets we use because the
distribution of the outliers is not guaranteed to be significantly different from that of the normal points, and this is empirically verified in our experiments.

5.2.2 Qualitative evaluation

KDDCup 1999

We compare our approach against ORCA [17], a state-of-the-art distance-based outlier detection algorithm which uses the Euclidean and Hamming distances as similarity measures for the continuous and categorical attributes respectively. ORCA finds the top-\(k\) outliers in a data set, with \(k\) being a user-supplied parameter. We set \(k\) equal to the number of outliers in the data set. For the comparison between ORCA and our algorithm, we used the 10% subset of the KDDCup 1999 training data as well as the testing data set, as ORCA did not complete in a reasonable amount of time on the full training data set.

Detection rates for the two approaches are reported in Table 5.1 (Note that “n/a” indicates that the attack was not present in that particular data set). Since the intrusion packets tend to occur in bursts, we mark an intrusion as detected if at least one instance in a burst is flagged as an outlier. Consequently, the detection (true positive) rates are in terms of the number of intrusion bursts detected, while false positive rates are in terms of the number of normal packets marked as outliers. Overall, our detection rates are very good and much better than those of ORCA. Our approach provides an overall detection rate of 95% compared to a detection rate of 45% afforded by ORCA. Furthermore, our approach gives us a false positive rate of 0.35%. This false positive rate is extremely good for anomaly detection schemes, especially considering our high detection rates. ORCA has a false positive rate of 0.43%, but this is not as significant considering its lower detection rates. For the detection rates of the one-pass version of our algorithm, see Table 5.2. Our algorithm’s low false positive rate can be attributed to a better representation for anomalous behavior.
<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Detection Rate (10% Training)</th>
<th>Detection Rate (Testing)</th>
<th>Detection Rate (Training)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Our Approach/ORCA</td>
<td>Our Approach/ORCA</td>
<td>Our Approach</td>
</tr>
<tr>
<td>Apache 2</td>
<td>n/a</td>
<td>100%/0%</td>
<td>n/a</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>94%/63%</td>
<td>90%/100%</td>
<td>91%</td>
</tr>
<tr>
<td>Back</td>
<td>100%/5%</td>
<td>n/a</td>
<td>98%</td>
</tr>
<tr>
<td>FTP Write</td>
<td>28%/88%</td>
<td>n/a</td>
<td>33%</td>
</tr>
<tr>
<td>Guess Password</td>
<td>100%/21%</td>
<td>100%/0%</td>
<td>100%</td>
</tr>
<tr>
<td>Imap</td>
<td>100%/13%</td>
<td>n/a</td>
<td>100%</td>
</tr>
<tr>
<td>IP Sweep</td>
<td>42%/3%</td>
<td>28%/0%</td>
<td>37%</td>
</tr>
<tr>
<td>Land</td>
<td>100%/66%</td>
<td>n/a</td>
<td>100%</td>
</tr>
<tr>
<td>Load Module</td>
<td>100%/100%</td>
<td>n/a</td>
<td>100%</td>
</tr>
<tr>
<td>Multihop</td>
<td>94%/57%</td>
<td>70%/75%</td>
<td>94%</td>
</tr>
<tr>
<td>Named</td>
<td>n/a</td>
<td>100%/40%</td>
<td>n/a</td>
</tr>
<tr>
<td>Neptune</td>
<td>92%/1%</td>
<td>n/a</td>
<td>98%</td>
</tr>
<tr>
<td>Nmap</td>
<td>94%/8%</td>
<td>n/a</td>
<td>91%</td>
</tr>
<tr>
<td>Perl</td>
<td>100%/100%</td>
<td>n/a</td>
<td>100%</td>
</tr>
<tr>
<td>Phf</td>
<td>0%/25%</td>
<td>20%/100%</td>
<td>0%</td>
</tr>
<tr>
<td>Pod</td>
<td>45%/12%</td>
<td>100%/18%</td>
<td>54%</td>
</tr>
<tr>
<td>Port Sweep</td>
<td>100%/13%</td>
<td>100%/3%</td>
<td>100%</td>
</tr>
<tr>
<td>Root Kit</td>
<td>33%/70%</td>
<td>n/a</td>
<td>33%</td>
</tr>
<tr>
<td>Satan</td>
<td>75%/9%</td>
<td>n/a</td>
<td>72%</td>
</tr>
<tr>
<td>Saint</td>
<td>n/a</td>
<td>100%/1%</td>
<td>n/a</td>
</tr>
<tr>
<td>Sendmail</td>
<td>n/a</td>
<td>50%/50%</td>
<td>n/a</td>
</tr>
<tr>
<td>Smurf</td>
<td>24%/11%</td>
<td>21%/0%</td>
<td>22%</td>
</tr>
<tr>
<td>Snmpgetattack</td>
<td>n/a</td>
<td>52%/0%</td>
<td>n/a</td>
</tr>
<tr>
<td>Spy</td>
<td>100%/100%</td>
<td>n/a</td>
<td>100%</td>
</tr>
<tr>
<td>Teardrop</td>
<td>50%/1%</td>
<td>n/a</td>
<td>30%</td>
</tr>
<tr>
<td>Udpstorm</td>
<td>n/a</td>
<td>0%/0%</td>
<td>n/a</td>
</tr>
<tr>
<td>Warez Client</td>
<td>48%/3%</td>
<td>n/a</td>
<td>43%</td>
</tr>
<tr>
<td>Warez Master</td>
<td>0%/15%</td>
<td>n/a</td>
<td>25%</td>
</tr>
<tr>
<td>Xlock</td>
<td>n/a</td>
<td>50%/66%</td>
<td>n/a</td>
</tr>
<tr>
<td>Xsnoop</td>
<td>n/a</td>
<td>100%/100%</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 5.1: Detection rates - (a) Our approach and ORCA on 10% KDDCup1999 training data set (b) entire testing data set, (c) Our approach on entire KDDCup 1999 training data set
obtained by capturing the dependence between categorical attributes and continuous attributes. On the testing data set our approach is able to detect most of the unknown attacks (a problem for almost all of the KDDCup 1999 participants). Note that our approach is a general anomaly detection scheme and has not been trained to catch specific intrusions – one can use this approach in conjunction with a signature detector to handle this problem [99]. In terms of execution time, our algorithm processed the 10% sample of the KDDCup 1999 training data set in approximately 8 minutes. ORCA, on the other hand, took 212 minutes to process the same data set.

**Adult Data**

Here we are unable to make a comparison with ORCA, since this data set does not provide us with labeled outliers. The following data records are marked as the top outliers:

- A 39 year old self-employed male with a doctorate, working in a Clerical position making less than 50,000 dollars per year
- A 42 year old self-employed male from Iran, with a bachelors degree, working in an Executive position making less than 50,000 dollars per year
- A 38 year old Canadian female with an Asian Pacific Islander origin working for the US Federal Government for 57 hrs per week and making more than 50,000 dollars per year.

Apart from these, the other outliers we find tend to be persons from foreign countries that are not well represented in this data set. On inspection, one can affirm that these records are indeed anomalous.
US Congressional Voting Data

Here we are unable to compare with ORCA because, as in the previous case, this data set does not contain labeled outliers. Many of the top outliers contain large numbers of missing values, though there are examples in which this is not the case. One such example is a Republican congressman, who has four missing values, but also voted significantly differently from his party on four other bills. According to the conditional probability tables supplied with the data set, the likelihood of a Republican voting in such a manner is very low.

Synthetic Data

We consider a synthetic data set with 8 categorical attributes and 10 continuous attributes. As ORCA uses hamming distance, it is unable to effectively capture distance between categorical attributes. Consequently, it achieves a detection rate of 55% with a false positive rate of 0.45%. Our algorithm achieves a detection rate of 76.2% with a false positive rate of 0.2% because of a better representation of categorical data. ORCA processed the data set in 350 seconds, while our algorithm processed this same data set in 214 seconds.

Remarks

Our detection rates on the KDDCup and synthetic data sets are better than those of ORCA. However, we would like the reader to note that as the number of categorical attributes in a data set decreases, we expect ORCA to outperform our algorithm. The reason is that as the number of categorical attributes decreases, the size of the itemset lattice also decreases. As a result, there are fewer covariance matrices with which to model the continuous attributes. In the case that there are no categorical attributes in the data, our approach
uses only a single covariance matrix to model the entire data set, a severe approximation. As a result, on such data sets it is not expected to do well when compared to an approach such as ORCA. However, this decrease in accuracy as a function of the number of categorical attributes is difficult to demonstrate empirically, as it is highly dependent upon which attributes contain the most important information in the data set. Closely related to this phenomenon is the effects of our approximation scheme (see Section 5.1.4 and the experimental results in Section 5.2.4), in which we manually restrict the size of the itemset lattice. We emphasize however, that our approach is intended for mixed-attribute data sets (such as network traffic data sets) only, and not for data sets with attributes of a single type. In future, we will examine methods of extending our approach to data sets containing only a single attribute type.

5.2.3 Speedup in a Distributed Setting

First, we explore the speedup obtained when running our distributed outlier detection algorithm on two, four, and eight sites. The KDDCup 1999 training data set and the synthetic data set is split evenly between the sites for this experiment. The other data sets are too small for this experiment. Figure 5.7(a) shows the speedup obtained on the two data sets. As there is only one round of communication, the overall message passing overhead is minimal. Most of the time is spent in the two phases a) building the local model in the first pass and b) finding outliers in the second pass. Consequently, each node works independently, and we see up to 7.7-fold speedup on 8 sites. The speedup appears to be linear. The slight reduction in speedup with an increasing number of sites is due to increasing communication overhead associated with the local model exchange.
Next, we vary the link bandwidth between the sites in our controlled environment in order to simulate a network spanning larger distances. As shown in Figure 5.7(b), for a wide area setting consisting of eight nodes, efficiency varies from a low of 69% to a high of 96%, for link bandwidths equal to 1 MB/s and 100 MB/s respectively. Even when link bandwidth is equal to 10 MB/s, our approach achieves an efficiency of 95%. This is suggestive of good scalability for distributed outlier detection across different subnets in a local area network.

5.2.4 Benefits of the Approximation Scheme

We measure the execution time, detection rate and the false positive rate as a function of the number of itemset lattice levels maintained. The primary benefit of our approximation scheme is that we achieve far better execution times if we maintain fewer lattice levels, as can be seen in Figure 5.8. However, our detection rates decrease as the number of lattice levels decrease. Figure 5.9 shows detection rates and false positive rates with increasing lattice levels on the KDDCup 1999 training and testing data sets. False positive rates are not affected significantly with changing lattice levels. On the other hand, detection rates seem to increase as the number of lattice levels increase to 3, after which they stabilize. This indicates the use of 3-attribute dependencies in the outlier detection process. We also note that the processing rate per 1000 network transactions is within reasonable response time rates even for a lattice level equal to four. Ideally, one would like to tune the application based on the learning curves in an application specific manner, as each application will exhibit a specific dependence behavior. Empirically, it appears from Figure 5.8 that execution time grows quadratically with the number of lattice levels.

Detection and false positive rates for the Adult data set are not available since the data is unlabeled.
Figure 5.7: (a) Speedup (b) Expected efficiency in a wide area network with 8 sites

Figure 5.8: Execution time variation with increasing lattice levels

Figure 5.9: (a) Detection rates (b) False positive rates

106
5.2.5 Execution time variation with number of attributes

We next evaluate how execution time varies with increasing number of categorical and continuous attributes using synthetic data. First, we measure the execution time as we increase the number of categorical attributes, with 10 continuous attributes. Next, we measure the execution time as we increase the number of continuous attributes, with 4 categorical attributes. As can be seen in Figure 5.10, in both cases there appears to be a polynomial dependency between the execution time and the number of categorical and continuous attributes.

5.2.6 Evaluation of one pass approach
Impact on detection quality

First, we evaluate the detection quality of our one-pass algorithm in a centralized setting on the KDDCup 1999 and synthetic data sets. Comparing Tables 5.1 and 5.2 tells us that detection quality does not deviate significantly from that of our two-pass approach on the KDDCup 1999 data set. The false positive rate remains unaffected. This trend also holds
<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Detection rate Centralized Training/Testing</th>
<th>Detection rate Distributed (2 sites) Training/Testing</th>
<th>Detection rate Distributed (4 sites) Training/Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 2</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>91%/90%</td>
<td>91%/90%</td>
<td>91%/90%</td>
</tr>
<tr>
<td>Back</td>
<td>97%/n/a</td>
<td>97%/n/a</td>
<td>96%/n/a</td>
</tr>
<tr>
<td>FTP Write</td>
<td>33%/n/a</td>
<td>33%/n/a</td>
<td>33%/n/a</td>
</tr>
<tr>
<td>Guess Password</td>
<td>100%/100%</td>
<td>100%/100%</td>
<td>100%/100%</td>
</tr>
<tr>
<td>Imap</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
</tr>
<tr>
<td>IP Sweep</td>
<td>37%/28%</td>
<td>35%/28%</td>
<td>32%/28%</td>
</tr>
<tr>
<td>Land</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
</tr>
<tr>
<td>Load Module</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
</tr>
<tr>
<td>Multihop</td>
<td>94%/100%</td>
<td>94%/100%</td>
<td>94%/100%</td>
</tr>
<tr>
<td>Named</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
</tr>
<tr>
<td>Neptune</td>
<td>98%/n/a</td>
<td>96%/n/a</td>
<td>94%/n/a</td>
</tr>
<tr>
<td>Nmap</td>
<td>91%/n/a</td>
<td>90%/n/a</td>
<td>90%/n/a</td>
</tr>
<tr>
<td>Perl</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
</tr>
<tr>
<td>Phf</td>
<td>0%/20%</td>
<td>0%/20%</td>
<td>0%/20%</td>
</tr>
<tr>
<td>Pod</td>
<td>53%/100%</td>
<td>52%/100%</td>
<td>52%/100%</td>
</tr>
<tr>
<td>Port Sweep</td>
<td>100%/100%</td>
<td>100%/98%</td>
<td>100%/97%</td>
</tr>
<tr>
<td>Root Kit</td>
<td>33%/n/a</td>
<td>33%/n/a</td>
<td>33%/n/a</td>
</tr>
<tr>
<td>Satan</td>
<td>72%/n/a</td>
<td>71%/n/a</td>
<td>70%/n/a</td>
</tr>
<tr>
<td>Saint</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
</tr>
<tr>
<td>Sendmail</td>
<td>n/a/50%</td>
<td>n/a/50%</td>
<td>n/a/50%</td>
</tr>
<tr>
<td>Smurf</td>
<td>22%/21%</td>
<td>21%/20%</td>
<td>20%/20%</td>
</tr>
<tr>
<td>Snmpgetattack</td>
<td>n/a/52%</td>
<td>n/a/52%</td>
<td>n/a/52%</td>
</tr>
<tr>
<td>Spy</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
<td>100%/n/a</td>
</tr>
<tr>
<td>Teardrop</td>
<td>49%/n/a</td>
<td>49%/n/a</td>
<td>49%/n/a</td>
</tr>
<tr>
<td>Udpstorm</td>
<td>n/a/0%</td>
<td>n/a/0%</td>
<td>n/a/0%</td>
</tr>
<tr>
<td>Warez Client</td>
<td>43%/n/a</td>
<td>42%/n/a</td>
<td>41%/n/a</td>
</tr>
<tr>
<td>Warez Master</td>
<td>0%/n/a</td>
<td>0%/n/a</td>
<td>0%/n/a</td>
</tr>
<tr>
<td>Xlock</td>
<td>n/a/50%</td>
<td>n/a/50%</td>
<td>n/a/50%</td>
</tr>
<tr>
<td>Xsnoop</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
<td>n/a/100%</td>
</tr>
</tbody>
</table>

Table 5.2: Detection rates for the single pass approach (Centralized and Distributed) for KDDCup 1999 data
true for the synthetic data set, on which the detection rate fell to 0.75 (from 0.762) with the false positive rate remaining the same. Next, we compare our algorithm’s detection quality as we scale up from a centralized to a distributed setting with two and four sites. Table 5.2 presents detections rates for different attacks in a centralized and distributed setting for the KDDCup 1999 data set. Our detection rates remain nearly unchanged as we move to two and four sites. The slight reduction in detection rate is attributed to data points that are flagged as local normals when they are global outliers. Similarly, the detection rate fell to 0.74 on the synthetic data set with four sites. On both the data sets, false positive rate remains unaffected. One can see that in practice, the number of global outliers that are being missed is not that significant.

**Scalability**

We explore the speedup obtained when running our distributed one pass outlier detection algorithm on two, three and four sites. The KDDCup 1999 and synthetic data sets are evenly split between the nodes for this experiment. Figure 5.11(a) shows speedup obtained on the two data sets. Since there are relatively few outliers in the data set, and we have a low false positive rate, there is very little communication overhead. Therefore, there is very little synchronization between the sites and each site is able to work independently. As the number of nodes increases, the communication overhead also increases, as more nodes are involved in the local outlier exchange. As a result we see a slight reduction from the ideal speedup. For this reason, efficiency falls to 95% on the two data set when using four sites.

As nodes primarily communicate by exchanging outliers, which are small messages, link latency will be the primary performance bottleneck in a wide area setting. We vary the average link latency between the nodes in our controlled environment to simulate a wide area network spanning larger distances. As shown in Figure 5.11(b), efficiency falls
to 90% for the KDDCup 1999 data set when using 4 sites, with an average link latency of 25ms. This is representative of networks spanning across several states (for instance, we have a 25ms average latency between a machine located at The Ohio State University and the University of Rochester) and excellent scalability. We would like to note that execution time per 1000 transactions for the one-pass approach does not change significantly when compared to the two pass approach. This is because the cumulative work done in the one-pass and two-pass approaches is nearly the same.

5.3 Conclusions

Outlier detection is an important problem for a wide range of data mining applications. To date, there have not been any general-purpose outlier detection algorithms that work on mixed attribute data in a distributed setting. In this chapter, we presented the first general-purpose distributed outlier detection algorithm that addresses these concerns. We also extended our algorithm to handle dynamic and streaming data sets. Experimental results validate the effectiveness of our approach on several real and synthetic data sets. In
the future, we will examine alternative approaches for building models of mixed attribute data sets that have reduced memory requirements and allow for faster processing. We will also explore techniques for extending our approach to data sets containing only a single attribute type.
CHAPTER 6

REDUCED-MEMORY OUTLIER DETECTION

In Chapter 5, we discussed variations on the LOADED algorithm, which discovers outliers in dynamic data sets containing a heterogeneous mixture of continuous and categorical attribute types. However, LOADED suffers from high memory requirements. In Section 5.1.4 we showed that reducing these memory requirements while maintaining the same detection accuracy results in an increase in computational complexity. In this chapter we propose an alternative to LOADED named RELOADED that drastically reduces the memory requirements of LOADED, as well as improves on its scalability, at a small cost to accuracy.

6.1 A Modified Outlier Detection Algorithm

As mentioned in Chapter 5, the original LOADED algorithm and its variations assign a data point an outlier score based on the support of that point’s itemsets (for the categorical attributes), and the divergence from the expected correlation or covariance values (for the continuous attributes). To that end, LOADED maintains a complete itemset lattice in memory, where each node of the lattice is augmented by a correlation or covariance matrix. Since each matrix is quadratic in size with respect to the number of continuous attributes, and the itemset lattice is exponential in size with respect to the number of categorical attributes, LOADED requires a relatively large amount of memory (see Section 5.1.4).
To address this memory consumption issue, we propose the **RELOADED** (*RE*duced memory *LOADED*) algorithm. Like ** LOADED**, **RELOADED** uses a set of covariance matrices to discover anomalous values of continuous attributes, but unlike ** LOADED**, it uses a set of classifiers instead of an itemset lattice to detect anomalous values of categorical attributes. To characterize the relationships between the categorical and continuous attributes, the covariance matrices are conditioned on the values of the categorical attributes, while the classifiers take into account the values of the continuous attributes. We can then define an anomalous data point as one that has a subset of attributes that take on unusual values given the values of the other attributes. The details of our approach are given below.

### 6.1.1 Predicting Values of Categorical Attributes

Consider a data point $P = (P_c, P_q)$ where $P_c$ is a vector of $m$ categorical attributes and $P_q$ is a vector of $n$ continuous (quantitative) attributes. To discover anomalous values of categorical attributes, we use a set of $m$ classifiers to predict the values of each of the $m$ categorical attributes and then compare against their actual values. The classifier $C_i$ used to predict the categorical attribute $P_{ci}$ is trained on data points of the form $P^i = (P^i_c, P_q)$, where $P^i_c$ is the vector of categorical attributes with the $i$th element removed. In turn, the value of the $i$th attribute is used as the class label. For example, assume that our data set has data points of the follow form:

$$P = (N_1, \ldots, N_m, Q_1, \ldots, Q_n)$$

where the $N$’s and $Q$’s are the categorical and continuous attributes, respectively. We use $n$ classifiers $C_1$ through $C_m$ to predict the values of the categorical attributes $N_1$ through $N_n$:

$$\hat{N}_i = C_i(N_1, \ldots, N_{i-1}, N_{i+1}, \ldots, N_m, Q_1, \ldots, Q_n)$$
If for attribute $i$, the predicted value $\hat{N}_i$ disagrees with that of the true value $N_i$, we increase the value of our anomaly score for that point. In this work, we utilize Naïve Bayes Classifiers, as they are relatively light-weight classifiers and can be trained incrementally, which is useful for our single-pass approach to anomaly detection. To train each classifier, we only need to calculate and store the probability mass function for each attribute, and since we assume that the distributions of the continuous attributes to be normal, we only need to store the mean and standard deviation for each continuous attribute.

6.1.2 Finding Anomalous Values of Continuous Attributes

Again, consider the data point $P = (P_c, P_q)$. We wish to discover to what degree $P_q$ violates the covariances conditioned on the values of the categorical attributes. First, note that $P_c$ can be considered as a set of attribute-value pairs:

$$P_c = \{(\text{attribute}_1, \text{value}_1), \ldots, (\text{attribute}_m, \text{value}_m)\}.$$

Recall from Chapter 5 that LOADED maintains correlation or covariance matrices for all subsets of all the $P_c$ in the data set, which is the root cause of its high memory consumption problem. To reduce memory consumption, RELOADED only maintains a covariance matrix for each unique attribute-value pair in the data set. To illustrate the magnitude of this reduction, consider a data set with $m$ categorical attributes, where the $i$th attribute can take on $r_i$ distinct values. In this case, RELOADED must maintain $\sum_{i=1}^{m} r_i$ covariance matrices in the worst case, while LOADED must maintain on the order of $2^m \prod_{i=1}^{m} r_i$ matrices in the worst case.

As stated above, for each unique attribute-value pair in the data set, there is a covariance matrix $C$. Let $C^d$ be the covariance matrix for the attribute-value pair $d$, where $C^d_{ij}$ is the covariance between continuous attributes $i$ and $j$. To determine the degree to which $P$
violates $C_{ij}^d$, we calculate the covariance score $c_{ij}^{P,d}$ for each pair of continuous attributes $i$ and $j$ of $P$:

$$c_{ij}^{P,d} = (P_{qi} - \mu_i^d) \times (P_{qj} - \mu_j^d)$$  \hspace{1cm} (6.1)$$

where $\mu_i^d$ and $\mu_j^d$ are the means of continuous attributes $P_{qi}$ and $P_{qj}$ for all points in the data set containing the attribute-value pair $d$. We can then compute the violation score $V$ of $P_q$ as:

$$V_\tau(P_q) = \sum_i \sum_j v_\tau(P_{qij})$$  \hspace{1cm} (6.2)$$

where

$$v_\tau(P_{qij}) = \begin{cases} 
0 & \text{if } |c_{ij}^{P,d}| \leq \tau \\
1 & \text{otherwise.} 
\end{cases}$$  \hspace{1cm} (6.3)$$

The function $v_\tau$ captures how much the covariance score of a pair of continuous attributes deviates from the covariance learned from the data set. If a pair has a deviation greater than $\tau$, the violation score is incremented.

### 6.1.3 The RELOADED Algorithm

The RELOADED algorithm for anomaly detection can be seen in Figure 6.1. The version presented in the figure is the single-pass version, which trains the classifiers and computes the covariance matrices incrementally. Therefore, the decision as to whether a given point is an anomaly or not is based only on the previously processed data points. The algorithm operates as follows. For each point in the data set, and for each categorical attribute $d$ of that data point, the appropriate classifier is trained. That classifier, in turn, is used to predict the appropriate value of $d$. If the prediction is wrong, the count of incorrect predictions is incremented. Next, the continuous attributes of the data point are used to incrementally compute the covariance matrix corresponding to the attribute-value pair $d$. 

115
Procedure: Reloaded

Input: DataSet, \( \tau \)

Output: AnomalyScores

1. For each point \( P = (P_c, P_q) \in DataSet \):
2. \( \#WrongPred = 0 \)
3. \( VScore = 0 \)
4. For each attribute-value pair \( d \in P_c \):
5. \( P'_c = P_c \) with \( d \) removed
6. Train \( Classifier_d \) on \( P'_c \)
7. \( Pred_d = Classifier_d(P'_c, P_q) \)
8. if \( (d \neq Pred_d) \)
9. \( \#WrongPred + + \)
10. Use \( P_q \) to update \( Covariance_d \)
11. \( VScore + = V_\tau(P_q) \)
12. End For
13. \( AnomalyScore[P] = f(\#WrongPred, VScore) \)
14. End For

end

Figure 6.1: The RELOADED algorithm

The cumulative violation score of the data point is incremented by the result of the \( V_\tau \) function (see Equations 6.2 and 6.3).

When all categorical attributes of a data point have been processed, the anomaly score of the data point is computed as a function \( f \) of the count of incorrect predictions and the violation score. The function \( f \) allows one to weight anomalies in the categorical and continuous attributes differently. One can see that if there are \( m \) categorical attributes and \( n \) continuous attributes, \( VScore \) can achieve a maximum value of \( m \times n^2 \), while there can only be \( m \) incorrect predictions of the categorical attribute values, meaning a simple summation or product of the two values would be much more sensitive to anomalous continuous values. In our implementation, we use the following function for point \( P_i \):

\[
AnomalyScore[P_i] = \frac{\left(\sum_{j=1}^{m} i - W_j \right)}{m} + \frac{V_\tau}{mn^2} \tag{6.4}
\]
where $W_j$ is the cumulative number of incorrect predictions of categorical attribute $j$ for previous $i$ data points. The term $\frac{i-W_j}{i}$ is the fraction of points so far where attribute $j$ was correctly predicted, meaning that the misprediction penalty is weighted by fraction of time in which the prediction is correct. Such a formula ensures that the categorical and continuous attributes have equal weight, and that categorical attributes with little or no correlation with the other attributes will not adversely affect the anomaly score (such attributes will have a large value for $W_j$, since they are often mispredicted). Finally, we must choose a threshold for $AnomalyScore$ in order to discriminate between the outliers and normal points. We do this by incrementally computing the mean and standard deviation of the $AnomalyScores$ seen so far, and flag any point as an outlier if it is more than $s$ standard deviations greater than the current mean.

We note that it is straightforward to convert the single-pass algorithm presented in Figure 6.1 into a two-pass algorithm for static data. We simply make two passes over the data set. In the first pass, we omit steps 2-3, 7-9, 11 and 13, while in the second pass we omit steps 6 and 10. We also note that the version of RELOADED presented above places equal weight on all previously processed points. Since older points are usually less relevant than more recent ones, we can implement a sliding-window variant that periodically instantiates new classifiers and covariance matrices and disposes of older ones.

### 6.1.4 Computational and Space Complexity

Consider a data set with $N$ data points, having $m$ categorical attributes and $n$ continuous attributes, and let each of the $m$ categorical attributes take on a maximum of $k$ distinct values. The processing of a data point has two stages. In the first stage, $m$ Naïve Bayes Classifiers are run to predict the values of each of the $m$ categorical attributes based on
the remaining attributes. If the classifier’s probability mass functions are stored in hash tables, the time it takes to train a single classifier on a single point is $O(m + n)$. For classification of a point, for each of the $m$ categorical attributes there are at most $k$ classes we must consider. The time necessary for computing the probability of a point belonging to a class is $O(m + n)$, and so the time required to predict the values of each of the $m$ categorical attributes is $O(mk(m + n)) = O(km^2 + kmn)$.

In the second stage we must incrementally compute the covariance matrix for the set of continuous attributes for each attribute-value pair of a given point. The time it takes to incrementally compute the covariance matrix is $O(n^2)$ for a given attribute-value pair. Since there are $m$ such attribute-value pairs, the total time taken to compute all covariances is $O(mn^2)$. Therefore the upper bound on the amount of time taken for each data point is $O(mn^2 + km^2 + kmn)$, which gives an upper bound of $O(N(mn^2 + km^2 + kmn))$ for the algorithm running on the entire data set. As can be seen, the algorithm is linear in the number of data points and the number of categorical attribute values, and quadratic in the number of categorical and continuous attributes. For comparison, LOADED has a worst-case execution time of $O(Nn^22^n)$ (see Section 5.1.4). Like Reloaded, it is linear in the number of data points and quadratic in the number of continuous attributes, but it is exponential in the number of categorical attributes. For comparison, the ORCA outlier detection algorithm [17] has a worst-case complexity of $O(N^2(n + m))$, though empirically it appears to have complexity $O(N^d(n + m))$, where $d$ is between 1 and 2.

In terms of space complexity, Reloaded requires $O(m(km + n))$ space to hold the classifiers, and $O(kmn^2)$ space to hold the covariance matrices. The space requirements of our algorithm are thus independent of the size of the data set, and quadratic in the number
of continuous and categorical attributes. This compares very favorably to LOADED, which requires $O(n^2(k + 1)^m)$ space in the worst case (see Section 5.1.4).

6.2 Experimental Results

6.2.1 Setup

We evaluate RELOADED’s performance with regard to detection quality, memory usage, and execution time. and compare it to that of LOADED and ORCA using a machine with a 2.8 GHz Pentium IV processor, and 1.5 GB of memory, running Mandrake Linux 10.1. Our implementations are in C++ and are compiled using gcc with O2 optimizations. Since ORCA finds the top-$k$ outliers in a data set, we set $k$ equal to the number of outliers in the data set. Also, we set LOADED to use only 4 lattice levels in all experiments. We compare these algorithms using the following data sets.

KDDCup 1999 Intrusion Detection Data

The 1999 KDDCup data set [69] contains a set of records that represent connections to a military computer network where there have been multiple intrusions and attacks. This data set was obtained from the UCI KDD archive [19]. The original training data set has 4,898,430 data instances with 32 continuous attributes and 9 categorical attributes. The testing data set is smaller and contains several intrusions that are not present in the training data set. Since these data sets have an unrealistic high number of attacks, we process them so that the intrusions constitute only 2% of the data set, and the proportions of the different attacks is maintained. In network traffic, for some intrusions and attacks, packets tend to occur in bursts. Therefore, in the processed data sets, intrusion instances are not randomly inserted into the data, but occur in bursts that are randomly distributed through the data set. The final version of the processed training data set contains 983,561 instances with 10,710
attack instances, while the final version of the processed testing data set contains 61,917 instances with 1,314 attack instances.

**Adult Data**

The Adult data set [19], contains 48,842 data instances with 6 continuous and 8 categorical attributes. Since the algorithms we test differ in their abilities to handle missing data, we removed all records containing missing data, leaving 32,561 records. The data was extracted from the US Census Bureau’s Income data set. Each record contains an individual’s demographic attributes together with a class label indicating whether the person made more or less than 50,000 dollars per year.

**Synthetic Data**

Since there are very few publicly available large mixed-attribute data sets, we wrote a synthetic data set generator to produce data to compare performance with existing algorithms, and with varying data set characteristics. The generator can produce data sets with a user-supplied number of continuous attributes and categorical attributes. The data points are generated according to a user-supplied multi-modal distribution. The exact details can be found in Section 5.2.1. To create actual data sets for our experiments, we first generate a set of normal points from one distribution, and then separately generate a much smaller set of outliers from another distribution. The two sets are then randomly mixed to produce the final data set. However, we cannot guarantee that the distribution of the outliers is significantly different from that of the normal points. Therefore, the synthetic data sets are chiefly used for memory and execution time scaling experiments. In our experiments, we consider a synthetic data set containing a 1% mixture of outliers.
6.2.2 Detection Rate

Our first set of experiments compares the detection rates of RELOADED to both LOADED and ORCA. In particular we compare the detection rate of both the one- and two-pass versions of RELOADED to that of the two-pass version of LOADED, and ORCA. For RELOADED, we set $s$ equal to 2 and $\tau$ equal to 20 for the KDDCup testing data set, and to 1.75 and 12 respectively for the training data set. We do not include results for ORCA on the KDDCup training data set, as it cannot complete in a reasonable amount of time.

Detection rates for all the different algorithms are reported in Table 6.1 (Note that “n/a” indicates that the attack was not present in that particular data set). Since the intrusion packets tend to occur in bursts in our data set, we mark an intrusion as detected if at least one instance in a burst is flagged as an outlier. This is realistic, since a network administrator needs to be alerted only once that an intrusion is underway. Consequently, the detection (true positive) rates in the table are in terms of the number of intrusion bursts detected. The highest detection rates for each intrusion type are in bold. We report false positive
<table>
<thead>
<tr>
<th>Attack</th>
<th>KDDCup Testing</th>
<th>KDDCup Training</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reloaded</td>
<td>Loaded</td>
</tr>
<tr>
<td>Apache2</td>
<td><strong>100% (100%)</strong></td>
<td>100%</td>
</tr>
<tr>
<td>Back</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>72% (7%)</td>
<td>90%</td>
</tr>
<tr>
<td>FTP Write</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Guess Password</td>
<td>50% (30%)</td>
<td><strong>100%</strong></td>
</tr>
<tr>
<td>Imap</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>IP Sweep</td>
<td><strong>100% (45%)</strong></td>
<td>28%</td>
</tr>
<tr>
<td>Land</td>
<td><strong>100% (100%)</strong></td>
<td>0%</td>
</tr>
<tr>
<td>Load Module</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Multihop</td>
<td>63% (38%)</td>
<td>70%</td>
</tr>
<tr>
<td>Named</td>
<td>67% (33%)</td>
<td><strong>100%</strong></td>
</tr>
<tr>
<td>Neptune</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Nmap</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Perl</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Phf</td>
<td>80% (20%)</td>
<td>20%</td>
</tr>
<tr>
<td>Pod</td>
<td>96% (21%)</td>
<td><strong>100%</strong></td>
</tr>
<tr>
<td>Port Sweep</td>
<td><strong>100% (92%)</strong></td>
<td>100%</td>
</tr>
<tr>
<td>Root Kit</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Saint</td>
<td><strong>100% (100%)</strong></td>
<td><strong>100%</strong></td>
</tr>
<tr>
<td>Satan</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Sendmail</td>
<td>17% (0%)</td>
<td><strong>50%</strong></td>
</tr>
<tr>
<td>Smurf</td>
<td><strong>98% (31%)</strong></td>
<td>21%</td>
</tr>
<tr>
<td>Snmpgetattack</td>
<td>0% (0%)</td>
<td><strong>52%</strong></td>
</tr>
<tr>
<td>Spy</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Teardrop</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Udpstorm</td>
<td>0% (0%)</td>
<td>0%</td>
</tr>
<tr>
<td>Warez Client</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Warez Master</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Xlack</td>
<td>50% (100%)</td>
<td>50%</td>
</tr>
<tr>
<td>Xsnoop</td>
<td><strong>100% (100%)</strong></td>
<td><strong>100%</strong></td>
</tr>
</tbody>
</table>

Table 6.1: Detection rates for the KDDCup data sets. The best detection rates are in bold. The detection rates for the single-pass version of RELOADED is given in parenthesis.
Algorithm & KDDCup (Test) & KDDCup (Train) & Adult \\
--- & --- & --- & --- \\
RELOADED & 623 & 852 & 291 \\
LOADED & 49,328 & 595,280 & 58,316 \\
ORCA & 599 & n/a & 390 \\

Table 6.2: Peak heap usage in kilobytes.

rates in terms of the number of normal packets marked as outliers. LOADED has a false positive rate of 0.35%, which is good considering its high detection rates for many of the intrusions. ORCA has a false positive rate of 0.43%, but this is not as significant considering its low detection rates. RELOADED has detection rates comparable to LOADED on many intrusions, and does very well on a handful of intrusions (e.g. IP sweep and smurf) on which both LOADED and ORCA do poorly. However, the two-pass version of RELOADED has higher false positive rates of 1.5% for the testing data set and 3.6% for the training data set, but this is to be expected since it builds a less intricate model of the data in order to save on memory. The one-pass version of RELOADED has a relatively low 0.38% false positive rate on the testing data, but it also has only a 27% detection rate. On the training data, the detection rate is 63%, but the false positive rate is 15%. We note that for this comparison, we use the same parameters for $s$ and $\tau$ for the one-pass version as we do for the two-pass version. These values are not the optimum values for the single-pass version as the algorithm makes a decision as to whether a point is an outlier using a model derived from fewer points than the two-pass version. Indeed, different values of $s$ and $\tau$ give better detection and false positive rates.
6.2.3 Memory Usage

We first compare the memory usage of \textit{RELOADED} to that of both \textit{LOADED} and \textit{ORCA} when they are run on the KDDCup and Adult data sets. For \textit{RELOADED} and \textit{LOADED} we use single-pass approaches, as the amount of memory used does not vary with the number of passes. We set \textit{ORCA} to find the top 1,314 outliers in the KDDCup testing data set and the top 30 outliers in the Adult data set. As mentioned before, \textit{ORCA} does not finish in a reasonable amount of time on the KDDCup training data set. We measure memory usage by looking at the peak heap usage measured in kilobytes. The results can be seen in Table 6.2. Both \textit{RELOADED} and \textit{ORCA} consume less than one megabyte of memory, while \textit{LOADED} uses \textit{two to three orders of magnitude more memory}, even when we constrain the lattice to 4 levels. If we examine the cache performance on \textit{RELOADED} when run on the KDDCup testing data set, we find that it has 0.0003 L2 cache misses per instruction, which is indicative of good temporal locality. This is due to the fact that \textit{RELOADED}’s model of the data is compact enough to fit in L2 cache.

Unlike \textit{ORCA}, \textit{LOADED} and \textit{RELOADED} have greater than linear space complexity with respect to the number of categorical and continuous attributes, and so we empirically test how their peak memory usage scales as the number and types of attributes vary. In Figure 6.2(a) we plot peak memory usage versus the number of categorical attributes, while setting the number of continuous attributes equal to 5. It is evident that the memory requirements of \textit{LOADED} are very large and grow exponentially as the number of categorical attributes increase, while those of \textit{RELOADED} grow much more slowly. Note that we cannot run \textit{LOADED} on data sets with more than 15 categorical attributes, as our machines do not have sufficient memory. In Figure 6.2(b) we set the number of categorical attributes equal to 5 and then vary the number of continuous attributes. The peak memory requirements of
both RELOADED and LOADED increase at about the same rate, which is expected as they both use space that is quadratic in the number of continuous attributes. Note that even for 5 categorical attributes, LOADED requires significantly more memory to maintain the itemset lattice.

### 6.2.4 Execution Time

In our next set of experiments we compare the execution times of RELOADED, LOADED and ORCA. We use the single-pass versions of both RELOADED and LOADED. In our first experiment, we measured the execution times on the KDDCup testing data set. RELOADED takes 47 seconds to complete, compared to 109 seconds for LOADED and 303 seconds for ORCA. Next, we examine how execution time scales with the number of data points processed. We use synthetic data with 10 categorical and 5 continuous attributes. The results can be seen in Figure 6.3. For small data sets, ORCA out-performs both LOADED and RELOADED, but since it does not scale linearly, this advantage is lost for larger data sets. As we expect, the execution times of both RELOADED and LOADED scale linearly with the number of points. Note that LOADED does not scale as well as RELOADED, as it must compare each point with and update $\sum_{i=1}^{4} \binom{10}{i} = 385$ covariance matrices and itemsets, whereas RELOADED need only compare each point with and update 10 classifiers and covariance matrices.

While ORCA’s time complexity is linear in both the number of categorical and continuous attributes, RELOADED’s and LOADED’s complexity is not, and so in our next two experiments we compare how the execution of times of both RELOADED and LOADED scale for data sets with varying numbers of categorical and continuous attributes. In our first experiment, we set the number of continuous attributes equal to 5 and vary the number
Figure 6.3: Plot of execution time versus data set size.

Figure 6.4: Plot of execution time versus number of categorical attributes.
of categorical attributes from 1 to 15. The results can be seen in Figure 6.4. Though we limit the size of LOADED’s lattice, its execution time still increases exponentially with the number of categorical attributes, while that of RELOADED increases quadratically. In our second experiment we examine the scalability of both algorithms with respect to the number of continuous attributes. In this experiment we set the number of categorical attributes equal to 5 and vary the number of continuous attributes from 1 to 25. The results can be seen in Figure 6.5. For smaller numbers of continuous attributes, LOADED is more efficient than RELOADED, but RELOADED scales better for larger numbers of continuous attributes.

6.3 Conclusions and Future Work

In this paper we have presented a general-purpose outlier detection algorithm named RELOADED that is capable of discovering outliers in dynamic mixed-attribute data. It is designed to minimize both memory consumption and execution time. Our experimental results show that for about the same memory usage, RELOADED outperforms ORCA in terms of speed and detection rates. While RELOADED does not outperform LOADED in

Figure 6.5: Plots of execution time versus number of continuous attributes.
terms of detection rates, it does in terms of memory usage and speed. RELOADED’s low execution times and memory consumption make it a good candidate for embedded outlier detection systems, such as might be found in network interface card-based intrusion detection (see Chapter 3), or sensor networks. In future, we plan to examine alternatives to the Naïve Bayes Classifiers we currently use. Finally, we also plan to look at a hybrid RELOADED/LOADED approach to increase detection rates.
CHAPTER 7

A DISSIMILARITY MEASURE FOR COMPARING SUBSETS OF DATA

7.1 Introduction

Quantifying the similarity of (or the dissimilarity of or distance between) two objects is a central concept in data mining, as it forms the basis of classification, clustering, frequent itemset mining, etcetera. It is also the central element of abnormality detection, as abnormal objects are those that are most similar to some pattern (in the case of signature detection), or those objects which are most dissimilar from all other objects in a database (in the case of anomaly detection). Research in this area of quantifying similarity has primarily progressed along two fronts: object similarity [5, 75, 56] and attribute similarity [37, 146]. The former quantifies the distance between two objects (rows) in the database, while the latter refers to the distance between attributes (columns).

A related problem is that of determining the similarity or dissimilarity of two subsets of data. Most often an object is represented as a vector of attributes in a data set. However there are times when the objects of interest are better described by a matrix of attributes. Examples of such objects include multivariate time series, digitized images, and classes of data, where the matrix contains all data points (in vector format) belonging to a given
class. Basic approaches have involved using classification [67], clustering [76], and mining contrast sets [16]. However, these approaches build models of the data sets, instead of quantifying their differences. In Chapter 4 we examined high-contrast frequent itemsets as a means of quantifying the dissimilarity between the portions of a distributed data set held at different nodes.

In this chapter we examine the notion of quantifying the dissimilarity between different subsets of data. We propose a novel dissimilarity measure that can be used to quantify the differences between two data sets. We use the term *dissimilarity* instead of *distance*, since our measure may find that two data sets are very dissimilar even when they are relatively close to each other with respect to some distance metric (see our example in Section 7.2.2).

One motivating application for such a measure could be for analyzing clinical drug trials to detect the efficacy and hepatotoxicity of drugs. Here one can view each patient in the trial as a different data set, for which multiple observations at varying time points of various analytes are measured and stored. A dissimilarity measure in this context can help cluster patients into similar groups or alternatively detect anomalous patients, who may be prone to suffer from hepatotoxicity (we will return to this application in Chapter 8). Another application could be in financial stock market analysis where different subsets of the data (for example, different sectors or time periods) can be examined for change point detection, anomaly detection and clustering. This requires the development of a suitable dissimilarity measure.

Such a dissimilarity measure is useful in fields where it is important to discover the differences in the correlation structures of multiple multivariate data sets. For example, in network intrusion detection one may have a data set describing the number of bytes sent and received by each computer in a subnetwork. When comparing subsets of this data
set taken from different time points, changes in the overall mean may not be as important as changes in the correlations of the number of bytes received by different machines, as this may be indicative of one or more machines suffering from a denial of service or other attack. Another example of the utility of our dissimilarity measure is its usefulness for analyzing multivariate time series. It can be used to measure the similarity of two different time series, or two segments of the same time series, which is useful for such things as clustering and change point detection.

A suitable dissimilarity measure has several requirements. First, it must take into account as much of the information contained in the data sets as possible. For example, simply calculating the Euclidean distance between the centroids of two data sets is ineffective, as this approach ignores the correlations present in the data sets. Second, it must be user-tunable in order to account for domain knowledge. For example, in some domains it may be that differences in the means of two data sets may not be as important as differences in their correlation structures. In this case, differences in the mean should be weighted less than differences in the correlations. Third, the dissimilarity measure should be tolerant of missing and noisy data, since in many domains data collection is imperfect, leading to many missing attribute values.

In this chapter we propose a novel dissimilarity metric based on principal component analysis (PCA). Our measure consists of three components that separately take into account differences in the locations, correlations, and variances of the data sets being compared. As such, our measure takes into account much of the information in the data set. It is also possible to weight the components differently, so one can incorporate domain knowledge into the measure. Finally, our measure is robust towards noise and missing data. We demonstrate the efficacy of the proposed metric in a variety of application domains, including
anomaly detection, change detection and data set clustering, on both synthetic and real
data sets.

7.2 Algorithms

In this section we first present our dissimilarity measure and demonstrate its effectiveness with a small example data set. We then discuss in detail various applications of our dissimilarity measure that demonstrate its utility and flexibility.

7.2.1 Dissimilarity Measure

Our goal is to quantify the dissimilarity of two homogeneous $k$-dimensional data sets $\mathbf{X}$ and $\mathbf{Y}$. This measure of dissimilarity should take into account not only the distances between the data points in $\mathbf{X}$ and $\mathbf{Y}$, but the correlations between the attributes of the data sets as well.

In general, the dissimilarity of two data sets $\mathbf{X}$ and $\mathbf{Y}$ is denoted as $D(\mathbf{X}, \mathbf{Y})$. We define the function $D$ in terms of three dissimilarity functions that take into account the differences in location, rotation, and variance between the data sets. Each of these components are discussed separately below. These three components are combined by means of a product or weighted sum, the latter of which allows one to weight the components differently, so as to incorporate domain knowledge. For example, in the domain of network intrusion detection, one may be concerned with time series data sets where column $i$ represents the $i$th computer on a given subnetwork, and row $j$ represents the number of bytes received between times $t_{j-1}$ and $t_j$. When comparing subsets of this data set taken from different time points, large differences in the mean may be indicative of a denial-of-service attack. Alternatively, differences in the correlation of the number of bytes received by two different
machines may be indicative of one of the machines being used by an unauthorized user. Depending on what the user wishes to detect, the measure can be tuned in different ways.

**Distance Component**

To determine the distance between two data sets, there are a wide variety of distance metrics we can use. We have implemented several different distance metrics, including the single-link and complete-link distances, among others (see Section 4.2). In this work we primarily use the Euclidean distance between the centroids of each data set:

\[
D_d(\mathbf{X}, \mathbf{Y}) = |\mu_X - \mu_Y|^2.
\]

(7.1)

However, in some cases we will also use the Mahalanobis distance.

**Rotation Component**

The next component measures the degree to which the data set \( \mathbf{X} \) must be rotated so that its principal components point in the same direction as those of \( \mathbf{Y} \). The principal components of a data set are the set of orthogonal vectors such that the first vector points in the direction of greatest variance in the data, the second points in the orthogonal direction of the second greatest variance in the data, and so on [79, 129]. We consider \( \mathbf{X} \) and \( \mathbf{Y} \) to be most similar to each other when their principal components, paired according to their ranks, are aligned, and most dissimilar when all of the components of \( \mathbf{X} \) are orthogonal to those of \( \mathbf{Y} \).

More formally, given a data set \( \mathbf{X} \), consider the singular value decomposition (SVD) of its covariance matrix:

\[
\text{cov}(\mathbf{X}) = U \Lambda_X X^T.
\]

(7.2)
where the columns of $X$ are the principal components of the data set $\overline{X}$, arranged from 
left to right in order of decreasing variance in their respective directions, and $\Lambda_X$ is the 
diagonal matrix of singular values (eigenvalues). Note that one can also find the singular 
value decomposition of the correlation matrix of $\overline{X}$ as an alternative to the covariance 
matrix. To determine the rotation dissimilarity between the two data sets $\overline{X}$ and $\overline{Y}$, we 
measure the angles between their principal components.

Since the columns of $X$ and $Y$ are unit vectors, it follows that the diagonal of the 
matrix $X^T Y$ is the cosine of the angles between the corresponding principal components, 
and so our rotation dissimilarity measure $D_r$ is defined as the sum of the angles between 
the components:

$$D_r(\overline{X}, \overline{Y}) = \text{trace}(\cos^{-1}(\text{abs}(X^T Y))).$$

(7.3)

Since the signs of the principal components do not matter, taking the absolute value ensures 
that we will only be concerned with acute angles. It can be easily shown that if $\overline{X}$ and $\overline{Y}$ 
are $n$-dimensional data sets, then $D_r(\overline{X}, \overline{Y})$ only takes on values in the set $[0, \frac{n\pi}{2}]$, where 
a value of 0 infers that the principal components are exactly aligned according to the size 
of their corresponding eigenvalues, a value of $\frac{n\pi}{2}$ infers that the principal components are 
completely orthogonal.

Note that the rotation dissimilarity measure $D_r$ also accounts for some aspects of the 
differences in the covariance structures of $\overline{X}$ and $\overline{Y}$, since it measures the amount of rotation 
needed so that their respective principal components are aligned in order of decreasing variance. However, we still must account for the amount of variance in each direction, or 
the “shape” of the data sets.
Variance Component

We note that data sets can have different “shapes.” For example, in two dimensions, a data set with little or no correlation between its attributes has a scatter plot that is circular in shape, while the points of a data set with maximum correlation all lie along the same line. It may be the case that the principal components of $\mathbf{X}$ and $\mathbf{Y}$, are completely aligned, but they still have very different shapes. For example, consider data sets C and E in Figure 7.1. It will be shown in Section 7.2.2 that the principal components of C and E are nearly aligned, but it is obvious to see that they have different variance structures by looking at the shapes of their plots: data set C has a short ovular shape, while E is much more elongated.

To account for these differences in the shapes of the data sets, we examine the difference in the distributions of the variance over the principal components of $\mathbf{X}$ and $\mathbf{Y}$. More formally, consider the random variable $V_X$ having the probability mass function:

$$P(V_X = i) = \frac{\lambda^X_i}{\text{trace}(\Lambda_X)}$$

(7.4)

where $\Lambda_X$ is the diagonal matrix of singular values from Equation 7.2, and $\lambda^X_i$ is the $i$th singular value. $P(V_X = i)$ is then the proportion of the variance in the direction of the $i$th principal component. We can then compare the distributions of $V_X$ and $V_Y$ by finding the symmetric relative entropy:

$$SRE(V_X, V_Y) = \frac{1}{2}(H(V_X \| V_Y) + H(V_Y \| V_X))$$

(7.5)

where $H(X \| Y)$ is the relative entropy of two random variables $X$ and $Y$. The relative entropy is a common measure of the distance between two probability distributions [36]. We can then define the variance dissimilarity as the symmetric relative entropy:

$$D_v(\mathbf{X}, \mathbf{Y}) = SRE(V_X, V_Y).$$

(7.6)
Final Dissimilarity Metric

The dissimilarity between $\overline{X}$ and $\overline{Y}$ can now be defined in two different manners. Our basic formulation is given by:

$$D_{\Pi}(\overline{X}, \overline{Y}) = D_d \times D_r \times D_v. \quad (7.7)$$

A more flexible formulation is as a linear combination of the components, given by:

$$D_{\Sigma}(\overline{X}, \overline{Y}) = \beta_0 + \beta_d \times D_d + \beta_r \times D_r + \beta_v \times D_v. \quad (7.8)$$

This formulation allows the components to be weighted differently (or completely ignored) by means of varying the values of their coefficients (i.e. $\beta$). To avoid an unwanted bias towards one or more of the components, the coefficients must be chosen to normalize their respective components. This is straightforward for some components (for example $D_r$ only takes on values in the range $[0, \frac{n\pi}{2}]$), but not for others (for example, when using the Euclidean distance for $D_d$ on non-normalized data).

Since the coefficients allow the components to be weighted differently, a user can bias the measure to reflect domain knowledge. For example, $D_{\Sigma}$ reduces to the basic Euclidean distance between the centroids of the data sets when $\beta_d$ is set to 1 and the others are set to 0. However, on the other extreme, one may be more concerned with finding data sets with similar covariance structures, but may not be concerned with relative locations of the data sets, and so $\beta_r$ and $\beta_v$ can be set to some positive value, while $\beta_d$ is set to 0.

Missing Data

Our measure is also robust to missing data. If a data set $\overline{X}$ has records with missing attribute values, and assuming that the data has a normal distribution, one can use the Expectation-Maximization [39] algorithm to find the maximum-likelihood values of the
centroid $\mu_x$ and the covariance matrix $\text{cov}(X)$. The principal components one finds are the sample principal components [78], and one can develop confidence intervals to test the closeness to the true (population) principal components. If the missing data is not excessive, then the maximum likelihood/sample estimates of the components will be accurate, and the computation of the dissimilarity metric can continue as before. Other approaches for handling sparsely missing data involve just ignoring records with missing data completely. In Section 7.3.5 we present results that show simply ignoring missing data does not drastically affect the performance of our measure.

### 7.2.2 Example

In this example, we will look at each component in turn to show it influences the final value of the dissimilarity measure. Consider Figure 7.1, where we have plotted five different data sets labeled A through E. Each data set is similar to the others in different ways. For example, sets A and E have similar shapes, D and E have similar centroids, and B and D have similar slopes.

![Synthetic Data Sets](image)

Figure 7.1: A plot of five synthetic data sets.
In Table 7.1 we present the pairwise distance dissimilarities of the data sets. The bold face values represent the minimal dissimilarities between data sets. As we expect, data sets A and C are considerably similar to each other according to this measure, as are data sets D and E, while data set B is considerably dissimilar from all the other data sets. Note that while data sets A and C have similar means, they have extremely different covariance structures that are not taken into account by this measure.

In Table 7.2 we present the pairwise rotation dissimilarities of the data sets. As we expect, data sets A, B, and D are very similar to each other, since their principal components are pointed in nearly the same directions. We note that the most similar pair of data sets

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0.67</td>
<td>2.53</td>
<td>0.4</td>
<td>2.55</td>
</tr>
<tr>
<td>B</td>
<td>0.67</td>
<td>0</td>
<td>3.09</td>
<td>0.27</td>
<td>3.06</td>
</tr>
<tr>
<td>C</td>
<td>2.53</td>
<td>3.09</td>
<td>0</td>
<td>2.93</td>
<td>0.02</td>
</tr>
<tr>
<td>D</td>
<td>0.4</td>
<td>0.27</td>
<td>2.93</td>
<td>0</td>
<td>2.95</td>
</tr>
<tr>
<td>E</td>
<td>2.55</td>
<td>3.06</td>
<td>0.02</td>
<td>2.95</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.1: Dissimilarity: distance component.

Table 7.2: Rotation dissimilarity.
Table 7.3: Variance dissimilarity.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0.18</td>
<td>0.20</td>
<td>0.10</td>
<td>0.000009</td>
</tr>
<tr>
<td>B</td>
<td>0.18</td>
<td>0</td>
<td>0.0007</td>
<td>0.01</td>
<td>0.18</td>
</tr>
<tr>
<td>C</td>
<td>0.20</td>
<td>0.0007</td>
<td>0</td>
<td>0.01</td>
<td>0.21</td>
</tr>
<tr>
<td>D</td>
<td>0.096</td>
<td>0.009</td>
<td>0.014</td>
<td>0</td>
<td>0.099</td>
</tr>
<tr>
<td>E</td>
<td>0.000009</td>
<td>0.18</td>
<td>0.21</td>
<td>0.10</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.4: Total dissimilarity ($D_{Π}$).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>60.07</td>
<td>2.73</td>
<td>32.94</td>
<td>0.02</td>
</tr>
<tr>
<td>B</td>
<td>60.07</td>
<td>0</td>
<td>1.14</td>
<td>1.38</td>
<td>341.14</td>
</tr>
<tr>
<td>C</td>
<td>2.73</td>
<td>1.14</td>
<td>0</td>
<td>36.07</td>
<td>4.52</td>
</tr>
<tr>
<td>D</td>
<td>32.94</td>
<td>1.38</td>
<td>36.07</td>
<td>0</td>
<td>3.99</td>
</tr>
<tr>
<td>E</td>
<td>0.02</td>
<td>341.14</td>
<td>4.52</td>
<td>3.99</td>
<td>0</td>
</tr>
</tbody>
</table>

according to this measure is E and C, while according to the distance dissimilarity measure (see Table 7.1) they are the most dissimilar pair of data sets.

In Table 7.3, we present the pairwise variance dissimilarities. In this case, data sets A and E are very similar to each other, which is expected, since the plots of each are both long and thin. We also note that while data sets B and C are very similar to each other according to the variance dissimilarity measure, they are also the most dissimilar pair according to the rotation dissimilarity measure (see Table 7.2).

In Table 7.4, we present the total pairwise dissimilarity of the data sets. In this case we use the product form ($D_{Π}$) of our measure. We find that data sets A and E are the most similar, due to the high similarity of the distribution of their variances across their
principal components. Data set E is next most similar to data set D due to the proximity of their means, and E is also quite similar to data set C, since their principal components are rotated similarly. E is most dissimilar to data set B due to large differences in their respective means, rotations, and variances. However, a basic distance-based dissimilarity measure (for example, using just $D_d$) would rank B as the second-most similar data set to E (after D), as can be seen from Table 7.1.

7.2.3 Applications

In this section we present an overview of how our dissimilarity measure can be used in several common data analysis techniques. The techniques we consider are change point detection, anomaly detection, and data set clustering.

Change Point Detection

One application of our dissimilarity measure is change point detection. In change point detection, one wants to find the point(s) in a time series where there has been an abrupt change in the process generating the series [15]. Our algorithm for off-line change point detection

```plaintext
procedure FindChangePoints(series T, int W)
begin
  for each point $t \in T$
    $Before = \{\text{the } W \text{ points occurring before } t\}$
    $After = \{t\} \cup \{\text{the } W - 1 \text{ points occurring after } t\}$
    $Score[t] = D(Before, After)$
  end
  Filter $Score$ to find maxima
  Return the $t$ corresponding to maxima of $Score$
end.
```

Figure 7.2: The change point detection algorithm.
detection for multivariate time series is presented in Figure 7.2. It works by scanning over a
time series \( T \), comparing two successive windows of \( W \) data points using our dissimilarity
measure \( D \). It returns the maxima of \( D \) applied over \( T \). It follows that the maximum
value of \( D \) is achieved when the two successive windows are most different with respect
to their means, rotations, or variances, signaling that the underlying distribution generating
the time series has changed between the two windows. We present the experimental results
of running change point detection on stock market data in Section 7.3.2.

**Anomaly Detection**

A closely related problem to change point detection is anomaly detection. Whereas
change point detection seeks to discover points that mark a shift from one generating pro-
cess to another, anomaly detection seeks to discover points that are outliers with regard to
the current generating process. Outlier detection algorithms work by assigning an anomaly
score to each point in a data set based on its dissimilarity to the other points in the set.
The most dissimilar ones are marked as outliers. Since our measure is designed to measure
the dissimilarity between a pair of data sets, we cannot directly measure the dissimilarity
between a point and a data set. However, we can use our measure to assign an anomaly
score to a point:

\[
S_{\overline{X}}(x) = D(\overline{X}, \overline{X} - x). \tag{7.9}
\]

The anomaly score function \( S_{\overline{X}}(x) \) measures how much the mean and covariance structure
of \( \overline{X} \) would change if the data point \( x \) was removed. If the value of \( S_{\overline{X}}(x) \) is large, then \( x \)
must be introducing considerable distortion into the model of \( \overline{X} \).

We demonstrate the utility of our dissimilarity measures for outlier detection using the
above approach with a toy data set. We compare our measures to the Mahalanobis distance
Figure 7.3: Outliers in a data set discovered using different measures: (A) Mahalanobis; (B) $D_\Pi$; (C) $D_\Sigma$; (D) $D_\Sigma$ distance component only; (E) $D_\Sigma$ rotation component only; (F) $D_\Sigma$ variance component only.
metric, since it also incorporates information concerning the covariance structure of the data set (similar to the formulation in Equation 7.9, we calculate the distance from a point \( x \) to the centroid of \( \mathbf{X} - x \) using the covariance matrix of \( \mathbf{X} - x \)). Our data set contains 150 points, and we find the top 15 outliers according to each measure. The results can be seen in Figure 7.3. In these plots, normal points are denoted by pluses and outliers are denoted by stars. In Figure 7.3(A) we show the outliers discovered using the Mahalanobis distance metric. In Figures 7.3(B) and (C) we show the outliers discovered using our \( D_\Pi \) and \( D_\Sigma \) measures respectively (in the case of \( D_\Sigma \), we have chosen the \( \beta \)'s so that the components are normalized). As we expect, the results are quite similar, since they all take into account both the means and the covariances of the data. However, unlike the Mahalanobis distance metric and \( D_\Pi \) measure, the \( D_\Sigma \) measure is much more flexible, as the user is able to chose the values of the \( \beta \)'s. This flexibility is demonstrated in Figures 7.3(D)-(F), where we detect outliers using only the distance, rotation, and variance components respectively by setting the coefficient of the relative component to 1 and the others to 0. In each case, different outliers are found. For example, using the distance component only (Figure 7.3(D)), the outliers are those points on the extreme ends of the “arms” of the data set, whereas when we use the rotation component only (Figure 7.3(E)), the points not belonging to any of the “arms” are marked as outliers.

One can also use an alternative incremental form of anomaly detection that is applicable in domains where data sets are streaming or in the form of time series. In this form, one calculates \( D(\overline{\mathbf{X}}, \overline{\mathbf{X}} \cup \{x\}) \), where \( \overline{\mathbf{X}} \) is a sliding window of \( k \) data points, and \( x \) is the first data point following the window. This is similar to change point detection, except that it only concerns information that arrives prior to \( x \). We present experimental results of using this approach with our dissimilarity measure on stock market data in Section 7.3.3.
Data Set Clustering

One of the advantages of a dissimilarity measure for data sets is that it allows one to cluster the data sets into groups with similar means or variances, depending on how one weights the components. As a motivating example, consider a large business organization such as Wal-Mart, with national or international interests. Such organizations usually rely on a homogeneous distributed databases to store their transaction data. This leads to time varying, distributed data sources. In order to analyze such a collection of databases, it seems important to cluster them into small number of groups to contrast global trends with local trends so as to develop advertising campaigns targeted at specific clusters.

It is straightforward to perform agglomerative hierarchical clustering of data sets using our dissimilarity measure. If one has \( n \) data sets, one can construct an \( n \times n \) table containing the pairwise dissimilarities of the data sets. Once this table has been constructed, one can use any distance metric (e.g. single-link or complete-link) to perform the hierarchical clustering. We present experimental results on using hierarchical clustering for stock market data in Section 7.3.4. This table also facilitates non-hierarchical clustering approaches, such as the k-medoid approach [65]. This works by selecting several data sets at random to be medoids of the clusters, and then assigning the remaining data sets to a cluster with the most similar medoid. After this phase, the medoids are checked to see if replacing any of them with other data sets would reduce the dissimilarity in their respective clusters. If so, the process repeats until no medoids are replaced, or some other stopping criterion is met.
7.3 Experimental Results

7.3.1 Setup

In our experiments we utilize historical stock market data available from Yahoo! Finance [1]. We constructed several multivariate time series data sets, where each dimension is the adjusted closing price of a stock or stock index. The stock indices that we use are the Dow Jones (DJ), Standard and Poor’s 500 (S&P 500), and the 10-year Treasury Note (TN) indices from January 1962 until May 2005. We also used the stock prices of a set of six pharmaceutical companies (Abbott Laboratories, GlaxoSmithKline, Johnson and Johnson (J & J), Merck, Pfizer, and Wyeth) from August 1986 until June 2005. All of our implementations are done using Octave, an open-source version of Matlab.

7.3.2 Change Point Detection

In our first set of experiments, we examined our measure’s effectiveness when used for change point detection. One of our more impressive results comes from a bivariate data set containing the values of the Dow Jones and S&P 500 indices. In this experiment we normalized the data and set the window size $W$ equal to 100. We derived the principal components from the covariance matrix of the data. We eliminated the scores of the first and last months in order to avoid edge effects. The highest-scoring change point according to our dissimilarity metric ($D_{11}$) occurred on February 28, 2001. In Figure 7.4 we plot these two indices versus time, showing 100 points on either side of the change point. From this figure we can see that the indices become more highly correlated after the change point. The difference is more obvious in Figure 7.5. Here the values of the indices are plotted against each other, and the markers indicate whether the point comes from before or after the change point. As can be seen, the points fall into two distinct clusters depending on
whether they come before or after the change point. We note that when we perform SVD using the correlation matrices instead of covariance matrices of the data, the results are very similar, though the change points may be shifted by a few instances. For the example above, when we use the correlation matrix, we calculate the change point as February 23, 2001.

![Figure 7.4: Plot of stock indices centered on change point.](image)

**7.3.3 Anomaly Detection**

We test our incremental outlier detection algorithm on several data sets. The *Indices* data set contains all three stock indices (DJ, S&P 500, and TN). The *DJ/S&P 500* and *DJ/TN* data sets contain the two relevant indices only. The *Pharm.* data set contains all six pharmaceutical stocks (see Section 7.3.1), while the *Pfizer/Merck* data set contains only the two relevant indices. In our experiment we use $D_{II}$, performing SVD on the covariance matrices. Also, we vary $k$ (the size of the sliding window) over 12 different values (4, 5, 6,
8, 10, 15, 20, 30, 40, 60, 80, 100). We mark the dates of the top 30 outliers for each value of $k$, creating 12 lists of 30 dates each.

In Figure 7.6 we plot Pfizer’s and Merck’s stock prices during the year 2004. The vertical lines mark the days of the outliers. 2004 was tumultuous year for these stocks, as it contains six of the top 30 outliers according to our measure when we set $k$ equal to 15. Note that our measure is able to detect large changes in the means, as is the case for the September 30 and December 17 outliers, as well as more subtle outliers, such as the one occurring on March 11. In Figure 7.7 we show a scatter plot of Pfizer and Merck stock prices for March 11 and the 15 trading days preceding it. This clarifies why March 11 is marked as an outlier: In the 15 trading days prior to March 11, the Pfizer’s and Merck’s stock prices were relatively uncorrelated, but on March 11, the prices of both sank.

We also verify that our measure can detect known anomalies. To do this we search the 12 lists of outliers for several well-known dates. For example, we pick October 19, 1987, since the stock market suffered a major crash on that day, and March 16, 2000, as
Figure 7.6: Outliers in 2004 Merck and Pfizer stock prices.

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
<th>Indices</th>
<th>DJ/S&amp;P 200</th>
<th>DJ/TN</th>
<th>Pharm.</th>
<th>Pfizer/Merck</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-19-87</td>
<td>Market crash</td>
<td>92% (100%)</td>
<td>25% (58%)</td>
<td>92% (92%)</td>
<td>17% (75%)</td>
<td>25% (83%)</td>
</tr>
<tr>
<td>3-16-00</td>
<td>Largest DJ increase</td>
<td>0% (0%)</td>
<td>50% (0%)</td>
<td>8% (0%)</td>
<td>17% (0%)</td>
<td>0% (0%)</td>
</tr>
<tr>
<td>4-14-00</td>
<td>Largest DJ decrease</td>
<td>33% (8%)</td>
<td>58% (25%)</td>
<td>50% (17%)</td>
<td>0% (0%)</td>
<td>0% (0%)</td>
</tr>
<tr>
<td>9-17-01</td>
<td>WTC attack</td>
<td>8% (58%)</td>
<td>25% (67%)</td>
<td>42% (33%)</td>
<td>0% (0%)</td>
<td>0% (0%)</td>
</tr>
<tr>
<td>9-30-04</td>
<td>Vioxx™ warning</td>
<td>0% (0%)</td>
<td>0% (0%)</td>
<td>0% (0%)</td>
<td>58% (100%)</td>
<td>92% (100%)</td>
</tr>
<tr>
<td>12-17-04</td>
<td>Celebrex™ warning</td>
<td>0% (0%)</td>
<td>0% (0%)</td>
<td>0% (0%)</td>
<td>8% (0%)</td>
<td>75% (33%)</td>
</tr>
</tbody>
</table>

Table 7.5: Detection rates of notable outliers using the $D_{\Pi}$ measure and a Mahalanobis metric-based $D_{\Sigma}$ measure (in parenthesis).

that day currently holds the record for the largest increase of the Dow Jones index. We also pick September 30, 2004 and December 17, 2004, as those are the days when information was announced concerning serious side-effects of Merck’s Vioxx™ and Pfizer’s Celebrex™ drugs, respectively\(^5\). We compare our basic $D_{\Pi}$ measure against the $D_{\Sigma}$ measure that uses the Mahalanobis metric for the distance component.

\(^5\)Vioxx is a trademark of Merck and Company, Incorporated. Celebrex is a trademark of Pfizer, Incorporated.
In Table 7.5 we present our results as the percentage of the lists in which each date appeared for all of the data sets for both the $D_{II}$ measure and the Mahalanobis-based $D_{Σ}$ (which is given in parenthesis). Our measure discerns these anomalous days fairly well. The first four rows indicate anomalous days for the overall stock market, and the anomalies are reflected in the market index data sets. The last two rows indicate anomalous days for the Pharmaceutical sector, as the announcement concerning Vioxx\textsuperscript{TM} and Celebrex\textsuperscript{TM} had adverse effects on the price of Merck’s and Pfizer’s stocks, respectively. However, the effect of the announcements was not so great on the values of the market indices.

We note that in some cases, the Mahalanobis metric-based $D_{Σ}$ approach out-performs the $D_{II}$ measure, but in other cases, the $D_{II}$ measure out-performs the Mahalanobis-based approach. For example, the Mahalanobis-based approach more consistently detects the October 19, 1987 market crash and the September 30, 2004 Vioxx\textsuperscript{TM} announcement, while our measure more consistently detects the largest Dow Jones increase (March 16, 2000) and decrease (April 14, 2000). The reason is manner in which the two approaches detect...
anomalies. The Mahalanobis-based approach is biased towards the Mahalanobis distance between stock prices on the current day and the mean of the prices on the previous $k$ days. Therefore, the Mahalanobis-based approach is good at detected large changes in mean. The $D_{11}$ measure, however, also detects changes in the correlations. The Dow Jones anomalies involve a difference in mean of the value of the Dow Jones index, but this is not as drastic as the change in correlation that results when it is paired with other indices that do not have changes as large as that of the Dow Jones index.

### 7.3.4 Data Set Clustering

In our next experiment, we examine the effects of using our dissimilarity measure to perform agglomerative hierarchical clustering. Note, as mentioned earlier, one can also use k-medoids clustering here. Currently, we use the Pfizer/Merck data set, and extract the records for each month during the year 2004 to form 12 separate data sets (see Figure 7.8). We build a table of pair-wise dissimilarities between the monthly Pfizer/Merck data sets using the $D_{12}$ measure, with a slight bias towards the distance component to account for the drop in stock prices in the latter part of the year. Using this table, we perform hierarchical clustering using the single-link distance metric. The dendrogram resulting from the clustering can be seen in Figure 7.9. The results are expected: the data sets for January through June are clustered early, as they have similar means and positive correlations, and the data sets for October through December are not clustered together until very near the end, due to their large differences in means compared to the other months. We see that October and December cluster with each other first, which is notable since they are months most influenced by the Vioxx™ and Celebrex™ announcements, respectively.
Figure 7.8: Pfizer/Merck 2004 stock prices.

Figure 7.9: Dendrogram resulting from clustering of monthly data sets.
7.3.5 Missing Data

Finally, we examine the effects of ignoring records containing missing data. In this experiment we used the DJ/S&P 500 data set, and progressively removed 1%, 5%, 10%, 15%, and 20% of the records (i.e. that data set with 20% of the records removed is a subset of the data set with 15% of the records removed, and so on). For each of these data sets, we calculated the top 20 change points using the algorithm in Section 7.2.3, with \( W \) equal to 15, 40, and 100. We then compared each set of 20 change points to the top 20 change points found when there is no missing data. We counted the number of change points that matched to within some time window (on the order of one week for \( W \) equal to 15, and one month for \( W \) equal to 100). The percentage of correct matches for each data set and each value of \( W \) is presented in Figure 7.10. Our detection rates from a high of 100% (all change points found) for 1% missing data to a low of 70% when 15% to 20% of the data is missing. Note that in this experiment we assume we do not know which records are missing—we calculate the change point based on the \( W \) non-missing records coming before the point, and the \( W \) non-missing records coming after it. Therefore, the change point is calculated using only a subset of the records used if there was no missing data, plus some “extra” records would not be considered if there were no missing data. This fact, coupled with our detection rates, suggests that our approach is robust to both missing data, and the noise added by the “extra” records.

7.4 Conclusion

In this chapter we presented a dissimilarity measure for data sets that takes into account the means and correlation structures of the data sets. This dissimilarity measure is tunable, allowing the user to adjust its parameters based on domain knowledge. The measure
Figure 7.10: Percentage of change points found for differing degrees of missing data.

has many different applications, including change point detection, anomaly detection, and clustering, and our experimental results show its effectiveness in these areas. In future we want to use our dissimilarity measure to detect anomalous data sets. This is applicable to clinical trial data, where patients are represented by a multivariate time series of blood analyte values, and detection of anomalous patients can lead to early discovery of possibly serious side effects of the drug being tested. We also plan to explore the incremental aspects of our measure in order to apply to dynamic and streaming data sets. For example, the computational costs of calculating our dissimilarity measure on dynamic or streaming can be reduced by using incremental PCA techniques [9, 63].
CHAPTER 8

DISSIMILARITY MEASURES FOR DETECTING HEPATOTOXICITY IN CLINICAL TRIAL DATA

8.1 Introduction

Drug safety issues have received much attention in 2004. Several large pharmaceutical companies have issued warnings or removed their drugs from the market following reports of severe or deadly side-effects. Such events are harmful to the companies’ public images, and their financial status. Each company invests large amounts of money in developing and testing new drugs. Any drug under development or in clinical trials that does not make it to the market represents a huge loss for the company. Also, any drug that makes it to market but must be withdrawn represents a double loss for the company, as it is unable to recoup development costs, and may be held liable for any harmful effects of that drug. Therefore, pharmaceutical companies have an intense interest in discovering any harmful effects of their drugs as early as possible, so they can cease development or sales in order to save both lives and money.

The safety and efficacy of new drugs are determined using a set of clinical trials. Clinical trials occur in four phases. The ability to identify harmful drugs and cease development at least one phase earlier than usual can save a pharmaceutical company billions of dollars.
In pharmaceutical clinical trials, the efficacy and safety of a drug for treating a particular disease is studied by comparing the results from some healthy subjects and many patients who are randomly assigned to either the experimental drug, existing therapies for the disease, or a placebo. Safety is studied in many ways; serial clinical laboratory blood tests are used commonly to monitor biochemical changes in the body. A common reason for stopping a drug development project or causing discontinuation in a particular patient or group of patients are abnormal blood test values related to the liver, as it has a major detoxifying function. When liver tests are high, it is assumed that hepatotoxicity, or liver toxicity, is present. However, the rules for determining the presence of drug-induced hepatotoxicity are mostly qualitative and involve considerable clinical judgment. The current state-of-the-art in pharmaceutical research uses ad-hoc univariate rules applied to multiple analytes. More recently a rule known as “Hy’s Rule” [18] has been employed which requires the crossing of at least two thresholds. The problem of misclassification should be obvious, since hepatotoxicity may not be so much correlated with absolute elevated blood analyte values as it is with how the analytes move together. Our hypothesis is that Hy’s rule is not sufficient, and that correlations between analytes are extremely important for understanding the effects of a drug on liver toxicity.

Clinical trial data is usually in the form of a set of multivariate time series, where each variable corresponds to a blood analyte and each series corresponds to a different patient. In this chapter we continue our examination the notion of quantifying the dissimilarity between different sets of data that we started in Chapter 7. However, in this chapter we have the goal of detecting hepatotoxicity, and we propose additional dissimilarity measures

6The work presented in this chapter has been published previously in the SIAM International Conference on Data Mining [112] and was jointly conducted with Donald C. Trost of Pfizer, Inc. This work was sponsored by Pfizer as part of a collaborative project in pathodynamics methodology.
that can be used to quantify the differences between two data sets. Other applications of our measure for clinical trial data involve characterizing the differences between the different subsets of patients and discovering subpopulations that have a greater risk of hepatotoxicity. The measures we propose are based on principal component analysis (PCA). Our measures consists of components that separately take into account differences in the locations, and correlations of the data sets being compared. It is also possible to weight the components differently, so one can incorporate domain knowledge into the measure. Finally, our measure is robust towards noise and missing data. We demonstrate the efficacy of the proposed measures using clinical trial data provided by Pfizer that is known to contain subjects exhibiting hepatic changes.

8.2 Algorithms

As we discussed in Section 8.1, clinical trial data are presented in the form of a multivariate time series for each subject in the trial. At each time point, the values of various blood analytes are recorded. While there are many techniques for analyzing (multiple) times series data [10, 33, 43], clinical trial time series data is quite challenging. Clinical trial time series data sets suffer from irregular sampling, missing data, and varying lengths. This may be due to a variety of reasons, including missed appointments, unexplained absences, and drop outs. Furthermore, there are also several potential sources of noise. Measurement errors, laboratory bias, and circadian effects on analyte values (depending on when the blood sample was drawn) can be contributing factors to noise.

The basis of Hy’s rule, and the typical signal physicians look for when evaluating liver toxicity, is usually a significant and consistent departure from the normal levels of one or

7Different laboratories, where these tests are often analyzed, often have different protocols resulting in a significant variation in analyte values for the same subject.
more liver analytes. Moreover, it is usually the case that not all the analytes are affected simultaneously. A conclusion one can draw from these two statements is that the correlation among analytes should be capable of identifying such significant departures from the norm.

This key intuition leads us to use correlation or covariance matrices to represent patient data. We subsequently use principal component analysis-based methods for computing dissimilarity measures for such datasets. We note that correlation and covariance matrices can easily be imputed in the presence of missing data by using the Expectation-Maximization algorithm [39] to find the maximum-likelihood values of the covariance or correlation matrices. Moreover, principal components based techniques have been shown in the literature to be noise-tolerant [122].

8.2.1 Dissimilarity Measures

Our goal is to quantify the dissimilarity of two \( k \)-dimensional data sets \( \mathbf{X} \) and \( \mathbf{Y} \). Our measures take into account the correlations between the attributes of the two data sets. In general, the dissimilarity of two data sets \( \mathbf{X} \) and \( \mathbf{Y} \) is denoted as \( D(\mathbf{X}, \mathbf{Y}) \). We define the function \( D \) in terms of two dissimilarity functions that take into account the differences the magnitude and direction of the variance in the data sets. These components are combined by means of a weighted sum, which allows one to weight the components differently, so as to incorporate domain knowledge.

The first step in using our dissimilarity measures is to find the principal components of the data sets being compared. The principal components of a data set are the set of orthogonal vectors such that the first vector points in the direction of greatest variance in the data, the second points in the orthogonal direction of the second greatest variance in the data, and so on [129]. We consider \( \mathbf{X} \) and \( \mathbf{Y} \) to be most similar to each other when
their principal components, paired according to their ranks, are aligned and have the same magnitude, and most dissimilar when all of the components of $\mathbf{X}$ are orthogonal to those of $\mathbf{Y}$.

More formally, given a data set $\mathbf{X}$, consider the singular value decomposition (SVD) of its covariance matrix:

$$cov(\mathbf{X}) = U \Lambda_X X^T$$

(8.1)

where the columns of $X$ are the principal components of the data set $\mathbf{X}$, arranged from left to right in order of decreasing variance in their respective directions, and $\Lambda_X$ is the diagonal matrix of singular values. Note that one can also find the SVD of the correlation matrix of $\mathbf{X}$ as an alternative to the covariance matrix.

Having found the principal components, we can now represent each data set $\mathbf{X}$ as a single feature vector $F_X$:

$$F_X = \sqrt{\Lambda_1} \times X_1$$

(8.2)

where $X_1$ is the first principal component of the data set, or the first column of $X$ in Equation 8.1, and $\Lambda_1$ is its corresponding eigenvalue. That is to say each data set is represented by the scaled primary principal component vector pointing in the direction of greatest variance.

Having such a feature vector, we can then apply any standard distance metric. For example, applying the Euclidean distance metric:

$$D_e(F_X, F_Y) = |F_X - F_Y|^2$$

(8.3)

on the first principal component derived from the covariance matrix of the data would result in a value that simultaneously measures differences in direction and magnitude of the vector.
These two measures can be extended to account for the differences in the mean of the data sets. First we define the dissimilarity of the means of the data sets as follows:

\[ D_\mu(\mathbf{X}, \mathbf{Y}) = |\mu_X - \mu_Y|^2. \]  
(8.4)

that is to say, the Euclidean distance between the centroids of the two data sets. We can then define the extended \( D_e \) measure as follows:

\[ D_e(\mathbf{X}, \mathbf{Y}) = \beta_0 + \beta_1 \times D_\mu + \beta_2 \times D_e. \]  
(8.5)

This formulation allows us to weight differences in the means and correlations according to domain information. For example, in clinical trial data, differences in the means of the observations of two different subjects may be caused more by differences in demographic characteristics (e.g. sex, age, weight) than by any effect of the drug, and so one would want to weight the differences in correlations higher.

Finally, we note that we can generalize these measures to account for all the principal components as follows. Let \( F^i_X \) be the feature vector for the \( i \)th component:

\[ F^i_X = \sqrt{\Lambda_i} \times X_i. \]  
(8.6)

Then the \( D_e \) measure can be generalized as:

\[ D'_e(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{k} D_e(F^i_X, F^i_Y). \]  
(8.7)

### 8.2.2 Applications

In this section we present an overview of how our dissimilarity measures can be used to analyze the clinical trial data. The techniques we consider are anomaly (outlier) detection, and data set clustering.
Anomaly Detection

Detection of anomalies or outliers in clinical trial data is very important. Subjects’ analyte values may be anomalous for many reasons related to sample processing including subject ingestion of interfering substances, sampling handling conditions, analyzer error, and transcription error. If these data points can be identified and the cause attributed to a non-treatment-related event, then the data point may need to be removed from a particular analysis. Subjects’ values may be anomalous because they are having abnormal reactions to the drug. If this is the case, the drug maker may want to study more subjects similar to the anomalous ones to see if they are true anomalies or indicate a small sub-populations that may have toxic reactions to the drug. Using our dissimilarity measures, it is straightforward to implement basic outlier detection algorithms such as those described in [82]. These are nested-loop approaches that calculate the dissimilarity between each pair of data points (or in our case, each pair of subjects). Having calculated these values one can rank the data points (subjects) according to the sum of the dissimilarities from the $k$ most similar subjects.

Clustering

The dissimilarity measures we present above allow us to easily perform clustering of the subjects. Finding clusters of subjects in clinical trial data is helpful in that it allows us to identify sub-populations who may have a greater risk of hepatotoxicity, sub-populations on whom the drug may have little or no effect, sub-populations that may have a higher risk of severe side-effects, et cetera. This allows the drug makers to determine the efficacy of the drug, to determine dosage levels for different patients, and to determine if the side-effects are too severe or widespread to continue development of the drug. It is straightforward to
perform agglomerative hierarchical clustering of data sets using our dissimilarity measures. If one has \( n \) data sets, one can construct an \( n \) by \( n \) table containing the pairwise dissimilarities of the data sets. Once this table has been constructed, one can use any distance metric (e.g. single-link) to perform the hierarchical clustering.

### 8.3 Experimental Results

#### 8.3.1 Setup

The first dataset we use, henceforth referred to as \( D_1 \), consists of a set of subjects suffering from diabetes, who, in addition to their regular diabetes therapy, were receiving either a placebo (a formulation that includes only the inactive ingredients) or the study drug (drug A) for a diabetic complication. Since we are primarily concerned with hepatotoxicity, following suggestions from our domain experts we only consider eight serum analytes (often referred to in the literature as the liver panel): ALT (alanine aminotransferase), AST (aspartate aminotransferase), GGT (\( \gamma \)-glutamyltransferase), LD (lactate dehydrogenase), ALP (alkaline phosphatase), total bilirubin, total protein, and albumin. Using advice from a domain expert, we use the logarithm transformation of the first six analytes’ values (total protein and albumin are excepted). This dataset consists of 446 patients on placebo and 680 patients on drug. Development on this drug was discontinued in Phase III for various reasons including possible hepatotoxicity.

The second dataset we use, henceforth referred to as \( D_2 \), consists of a set of post-menopausal women, who again were given either a placebo or one of two drugs \( (B, C) \) (both are different from drug \( A \)). Again, we limit our focus to the liver panel. This dataset consists of 201 patients on placebo, 41 patients on drug \( B \), and 126 patients on drug \( C \). Both drugs \( B \) and \( C \) are on the market, and are expected to have little or no hepatotoxicity.
Both datasets suffer from the problems we mentioned earlier. They contain missing data, unequally spaced time series data for different patients, some patients had many readings over a period of time, others had much fewer etc. As noted earlier we transformed the data from each patient into a feature vector using the covariance matrix as described in Section 8.2.1. Since the differences in the mean are not significant in these data sets, we use the basic forms of the $D_e$ measure defined in equation 8.3 in these experiments. All of our implementations are done using Octave, an open-source version of Matlab.

8.3.2 Anomaly Detection

![Graphs showing anomaly detection results]

Figure 8.1: Top outlier rankings using (A) $D_e$ on Drug A in $D_1$; (B) $D_e$ on Drug B in $D_2$; (C) $D_e$ on Drug C in $D_2$.

In our first experiment, we want to see how our dissimilarity measures perform on the clinical trial data set of diabetic patients. As noted earlier we have two groups of patients: one on placebo, and the other on the drug under study. The experiment we conduct is to flag outliers from the dataset using the dissimilarity measures discussed in the previous section. Note that previous to drug intake the distributions of the two groups are nearly identical. If the people on drug tend to be flagged as outliers with a greater probability than expected,
then a reasonable conclusion is that there may be a hepatotoxic effect resulting from drug intake.

We rank the subjects according to the approach presented in Section 8.2.2. Once we have these outlier rankings for all the subjects in a given study, we can use them to determine not only which subjects are the most anomalous, but also to determine if the drug being studied has any appreciable effect. For example, if we examine a ranking of the subjects, we would expect the hepatotoxic patients to be highest-ranked, followed by the remaining subjects who were on the drug, and finally the patients who were given a placebo. However, a drug that has little or no effect on the liver tests is less likely to cause hepatotoxicity, and subjects on such a drug should not be very dissimilar from those on placebo, meaning that the ranking would be random. To examine the effects of the drug being studied, we use graphs such as those in Figure 8.1, where we plot the cumulative number of subjects on drug and on placebo given the outlier ranking using thick lines. The thin lines express the expected cumulative number of subjects on drug or placebo for a given ranking assuming the ranking is random.

In Figure 8.1 (A) we plot the outlier ranking arising from both the $D_e$ measure for the top 10% (113) of the outliers in $D_1$. We observe that the expected number of outliers from the drug group is exceeded by the actual number indicating a clear signal that the drug under question is causing a change in analyte behavior in the patients being flagged as outliers. We would like to note that Phase III continued for approximately two more years after these cases were completed. Had this signal been detected at that time, Pfizer might have been able to save on the resources it expended to continue Phase III.

In our second experiment we evaluate the performance of our method on the second dataset composed of healthy post-menopausal women. In Figure 8.1 (B) and (C), we plot
the top 10% of the outliers for both drugs using the $D_e$ measure. As expected, since these are healthy women taking either a placebo or drugs with no known hepatotoxic effects, the mixture of subjects on drug and placebo marked as outliers are near the expected levels.

Figure 8.2: Top Outlier rankings for Drug A as function of the number of components used in $D'_e$: (A) 2 components, (B) 3 components, (C) 4 components.

In our third experiment we examined what effect varying the number of principal components has on the outlier rankings. In this case, we varied the number of components used by $D'_e$ (see Equation 8.7) between 2 and 4 and applied it to the data sets for Drug A. The results can be seen in Figure 8.2. For reference, recall that Figure 8.1(A) shows the $D'_e$ measure with only 1 component. As can be seen from the graphs, when we move from 1 component to 2, there is little change. However, when we move to 3 components, we mark significantly more subjects on drug as outliers. This appears to be the optimum number of components in this case, for when we move to 4 components, the sensitivity decreases somewhat.

These experiments demonstrate an advantage of our approach over Hy’s rule. They show that we are capable on not only finding important differences in magnitude, but also in direction (correlation) that may be missed by Hy’s rule.
8.3.3 Data Set Clustering

In our final experiment we demonstrate the utility of using our dissimilarity measures to perform clustering. In this case we use a subset of the subjects corresponding to all males with diabetes who were taking the drug being studied, for a total of 450 subjects. We use the Euclidean dissimilarity measure $D_e$ and the covariance matrices and performing single-link hierarchical clustering. We find that clustering results in an intuitive grouping of the subjects.

One branch of the resulting cluster dendrogram corresponds to a cluster of subjects with relative low spikes in analyte values. These spikes may not be large enough to be considered a sign of hepatotoxicity according to Hy’s rule. Another branch of the full dendrogram corresponds to a cluster of subjects with very large spikes in analyte values, nearly an order of magnitude larger than those in the other cluster. Other branches show different behaviors that may not be indicative of hepatotoxicity, but may be related to the subjects’ demographic or other health attributes, which may aid in determining dosage levels.

8.4 Conclusion

Efficient and precise analysis of clinical trial data is very important to pharmaceutical companies, as it allows them to determine the efficacy and safety of a drug. Pharmaceutical companies want to halt development on unsafe and ineffective drugs as early as possible in order to save on development costs and to avoid unnecessary complications and severe side-effects that may lead to liability suits if the drug were to reach the market. Current approaches for detecting hepatotoxicity in clinical trial data sets have limited effectiveness,
since they typically ignore correlations between blood analytes. In this chapter we presented several dissimilarity measures for data sets that takes into account the means and covariance structures of the data sets. Our results on real clinical trial data show that our measures can be very helpful in detecting true hepatotoxicity and finding subpopulations of subjects who may have different reactions to the drug under study.
CHAPTER 9

CONCLUSIONS AND FUTURE WORK

9.1 Conclusions

Abnormality detection is an important problem in the field of data mining, as it allows one to automatically detect abnormal activity in data. Abnormality detection has important application areas, including fraud detection, network intrusion detection, and data cleaning. While there have been many approaches to abnormality detection presented over the years (as seen in Chapter 2), we have stated that they do not work well when certain common problems or constraints are taken into consideration. Some of these problems are: (a) there are (strict) limits on how much time can be spent by an algorithm to find anomalies; (b) the data set may be dynamic or contain a temporal component; (c) the data set may be distributed; (d) the data set may contain heterogeneous features (i.e. continuous and categorical features) and missing data. In this dissertation we have shown that abnormality detection techniques can be modified to operate under the constraints presented by real-world data and environments.

Specifically, we first looked at detecting network intrusions using the programmable processors available on modern network interface cards (NICs) in Chapter 3. Detecting network intrusions using NICs provides both an extra layer of security and an element of
reliability, as it can prevent the host operating system from being affected by the intrusion while leaving the host processor free to service other applications. The central difficulty with performing intrusion detection at the NIC level is the limited processing and memory resources of NIC. These factors drive us to utilize detection schemes that are light-weight yet effective, such as probability tables and frequent itemset-based systems. In Chapter 6 we presented another light-weight algorithm (RELOADED) that could also operate on NICs. We also note that such light-weight approaches can be used in other domains where there are computationally-constrained processors, such as sensor networks.

We next looked at abnormality detection in distributed and dynamic (and even streaming) data bases in Chapters 4 and 5. In Chapter 4 we looked at mining frequent itemsets from distributed and dynamic data bases. Since frequent itemsets represent items that co-occur frequently in transactional data, they can be used to model normality in a data set. In the distributed case, we can have both local and global models of normality, and by identifying high-contrast itemsets, we can identify anomalous itemsets whose support values vary widely across different sites, or discover sites that have a local model that is anomalous compared to the other local models. In Chapter 5 we looked at the problem of outlier detection in mixed-attribute data sets that could be both distributed and streaming. Our approach, called LOADED, also makes use of frequent itemsets to model dependencies between categorical attributes, and uses covariance matrices conditioned on the values of the itemsets to model the dependencies between continuous attributes and dependencies between continuous and categorical attributes. LOADED allows for detection of outliers in a single pass, or in two passes with greater accuracy. In the latter case, it achieves a detection accuracy greater than that of ORCA, a state-of-the-art distance-based outlier detection algorithm. We also presented a variation on LOADED called RELOADED in Chapter 6. Even
though RELOADED has slightly worse detection and false positive rates than LOADED, it has a greatly reduced space and time complexity compared to LOADED, making it a candidate for computationally-constrained environments such as NICs and sensor networks, as mentioned above.

Finally, we next looked at developing novel dissimilarity measures in Chapters 7 and 8. In many applications objects are represented by a vector of attributes, but there are cases where objects are represented by a set of vectors or a matrix of attributes. In these cases there are few dissimilarity measures to use in applications such as outlier detection and clustering. Therefore, in these two chapters, we examined quantifying the dissimilarity between two multivariate time series data sets, such as stock market data (as in Chapter 7) and pharmaceutical clinical trial data (as in Chapter 8). In particular, we wanted measures that can take into account not only differences in magnitude, but also differences in the correlation structure of the data sets. Such a dissimilarity measure can be used in distance-based style outlier detection algorithms, as well as in clustering algorithms. When used with such algorithms, outliers would be those data sets with anomalous correlation structures, and clusters would be made up with data sets with similar correlation structures. When these algorithms are applied, for example, to clinical trial data, they can help us to discover subpopulations who may be prone to liver toxicity or other side-effects of an experimental drug.

9.2 Future Work

There are many possibilities for expanding and extending the work presented in this dissertation, and the future promises to bring more complex and unforeseen constraints and domains. One domain in which to expand and extend our work is that of embedded
systems. We examined NIC-based intrusion detection in Chapter 3, but the work there can be extended to sensor networks and other systems with limited computational and communication resources. While the resources available on these embedded systems is increasing, they will always be more limited than those of a full-fledged computer system, and so the algorithms we must use in such systems must always take into account resource limitations that they do not have to consider in the general case. However, we can always use hybrid solutions. For example, in NIC-based intrusion detection, it is obvious that LOADED is too resource-hungry to run on a NIC, but RELOADED is lightweight enough to run there. However, LOADED has better true and false positive rates than RELOADED. A hybrid scheme can take the best of both worlds: RELOADED, running on the NIC can make a first pass and mark certain packets as suspicious, while LOADED, running on the host machine makes a pass over the packets marked by RELOADED, and makes the final determination as to whether a packet is indeed malicious.

It is also possible to allow these embedded systems to work together (if they have the ability to communicate, as is the case for NICs and sensor networks). Using the ideas of distributed global model construction that we explored for both ZIGZAG and LOADED in Chapters 4 and 5, respectively, these embedded systems could build a global model autonomously. Forming a global model from the local models built by the \( P(SrcIP \mid DstIP) \) anomalous client detectors in Section 3.2.1 of Chapter 3, for example, or the local models built by RELOADED running on a NIC should be relatively straightforward. Furthermore, if communication is (severely) limited or costly to perform, then the local systems may be able to use the “exchange-outliers” approach that LOADED uses in its distributed single-pass version.
A growing area of interest in data mining research is link-based analysis and graph mining. Researchers use these techniques to investigate such things as protein interaction networks [25] and social networks [55]. In Chapter 5 we touched upon the idea of links, saying in Section 5.1.1 that two data points were linked if they have an attribute-value pair in common, and that an outlier was a point that had few links, or only weak links with other points in a data set. Using the dissimilarity measures we discussed in Chapters 7 and 8, we can define a notion of links between complex data objects (for example, those represented by a multivariate time series): a link exists if the two objects have a minimum level of similarity. Representing a collection of multivariate time series (or any other collection of complex objects) as a graph will allow one to apply graph mining approaches for abnormality detection, or for data mining in general.

In all cases, the methods and results of the abnormality detection technique must be transparent to the end user. The end user, with his or her domain knowledge, must be able to select an algorithm that will return the results he or she desires, with the computational resources available. In short, abnormality detection techniques need to become more user-friendly, as we cannot expect all end-users to be experts in data mining or statistics. Providing easy use of these techniques to such users involves creating adaptive abnormality detection systems – systems that can automatically or with minimum guidance choose the correct algorithm and associated parameters to use to give the end user the desired results, while automatically adapting to the available computing resources. This requires not only analyses of user’s needs and evaluations of algorithms and systems, but the creation and consolidation of the theoretical aspects of abnormality detection in order to create a solid foundation on which everything else will be built.
BIBLIOGRAPHY


180


[122] Srinivasan Parthasarathy and C. C. Aggarwal. On the use of conceptual reconstruction for mining massively incomplete data sets. IEEE Transactions on Knowledge and Data Engineering.


183


