HYPOTHESIS MARGIN BASED WEIGHTING FOR FEATURE SELECTION USING BOOSTING: THEORY, ALGORITHMS AND APPLICATIONS

A Thesis Presented
by
Malak Alshawabkeh
to
The Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in the field of
Computer Engineering
Northeastern University
Boston, Massachusetts
April 2013
Feature selection (FS) is a preprocessing process aimed at identifying a small subset of highly predictive features out of a large set of raw input variables that are possibly irrelevant or redundant. It plays a fundamental role in the success of many learning tasks where high dimensionality arises as a big challenge. Many endeavors to cope with this problem have been attempted and various outstanding feature selection methods have been proposed. Recently, there has been a growing line of research in utilizing the concept of hypothesis margins to measure the quality of a set of features. However, most previous feature selection algorithms have been developed under the large hypothesis margin principles of the 1-NN algorithm, such as Simba. Little attention has been paid so far to exploiting the hypothesis margins of boosting to evaluate features. Boosting is well known to maximize the training examples’ hypothesis margins, in particular, the average margin that considers the whole margin distribution and thus include more information. In this thesis, we took an unusual approach for using boosting as an effective FS by utilizing the training examples’ mean margins. A weight criterion, termed Margin Fraction (MF), is assigned to each feature that contributes to the margin distribution combined in the final output produced by boosting. We argue that using the MF is more favorable for several reasons. First, boosting hypothesis margins have been used both for theoretical generalization bounds and as guidelines for algorithm design, and thus, a natural goal is to find learners (features) that achieve a maximum margin. Second, current boosting-based feature selection methods measure the relative importance of features based on the Confidence Ratio (CR) of the learned base hypothesis. However, while a feature may have a large CR, it will not contribute to a good overall margin unless its “conditional” margin is also large.

The thesis mainly consists of two parts. In part one, we establish a rigorous theoretical and mathematical basis for the proposed weighting and selection methodology, and we describe how to extend this methodology to handle the presence of imbalanced data; by defining a new weight metric, termed AUC Margin Fraction (AMF), that characterize the quality of a set of features based on the maximized Area Under ROC curve (AUC) margin it induces during the process of learning with boosting. Based on this we design two different embedded-based FS algorithms, the SBS-MF and the SBS-AMF. We then investigate the effectiveness of the proposed methods through extensive comparisons with other algorithms using real-world data.

In part two, we apply the proposed SBS-AMF method to design a real intrusion detection system (IDS) of virtual server environments utilizing only information available from the perspective of the
virtual machine monitor (VMM). VMM-based IDSs break the boundaries of current state-of-the-art
IDSs. They represent a new point in the IDS design space that trades a lack of program semantics
for greater malware resistance and ease of deployment. To test the effectiveness and robustness
of our proposed VMM IDS, we use different classes of servers, virtual appliances, and workloads,
as well as different classes of malwares. Our experimental results show that SBS-AMF achieves
significantly better detection performance on the data sets tested using the Local Outlier Factor
anomaly detection algorithm (LOF), and we obtained on average 96% detection rate and 5% false
alarm rate. These results indicate that sufficient information exists in features selected by SBS-AMF
to build real IDS that is not susceptible to the characteristics of the attack behavior, or to specific
workload.

Due to the growing popularity of Graphics Processing Units (GPUs) in general-purpose comput-
ing domains we applied this parallel computing approach to accelerate the LOF method, to enhance
the detection speed of the proposed VMM IDS, as near real-time performance is needed in order to
detect any malicious activity before the system becomes fully compromised. With the GPU-enabled
LOF CUDA implementation we achieved more than a 100X.
Acknowledgements

In the name of God, the Most Beneficent, the Most Merciful

At the end of this exciting and at times frustrating process, having finished this dissertation, I would like to take the opportunity to express my gratitude to all that have supported me through this long journey.

I would like to express my deepest gratitude and respect to my research advisor, Professor David Kaeli. I have always benefited from his expertise supervision, brilliant ideas, continuous enthusiasm, rigorous attitude to science and valuable advice and extensive knowledge. I would like to thank him for providing me with research grants and the great opportunity to have received field research training.

I’m also most grateful to Professor Jay A. Aslam, his constructive guidance, generous support and inspired suggestions on my work during the years. He has always had shrewd insight and given valuable detailed comments and suggestions which have greatly helped to improve the work.

I sincerely thank Professor Jennifer Dy, for her invaluable help and important suggestions and remarks throughout my research.

My family, to whom I dedicate this dissertation, my beloved husband Motasem and my precious children Reyad and Ryan, thank you for all you’ve sacrificed during this doctoral journey: time with me, home-cooked meals, a clean house, my presence in your lives. Thank you for giving me the time, place, and quiet I needed to study, read, and write. I could not have accomplished this without your support, unconditional encouragement and patience. My parents Younis and Hanan, who instilled in me a passion for learning and a desire to do my personal best, my most sincere and deepest gratitude for your never ending, unconditional love, encouragement, guidance, wisdom and support. My wonderful brothers Tariq, Hisham and Mohammed and my lovely sister Daliah, each of you has helped to shape the human being I am becoming. I am grateful to all of you for your love, your presence, and your collective impact in my life.
Contents

Abstract iii

Acknowledgements v

1 Introduction 3

1.1 Curse of Dimensionality ........................................... 4
1.1.1 Illustrative examples ........................................... 4
1.1.2 Dimensionality reduction ...................................... 6
1.2 Introduction to Feature Selection ................................ 7
1.3 Boosting Algorithm .................................................. 10
1.4 Motivations .......................................................... 11
1.5 Scope and Contributions of The Thesis ......................... 13
1.6 Organization of The Thesis ........................................ 15

2 Feature Selection in Machine Learning 17

2.1 Characteristics of Feature Selection Algorithms ............... 17
2.2 Feature Selection Categories ...................................... 18
2.2.1 Filter methods .................................................. 18
2.2.2 Wrapper methods .............................................. 19
2.2.3 Embedded methods ............................................. 19
2.3 Subset Search Methods ............................................. 20
2.4 Feature Selection with Hypothesis Margin ..................... 21
2.4.1 RELIEF algorithm ............................................. 22
2.4.2 Simba algorithm ................................................ 23
2.5 Feature Selection in Ensemble Methods ......................... 23
2.5.1 Ensemble methods .............................................. 23
2.5.2 Boosting-based feature selection techniques ............... 25
2.5.3 Decision stumps ............................................... 25
2.5.4 BDSFS algorithm ............................................... 26

vi
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.5</td>
<td>AdaBoost-CR algorithm</td>
</tr>
<tr>
<td>2.6</td>
<td>Limitations of Existing Methods</td>
</tr>
<tr>
<td>2.7</td>
<td>Learning from Imbalanced Data</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Feature selection on imbalanced data</td>
</tr>
<tr>
<td>2.7.2</td>
<td>FAST algorithm</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Limitations of existing methods</td>
</tr>
<tr>
<td>2.8</td>
<td>Comparison of Feature Selection Algorithms</td>
</tr>
<tr>
<td>2.8.1</td>
<td>Performance evaluation of the selected feature subsets</td>
</tr>
<tr>
<td>2.8.2</td>
<td>Re-sampling techniques for the performance evaluation</td>
</tr>
<tr>
<td>2.8.3</td>
<td>Benchmark data sets</td>
</tr>
<tr>
<td>2.9</td>
<td>Chapter Conclusions</td>
</tr>
<tr>
<td>3</td>
<td>Feature Weighting and Selection Using Hypothesis Margin of Boosting</td>
</tr>
<tr>
<td>3.1</td>
<td>Margin Concept</td>
</tr>
<tr>
<td>3.2</td>
<td>Notation</td>
</tr>
<tr>
<td>3.3</td>
<td>AdaBoost Algorithm</td>
</tr>
<tr>
<td>3.3.1</td>
<td>AdaBoost margin</td>
</tr>
<tr>
<td>3.3.2</td>
<td>AdaBoost and overfitting</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Average margin bound</td>
</tr>
<tr>
<td>3.4</td>
<td>Margin Fraction for Feature Weighting and Selection</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Stumps and larger trees as weak learners</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Margin fraction weight (MF)</td>
</tr>
<tr>
<td>3.5</td>
<td>AUC Margin Fraction Weighting and Selection</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Preliminaries</td>
</tr>
<tr>
<td>3.5.2</td>
<td>AUC margin fraction weight (AMF)</td>
</tr>
<tr>
<td>3.6</td>
<td>Boosting for Feature Selection</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Standard boosting</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Boosting with feature removal</td>
</tr>
<tr>
<td>3.7</td>
<td>Design and Implementation</td>
</tr>
<tr>
<td>3.7.1</td>
<td>SBS-MF algorithm</td>
</tr>
<tr>
<td>3.7.2</td>
<td>SBS-AMF algorithm</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Computational complexity analysis</td>
</tr>
<tr>
<td>3.8</td>
<td>Chapter Conclusions</td>
</tr>
<tr>
<td>4</td>
<td>Experimental Evaluation of The Proposed Feature Selection Algorithms</td>
</tr>
<tr>
<td>4.1</td>
<td>Data sets</td>
</tr>
<tr>
<td>4.1.1</td>
<td>UCI datasets</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Imbalanced datasets</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

vii
4.2 Performance Assessment

4.2.1 Learning algorithms

4.2.2 Results evaluation

4.3 Performance Evaluation of SBS-MF Algorithm

4.3.1 Performance results

4.3.2 Comparison between margin fraction and contribution ratio

4.4 Performance Evaluation of SBS-AMF Algorithm

4.4.1 Experimental setup

4.4.2 Performance results

4.4.3 The effect of different class ratios

4.5 Discussion

4.5.1 Analyzing performance

4.5.2 Impact of correlated features

4.5.3 Addressing potential weaknesses

4.6 Chapter Conclusions

5 Intrusion Detection System for Virtualization: Framework and Data

5.1 Intrusion Detection System

5.2 Virtualization

5.2.1 Process VM

5.2.2 System VM

5.3 Related work

5.3.1 Program-level IDS

5.3.2 Operating system-level IDS

5.3.3 Virtual machine monitor-level IDS

5.3.4 IDS comparison

5.4 Proposed VMM IDS

5.5 The Pattern Builder Phase

5.5.1 Raw data

5.5.2 Feature generation

5.5.3 Feature framework design and implementation

5.6 The Pattern Selector Phase

5.6.1 Related work to feature selection for IDS

5.6.2 VMM IDS features

5.6.3 Applying SBS-AMF to the VMM-IDS data

5.7 The Detector Phase

5.7.1 Supervised learning

5.7.2 Unsupervised learning
5.8 Enhance VMM IDS Detection Speed ................................................. 110
5.8.1 CUDA programming model ...................................................... 110
5.8.2 LOF CUDA implementation ...................................................... 111
5.9 Chapter Conclusions ................................................................. 112

6 Intrusion Detection System for Virtualization: Performance Evaluation 113
   6.1 Experimental Setup ............................................................... 113
   6.2 VMM IDS Feature Reduction Study .......................................... 115
       6.2.1 VMM IDS features issues ............................................... 115
       6.2.2 Feature analysis ............................................................ 117
   6.3 VMM IDS Detection Performance Evaluation: Supervised Learning .... 119
   6.4 VMM IDS Detection Performance Evaluation: Unsupervised Learning .. 121
   6.5 VMM IDS Performance Detection Speed Results .......................... 126
   6.6 Discussion ........................................................................... 130
       6.6.1 Impact of sliding window size ......................................... 130
       6.6.2 Threshold selection ......................................................... 130
       6.6.3 Alarming mechanism ....................................................... 132
   6.7 Chapter Conclusions ............................................................... 133

7 Summary and Conclusion ............................................................ 135
   7.1 Contributions ........................................................................ 136
   7.2 Future Work ......................................................................... 137

Bibliography ................................................................................. 139
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Learning process.</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Example of gene selection problem.</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Example of learning in intrusion detection.</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>Two principles of dimensionality reduction.</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>AdaBoost algorithm.</td>
<td>10</td>
</tr>
<tr>
<td>1.6</td>
<td>Classifier’s performance usually degrades for a large number of features.</td>
<td>11</td>
</tr>
<tr>
<td>2.1</td>
<td>Feature selection process.</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Filter method.</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Wrapper method.</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Embedded method.</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>Subset search methods.</td>
<td>21</td>
</tr>
<tr>
<td>2.6</td>
<td>A logical view of the ensemble learning method [96].</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>An example of a decision stump that discriminates between two of three classes of Iris flower data set: Iris versicolor and Iris virginica.</td>
<td>25</td>
</tr>
<tr>
<td>2.8</td>
<td>Example of imbalanced data.</td>
<td>29</td>
</tr>
<tr>
<td>2.9</td>
<td>ROC curve.</td>
<td>33</td>
</tr>
<tr>
<td>2.10</td>
<td>k-fold cross validation.</td>
<td>34</td>
</tr>
<tr>
<td>3.1</td>
<td>Sample Margin (SM) measures how much can an instance travel before it hits the decision boundary. On the other hand Hypothesis Margin (HM) measures how much can the hypothesis travel before it hits an instance [28].</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Boosting $H(x)$, $\rho(x)$ and CDF values for training instances for Prostate dataset after 5 rounds.</td>
<td>44</td>
</tr>
<tr>
<td>3.3</td>
<td>Boosting $H(x)$, $\rho(x)$ and CDF values for training instances for Prostate dataset after 10 rounds.</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>Boosting $H(x)$, $\rho(x)$ and CDF values for training instances for Prostate dataset after 50 rounds.</td>
<td>46</td>
</tr>
</tbody>
</table>
6.1 Two dimensional scatter plots. Examples of a) redundant strongly correlated features, b) presumably redundant features, c) class conditional dependent features, d) irrelevant features. ................................................................. 116

6.2 Impact of boosting number of rounds on AUC for EnhancedBoost algorithm. ..... 119

6.3 ROC results for the VMM IDS detection performance for each workload with KNN and LOF using two different feature selection methods. ........................................ 122

6.4 Example of detecting a Downloader Malware with LOF using the selected features of SBS-AMF((Top), and AdaBoost-CR (Bottom). ......................................................... 125

6.5 Impact of each factor on execution time of the VMM-IDS using LOF algorithm (right plots in log-scale). ............................................................. 127

6.6 VMM IDS speed-up. (Top) impact of feature set size. (Bottom) impact of neighborhood size. .............................................................. 129

6.7 Sliding window. ........................................................................... 130

6.8 Impact of window size. ................................................................. 131

6.9 Box-plot example. ................................................................. 132

6.10 The boxplot for each workload based on normal data. ......................... 133
List of Tables

2.1 Characteristics of each feature selection category. ........................................... 20
4.1 UCI data sets description. .................................................................................. 63
4.2 Imbalanced data sets description. .................................................................... 64
4.3 Classification accuracy rates obtained by 1-NN and SVM without feature selection. 66
4.4 A comparison of classification accuracies of classifiers using four feature selection algorithms on UCI datasets. ................................................................. 67
4.5 $F_1$ measure classification rates obtained by 1-NN and SVM without feature selection. 76
4.6 A comparison of classification accuracies using $F_1$ measure of classifiers using four feature selection algorithms on microarray datasets. ............................. 76
4.7 Iris data set statistics. ......................................................................................... 81
4.8 Correlation coefficient for features of Iris data. ................................................ 83
5.1 VM Taxonomy [92]. ......................................................................................... 88
5.2 IDS level comparison ....................................................................................... 91
5.3 Summary of the Virtual events, the additional information extracted for the events, and what they mean for the OS. ................................................................. 96
5.4 Examples of basic features constructed for the VMM-IDS. ............................... 102
5.5 Examples of complex features constructed for the VMM-IDS. ........................... 102
6.1 Normal Workload (appliances). ...................................................................... 114
6.2 Malware Categories. ....................................................................................... 114
6.3 Top 5 selected features by SBS-AMF for Database workload. ....................... 118
6.4 Top 5 selected features by SBS-AMF for Web workload. ............................... 118
6.5 Top 5 selected features by SBS-AMF for EMail workload. ............................. 118
6.6 Supervised learning results: a comparison of detection accuracy performance for each workload using, EnhancedBoost, and AdaBoost. ............................. 120
6.7 Unsupervised learning results: a comparison of detection accuracy performance of $KNN$ for each workload using two feature selection algorithms. ........... 123
6.8 Unsupervised learning results: a comparison of detection accuracy performance of LOF for each workload using two feature selection algorithms. .......................... 124
6.9 Comparison of the VMM IDS computation time, with respect to the size of data set (n) and the number of neighbors (k). All numbers in seconds. ............................ 128
6.10 Comparison of the VMM IDS computation time, with respect to the size of data set (n) and the feature set size (No.features). All numbers in seconds. .......................... 128
List of Symbols

General mathematical symbols, variables, and operations:

- \( \mathbb{R} \): set of real numbers
- \( \mathbb{N} = \{1, 2, 3, \cdots \} \): set of all natural numbers
- \( \log(x) \): natural logarithm of \( x \)
- \( \log_b(x) \): logarithm to the base \( b \) of \( x \)
- \( \text{argmax}_x f(x) \): the value of \( x \) that leads to the maximum of \( f(x) \)
- \( \text{argmin}_x f(x) \): the value of \( x \) that leads to the minimum of \( f(x) \)

Sets and functions:

- \( X \subseteq \mathbb{R}^n \): definition domain of the input set of features
- \( S = \{(x_i, \ell_i)\}_{i=1}^m \): finite set of \( m \in \mathbb{N} \) labeled examples, \( \ell_i \in \mathcal{L} \) is a class label corresponding to the sample \( x_i \in X \)
- \( \Upsilon = \{\upsilon_1, \upsilon_2, \cdots, \upsilon_n\} \): finite set of \( n \in \mathbb{N} \) input features
- \( D \subseteq S \): training (design) data set

Boosting algorithm symbols:

- \( \mathcal{H} \): a hypothesis space
- \( h \): a base learner
- \( T \in \mathbb{N} \): number of rounds
- \( \epsilon \): weighted classification error
- \( \alpha \): hypothesis weight vector
- \( \rho \): the margin parameter
- \( \rho^* \): the maximum margin
- \( \bar{\rho} \): the average margin
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMF</td>
<td>AUC Margin Fraction</td>
</tr>
<tr>
<td>AUC</td>
<td>Area Under ROC curve</td>
</tr>
<tr>
<td>BER</td>
<td>Balanced Error</td>
</tr>
<tr>
<td>BDSFS</td>
<td>Boosted Decision Stump Feature Selection</td>
</tr>
<tr>
<td>CHI</td>
<td>Chi-square Statistic</td>
</tr>
<tr>
<td>CR</td>
<td>Contribution Ratio</td>
</tr>
<tr>
<td>FN</td>
<td>False Negative</td>
</tr>
<tr>
<td>FP</td>
<td>False Positive</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>HM</td>
<td>Hypothesis Margin</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IG</td>
<td>Information Gain</td>
</tr>
<tr>
<td>$k$-NN</td>
<td>$k$ Nearest Neighbour</td>
</tr>
<tr>
<td>LOF</td>
<td>Local Outlier Factor</td>
</tr>
<tr>
<td>MF</td>
<td>Margin Fraction</td>
</tr>
<tr>
<td>OR</td>
<td>Odds Ratio</td>
</tr>
<tr>
<td>ROC</td>
<td>Receiver Operating Characteristic</td>
</tr>
<tr>
<td>SBS</td>
<td>Sequential Backward Search</td>
</tr>
<tr>
<td>SFS</td>
<td>Sequential Forward Search</td>
</tr>
<tr>
<td>SM</td>
<td>Sample Margin</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>TP</td>
<td>True Positive</td>
</tr>
<tr>
<td>TN</td>
<td>True Negative</td>
</tr>
<tr>
<td>VMM</td>
<td>Virtual Machine Monitor</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

With the widening availability and small size of modern computer systems, intelligent learning systems are rapidly gaining popularity for wide ranges of applications. Learning systems have found their way to all manners of application domains: the stock market, financial customer modeling and risk assessment, industrial monitoring and control, assembly robotics, global and personal information retrieval and filtering, and the computer entertainment industry. This success is easily attributed to the fact that learning systems are cost-effective when they are applicable. Computing equipment prices have dropped significantly over the past decade, while human expertise has remained steadily expensive. Even a fraction of the knowledge of a highly paid and competent consultant built into a computation system is clearly very desirable. In addition, a system that learns automatically from historical data typically works faster than a human expert.

The process of a learning system might involve presenting the learning algorithm with a set of examples, each of which contains a set of observations (features) and corresponding task. This provides the experience to the learner. Later, the learner uses this experience to perform the required task. Its performance determines the effectiveness in learning the task. Consider a learner that is trying the skill of predicting the next day’s weather based upon today’s climatic conditions. In this case the set of feature values would be the climatic conditions recorded on a particular day in the past. They might contain that day’s high and low temperatures, humidity, rainfall, etc. and the prediction would be the corresponding weather for the next day. We want learners that are effective for future examples.

The learning process, known as recognition/classification, is characterized by one of the two following tasks. *Supervised* classification is a problem of establishing decision regions between examples/patterns and assigning an unknown pattern into one of the predefined classes. In *unsupervised* classification, classes are learned based on the similarity of patterns. The learning system operates in two modes - *learning* (training) from a given set of examples and *classification* (testing), see Figure1.1.
Figure 1.1: Learning process.

1.1 Curse of Dimensionality

Unfortunately, however, most learning systems suffer from intractability issue known as the curse of dimensionality. Learning systems exhibit intractable behavior with respect to two different parameters: the number of distinct examples to be processed by the system; and the number of observable features per example, known as dimensionality.

The former typically affects only the training stage of the system and, depending on the system’s ultimate intended use, may be acceptable. Dimensionality, on the other hand, adversely affects both the training and runtime stages of a learning system’s lifetime.

Many systems exhibit non-polynomial complexity with respect to dimensionality. This imposes a ceiling on the applicability of such approaches, especially to real world applications. The curse of dimensionality limits the applicability of learning systems to small, well-analyzed domains, rendering otherwise elegant methodologies incapable of performing satisfactorily on arbitrary domains.

1.1.1 Illustrative examples

The following examples of two real applications in genomic microarray analysis [108, 110] and intrusion detection [64, 11] domains, show the issues arising from learning with high dimensional data.

Example of gene selection problem

In the gene selection problem, a machine learning algorithm might be provided with the information about a set of patients in a hospital (e.g., information from the doctor’s interview of the patient, results from lab tests, tissue biopsy, etc.). The algorithm task is to separate healthy patients from cancer patients based on their “profile”. The machine learning algorithm constructs a model trying to identify which information is predictive of cancer. One of the key problems for this task is to determine which information (which features) describing the patient is important and which is not.
CHAPTER 1. INTRODUCTION

Figure 1.2: Example of gene selection problem.

Usually fewer than 100 examples (patients) are available altogether for training and testing. But the number of features of instances under consideration is extremely large (e.g., ranges from 6000 to 60,000), while only a small subset of features is found to be responsible for causing cancer [43]. Figure 1.2 shows an example of diagnostics task in medicine and genetic engineering problem. Developing methods that can be used to find or select an appropriate subset of features from a large set of features is very important.

Example of intrusion detection system

Intrusion detection system (IDS) is a mechanism designed to detect abuses, misuses, and unauthorized uses of a computer system, caused by either insiders or outsiders—typically identify intrusions by recognizing known patterns of attacks or by revealing anomalous behavior of protected resources: network traffic, memory accesses, and/or system calls. The main purpose is to detect most of the attacks, give very few false alarms, copes with large amount of data, and is fast enough to make real-time decisions.

Designing IDS requires collecting an enormous amount of data, see Figure 1.3. A single data set can include thousands of extracted raw events; an event trace is usually in a form that is not suitable for performing detection. Therefore, the IDS constructs features from the event traces. Some of these features may be irrelevant, poor predictors for identifying intrusions, or redundant due to their high inter-correlation with other features. Moreover, the process of constructing features involves significant computation. Therefore, learning from high dimensional intrusion data not only hinders the detection speed but also has significant impacts on the detection performance of IDS.
1.1.2 Dimensionality reduction

A common solution is to reduce the dimensionality of the input feature space and to employ only those features that are most relevant to the given problem. A fewer features allow to design less complex classifiers with better parameter estimates, and perhaps with improved prediction accuracy. There are two approaches leading to dimensionality reduction. They are referred to as:

i) Feature Selection—outputs a subset of the original features without any further change.

ii) Feature Extraction—transforms the input features into a completely different space.

A block diagram of both principles is shown in Figure 1.4.
It is a matter of a specific application and character of the data which of the two methods to choose. Although the feature extraction is more general and the transformation mapping may provide features with a better discriminatory ability than the best subset of the input features, new features may not have clear (physical) interpretation. In addition, there are no savings in the measurement cost and data storage as the entire original feature space is needed.

The feature selection is beneficial especially for problems where some sensory inputs are likely to carry a little of useful information for the class discrimination, or if there are very strong correlations between sets of input observables so that very similar information is repeated in several variables, or if measurements on an examined object (or process) are costly and it is advisable to reduce their number. Furthermore, features keep their original physical meaning because no transformation of data is made. This may be important for a better problem understanding in some applications (e.g., in medicine) as only relevant information is analyzed. The scope of both approaches is very broad. In this thesis, we focus only on the feature selection topic.

In this chapter, we present to the reader an introduction to some basic feature selection concepts. This includes a very brief overview of the scope of feature selection definition, and an introduction to the categories of feature selection that are common in machine learning. We discuss in detail boosting algorithm and the concept of margin, as this concept is directly relevant to this work. We then describe the challenges and our motivations. Finally, we discuss the scope and contributions of our work, and give an overview of the remainder of thesis.

1.2 Introduction to Feature Selection

Feature selection is defined as the preprocessing process aimed at identifying a small subset of highly predictive features out of a large set of raw input variables that are possibly irrelevant or redundant. It plays a fundamental role in the success of many learning tasks where high dimensionality arises as a big challenge, most evidently, in text categorization [36], image retrieval [95, 34], genomic microarray analysis [108, 110], and intrusion detection [64, 11].

Feature selection formulation problem

Definition 1.2.1 In classic supervised learning, one is given a training set $\mathcal{S}$ of labeled instances $m \in \mathbb{N}$ (observations) acquired for the examined object (or process). An instance $m$ is typically described as an assignment of feature set $\mathcal{U} = \{\upsilon_1, \upsilon_2, \cdots, \upsilon_n\}$, and it is characterized by a label $\ell_i \in \mathcal{L}$, where $\mathcal{L} = \{\ell_1, \ell_2, \cdots, \ell_c\}$ is a finite set of $c \in \mathbb{N}$ class labels. Consider a criterion function $J : \Phi_k \to \mathbb{R}$ scoring the quality (e.g., discriminatory ability) of feature subsets.

Let us assume that the higher the criterion value is the more useful information the features carry. Feature selection can be viewed consequently as a problem of finding such a combination of
features, which maximizes the criterion function $J$.

The goal is to find the best subset of $d$ features that contribute most to class discrimination. The number of possible such subsets is $n!/(d!(n-d)!)$, which can be very large even for moderate values of $n$ and $d$. Therefore, one resorts to various heuristics for searching through the space of possible features.

In other words, the task for feature selection is to induce a hypothesis (classifier) that accurately predicts the labels of novel instances. The learning of the classifier is inherently determined by the feature-values. In theory, more features should provide more discriminating power, but in practice, with a limited amount of training data, excessive features will not only significantly slow down the learning process, but also cause the classifier to over-fit the training data as irrelevant or redundant features may confuse the learning algorithm [110]. Feature selection is the process of identifying and removing as much irrelevant and redundant information as possible.

Throughout this work we shall assume the following: all measurements on an examined object are assumed to have continuous values, and for simplicity the further work is restricted only to dichotomy, i.e., two-class problems (multiple-class problems can be converted to dichotomy), where sets of labels are assumed to be $\mathcal{L} = \{-1, +1\}$.

**Feature selection goals**

The main reasons for applying the feature selection are basically the following:

i) Features can be expensive to acquire. This includes measurement cost, data preprocessing, their transfer and storage, and perhaps computational reasons.

ii) More features need more samples for training to get a good generalization capability of a classifier (recall the curse of dimensionality). Furthermore, data is usually sparse in high dimensional spaces and this leads to over-fitting and essentially bad parameter estimates as not enough samples are often available.

The effects of feature selection have been widely recognized for its abilities in [3, 17]:

i) Facilitating data interpretation.

ii) Reducing measurement and storage requirements.

iii) Increasing processing speeds.

iv) Defying curse of dimensionality.

v) Improving generalization performance.
CHAPTER 1. INTRODUCTION

Classes of Feature Selection

Feature selection algorithms for machine learning typically fall into three broad categories [54]: those that use feature selection to filter features passed to induction [51], those that treat feature selection as a wrapper around the induction process [54] and those embed the selection within the basic induction algorithm [60]. Filter methods can be a fast and easy solution, but they are not usually optimal since they do not account for the mechanism of the learning algorithm utilized. Wrapper methods consist of searching for the subset of features that minimize the generalization error. These methods often give better results (in terms of the final predictive accuracy of a learning algorithm) than filters and embedded methods because feature selection is optimized for the particular learning algorithm used. However, since a learning algorithm is employed to evaluate each and every set of features considered, wrappers are prohibitively expensive to run, and can be intractable for large databases containing many features since the search is a combinatorial problem that is NP-hard. Furthermore, since the feature selection process is tightly coupled with a learning algorithm, wrappers are less general and must be re-run when switching from one learning algorithm to another.

In embedded methods, as the name suggests, feature selection is embedded into the learning algorithm. The major advantages of embedded methods are the significant reduction in computational time and the data-driven knowledge discovery of important features in classification. The computational speedup is derived from the fact that the embedded method is able to determine significant features from the greatly reduced training sets within a few iterations. The cost of finding an optimal subset of features does not overshadow the benefit of reduced overall training time.

Feature selection search strategies

There are many strategies for feature selection. For example, one can define an objective function, e.g., one that measures accuracy on a fixed held out set, and use sequential forward or backward selection.

A Sequential Forward Selection (SFS) is a bottom-up search where new features are added to a feature set one at a time. At each stage, the chosen feature is one that, when added to the current set, maximizes the objective. The feature set is initially empty. The algorithm terminates when the best remaining feature worsens the objective, or when the desired number of features is reached. The main disadvantage of this method is that it does not delete features from the feature set once they have been chosen. As new features are found in a sequential, greedy way, there is no guarantee that they should belong in the final set.

On the other hand, Sequential Backward Selection (SBS) is the top-down analog of SFS: Features are deleted one at a time until $d$ features remain. In this proposal a backward sequential selection is used because of its lower computational complexity compared to other algorithms and its optimality in the subset selection problem [27].
1.3 Boosting Algorithm

Boosting has attracted much attention in the machine learning community mainly because of its excellent performance and computational attractiveness for large datasets [88, 71].

The main breakthrough came with Freund and Schapire’s most successful AdaBoost algorithm [37, 38]. The essence of AdaBoost is to learn a number of simple weak classifiers that are linearly combined into a single strong classifier (see Figure 1.5). The major advantage of AdaBoost algorithm is the adaptive selection of discriminative and complementary features during the training process which most often yields to better feature or variable selection while keeping or even increasing the prediction accuracy.

**Boosting margin concept**

AdaBoost has the property that it does not often seem to suffer from overfitting, even after a large number of iterations [20, 84]. Generally, a learning algorithm is said to overfit if it is more accurate in fitting known data (training) but less accurate in predicting new data (testing). Schapire et al., [87] explained, to some extent, a reasonable explanation to the success of AdaBoost by the margin theory. The margin of a boosted classifier is a number between $-1$ and $1$, that according to the margin theory, can be thought of as a “confidence” measure of a classifier’s predictive ability, or as a guarantee on the generalization performance. If the margin of a classifier is large, then it tends to perform well on test data. If the margin is small, then the classifier tends not to perform so well. Furthermore, Schapire et al., showed that AdaBoost has a tendency to increase the margins on the training examples. Thus, though not entirely complete, their theory and experiments strongly supported the notion that margins are highly relevant to the behavior and generalization performance of AdaBoost.
CHAPTER 1. INTRODUCTION

1.4 Motivations

The computational cost of classification grows quadratically with data dimension size, making feature selection an important issue, see Figure 1.6.

Filter based feature selection algorithms can be a fast and easy solution, but they are not usually optimal since they do not account for the mechanism of the learning algorithm utilized. Wrapper methods, on the other hand, often give better results (in terms of the final predictive accuracy of a learning algorithm) than filters and embedded methods because feature selection is optimized for the particular learning algorithm used.

However, since a learning algorithm is employed to evaluate each and every set of features considered, wrappers are prohibitively expensive to run, and can be intractable for large databases containing many features since the search is a combinatorial problem that is $NP$-hard. Furthermore, since the feature selection process is tightly coupled with a learning algorithm, wrappers are less general and must be re-run when switching from one learning algorithm to another.

The following are the main motivations for our work:

i) The limitations of existing research clearly suggest that we should pursue a different framework of feature selection that allows efficient analysis of both feature relevance and redundancy for high-dimensional data. For these reasons, an embedded approach to feature selection for machine learning is explored in this thesis. The major advantages of embedded methods are the significant reduction in computational time and the data-driven knowledge discovery of important features in classification. The computational speedup is derived from the fact that the embedded method is able to determine significant features from the greatly reduced training sets within a few iterations. The cost of finding an optimal subset of features does not overshadow the benefit of reduced overall training time.

ii) Boosting has attracted much attention in the machine learning community mainly because of its excellent performance and computational attractiveness for large datasets [88, 71], and has been used, in particular for feature selection, with great success in many applications like face recognition [90, 32], text mining [105] and intrusion detection [11, 44], we show that our boosting-based proposed work introduces significant enhancements to currently-existing techniques. These benefits are detailed in Chapter 3.

iii) Recently, there has been a growing line of research in utilizing the concept of Margin for feature selection...
selection. Margins play a crucial role in the current machine learning research as they tend to give a strong indication of a learner’s performance in practice. To our knowledge, almost no previous work has exploited the characteristics of the hypothesis margins of boosting to determine the quality of features.

iv) Using the right feature selection method is critical to achieve the best performance when the prediction model is trained using an imbalanced data set. In such cases, standard classifier performance can be hindered, since classification tends to be overwhelmed by the majority classes and ignores minority classes. Traditional feature selection methods fail to account for imbalanced class distributions, leading to poor predictions for minority class samples. Here we propose to extend our proposed boosting-based feature selection to handle the presence of data with imbalance class distributions.

v) Virtualization is becoming an increasingly popular service hosting platform. Recently, IDSs which utilize virtualization have been introduced. One particular challenge is considered here, learning from high dimensional imbalanced data, which is common in security-related system, and is an intrinsic problem to intrusion detection. Using our proposed feature selection algorithms we aim to design a lightweight, efficient and effective Virtualization IDS.

This thesis enhances the current state-of-the-art and makes key contributions to the following areas:

- **Machine Learning/Datamining** – Devising classifier independent (embedded) feature selection techniques which should be able to reduce the dimensionality of the given problem by eliminating redundant and irrelevant features and handle the presence of imbalanced data using the concept of the hypothesis margin of boosting.

- **Cyber Security for Virtualization** – Applying the proposed feature selection to design a novel lightweight anomaly intrusion detection in virtualization environments.

- **Graphics Processing Unit (GPU) for Machine Learning** – Developing a GPU-enabled CUDA implementation to accelerate the *Local Outlier Factor* (LOF) anomaly detection algorithm.

Next, we discuss the major contributions of this work and describe the organization of the remainder of the thesis.
1.5 Scope and Contributions of The Thesis

The thesis consists of two main parts: part one focuses on the basic research issues arising in feature selection, and claims that feature selection for supervised machine learning tasks can be accomplished on the basis of the margin that boosting algorithm produces. In particular, this part of the thesis investigates the following hypothesis:

A good feature subset is one that contains features that contribute more to the margin distribution associated with the weighted linear combination that boosting produces.

In part two, we apply the proposed feature selection method to build a real application in the domain of cyber-security.

The contributions of this thesis are:

**Proposed feature selection algorithm**

Devised a classifier independent (embedded) feature selection technique which should be able to reduce the dimensionality of the given problem by eliminating redundant and irrelevant features. In particular, we presented a novel framework of margin based weighting for feature selection which extensively explores the hypothesis margins of boosting.

We introduce the idea of utilizing the training examples’ average margin to measure the quality and the relative importance of features. For this purpose, we proposed an evaluation function which assigns weights to subsets of features according to the margin they induce. The weight, termed Margin Fraction (MF), is measured by computing the cumulative effect each feature has on the average margin associated with the weighted linear combination that boosting produces, i.e., the margin fraction that is due to a feature. The problem of searching the “best” subset of features is solved by means of a greedy algorithm based on backward selection [54]. This work has been featured in the IEEE International Conference on Data Mining series (ICDM).

**Imbalanced data classification in the context of feature selection**

Most current existing feature selection methods fail to account for imbalanced class distributions, leading to poor predictions for minority class samples, we proposed to extend the weighting and selection methodology to handle the presence of imbalanced data, by defining a new weight metric that characterize the quality of a set of features based on the maximized the Area Under ROC curve (AUC) margin it induces during the process of learning with boosting. We refer to it as AUC Margin Fraction (AMF). This work has been featured in the International Conference on Machine Learning and Applications (ICMLA) [6], and has been presented at the Broadening Participation in Data Mining Workshop, co-located with
the Society for Industrial and Applied Mathematics (SIAM) International Conference on Data Mining (SDM) [5] and at Women in Machine Learning (WiML) [8] workshop.

Evaluation methodology

We created grounds for a fair comparison of the proposed feature selection algorithms through extensive comparisons with other state-of-the-art methods, including boosting-based and margin-based feature selection algorithms using real-world high-dimensional data from UCI repository [77] and datasets that contain a large number of features with a small number of samples and significant imbalance between the two classes. We also try to address potential issues in feature selection, such as the impact of the existence of correlated features, and exploring differences in performance on highly imbalanced classes versus balanced classes, and we lay out the limitations of our proposed methods and how these potential weaknesses can be addressed.

Developing lightweight anomaly intrusion detection system in virtualization

We applied the proposed feature selection algorithm to design effective IDS for virtual server environments, by addressing the following issues:

1. Constructing different types of features from raw data (events) utilizing information embedded within the virtual machine monitor (VMM) level.
2. Handling the presence of excessive features in an intrusion data whose class distributions are imbalanced.
3. Designing a novel supervised detection algorithm based on Enhanced boosting algorithm with Decision Stumps to detect various categories of attacks.
4. Applying two unsupervised anomaly detection algorithms by training only with normal data with four different commercial virtual appliances and on a set of approximately 300 real-world malwares. We will show that with an average of 5% false alarms, it is possible to achieve an average of 96% true detections.

This work has been published in Secure and Resilient Architectures and Systems (SRAS) [10], the IEEE International Conference on Tools with Artificial Intelligence (ICTAI), the Operating Systems Review (OSR) journal [13], and the International Conference on Machine Learning and Applications (ICMLA) [11]. In addition, an invited talk was given on this work at the Northeastern University/MIT Lincoln Labs Cyber Security Meeting [4].

Accelerating LOF algorithm with GPUs for IDS

We developed a GPU-enabled CUDA implementation for the Local Outlier Factor (LOF) [21] method. Our motivating application was to accelerate the IDS framework. This application
CHAPTER 1. INTRODUCTION

requires that we can perform both timely and accurate detection of anomalies. LOF is a very powerful outlier detection method; however, it is very computationally expensive. Thus, it becomes a challenging issue when it comes to designing efficient and reliable IDS. Due to the growing popularity of GPUs in general-purpose computing domains we applied this parallel computing approach to accelerate LOF. We achieved more than a 100X. This work has been presented and selected as a best poster in Research, Innovation and Scholarship Expo (RISE) [9]. Also it has been published in the Workshop on General Purpose Processing Using GPUs [7].

1.6 Organization of The Thesis

Chapter 2

In this chapter, we review state-of-the-art and related work. We start with an overview of the scope of feature selection approaches in machine learning. We then give a summary of existing feature selection techniques that are related to our proposed approach. This is essential to understand and address several of the limitations of each of these techniques. We also review the issues arising in learning with high-dimensional imbalanced data.

Chapter 3

This chapter introduces the theory and math basis for the proposed hypothesis weighting and selection for feature selection. We develop in detail the methods to both construct boosting hypothesis from features using decision stumps and to evaluate the quality of features based in their relevance to the induced margin. We then show how to extend the proposed methodology to handle the presence of data with imbalance class distributions, by defining new weighting function based on the maximized AUC margin induced during the boosting learning process over positive and negative samples.

Chapter 4

Here we describe the effectiveness of our proposed feature selection algorithms through extensive comparisons with other state-of-the-art feature selection techniques. The results are obtained using real-world high-dimensional data.

Chapter 5

In this chapter, we discuss how to design a real intrusion detection system (IDS) of virtual server environments utilizing only information available from the perspective of the virtual machine monitor (VMM). We begin with a short overview of the concept of IDS and virtualization technology, and
the related work. We also describe the main goals and advantages of designing such VMM IDS, and we provide details about the proposed VMM IDS framework, including: data gathering, feature extraction, feature selection and anomaly detection.

Chapter 6

Here we provide evidence that an effective VMM IDS can be constructed. We first discuss the features reduction study and analysis. We then present the accuracy results we obtain using, first the supervised learning method (boosting), and then with unsupervised learning methods (KNN and LOF). Finally, we show how to accelerate the detection speed performance of the VMM IDS by applying a GPU parallel computing approach to accelerate the LOF algorithm.

Chapter 7

This chapter concludes our work and suggests possible topics for future research.
Chapter 2

Feature Selection in Machine Learning

In this chapter, we present background information on the concept of feature selection and review work related to our research. We begin by summarizing basic principles applied in feature selection such as filters, wrappers and embedded methods. We then, review the related literature, including: margin-based and boosting-based feature selection algorithms and we show their limitations. Next, we address the issues and the problems occur in learning with high-dimensional imbalanced data. Finally, we outline several problems arising in a design and evaluation methodology of various feature selection algorithms.

2.1 Characteristics of Feature Selection Algorithms

There are many approaches to feature selection proposed in the literature, however, all in principle involve two main ingredients (as shown in Figure 2.1):

i) A search strategy which explores the set of all feature subsets in a purposeful manner.

ii) A criterion (objective) function which evaluates those feature subsets.

The search strategy is independent of the criterion function used. The best subset of features is found by optimizing (usually maximizing) the

Figure 2.1: Feature selection process.
criterion function. The best performance of the selected features can be achieved when both the feature selection and classification stages are optimized together using the same criterion function [61], e.g., the prediction accuracy.

The criterion function guiding the search for the best features is usually some kind of separability measure between classes. It can be either classifier independent (i.e., filter approach) or classifier specific (i.e., wrapper approach or embedded method) [54].

2.2 Feature Selection Categories

Feature selection methods can be grouped into one of three groups: filter, wrapper, and embedded. The subsequent sections will describe each method in detail.

2.2.1 Filter methods

Coined by John, Kohavi and Pfleger in [54], filter methods have their name because they filter out irrelevant features before induction occurs. The process uses general characteristics of the training set to select some features and exclude others. Since filtering methods do not involve the use of a learning algorithm to evaluate candidate sets, they can be combined with any learning algorithm after the filtering is complete. Moreover, filter methods are a computationally effective form of data preprocessing, especially when compared to wrapper methods. Figure 2.2 describes a generalized form of a filter algorithm.

Given dataset \( D \), begin with a given subset \( S_0 \) (an empty set, a full set, or any randomly selected subset) and search through the feature space using a particular search strategy. Each generated subset \( S \) is evaluated by an independent measure \( M \) and compared with the previous best. If better, it’s regarded as the current best subset. The search iterates until a predefined stopping criterion is reached. The algorithm outputs the last current best subset as the final feature subset.

One of the famous filter methods is RELIEF algorithm [51]. RELIEF assigns a “relevance” weight to each feature, which represents the relevance of the feature to the target concept. It samples instances randomly and updates the relevance values based on the difference between the selected instance and the nearest instances of the same and opposite class (“near-hit” and “near-miss”). More details about this algorithm will be given in Section 2.4. There are some drawbacks to filter methods however. First, most filtering methods require the pre-selection of a set number of features to be chosen. If the number is too high, irrelevant features will be kept and accuracy may suffer. If the number is too low, useful features may not be selected, once again affecting accuracy.
Some remedies to this problem include using a hold-out set to test for the best \( k \). However, this may negatively impact what is perhaps the biggest benefit of a filtering algorithm, its relative speed. A second drawback is that filtering algorithms may miss features that would otherwise be useful to the learning algorithm which will predict unseen instances. Since the algorithm bases its selection purely on metrics, it may miss features deemed useful by some learning algorithms and perhaps irrelevant in others.

### 2.2.2 Wrapper methods

Wrapper methods occur outside the basic learning algorithm, but also use said learning algorithm as a subroutine, rather than just as a post-processor. For this reason, John, Kohavi, and Pfleger [54] refer to them as wrapper methods.

Each candidate subset is evaluated by running the selected data through the learning algorithm and using the estimated accuracy of the resulting classifier as its evaluation metric as shown in Figure 2.3. As aforementioned, the biggest benefit to using wrapper methods is their tendency to outperform (prediction accuracy) their filter and embedded counterparts. The general argument is that the classifier which will use the feature subset should provide a better estimate of accuracy than a separate metric that may have an entirely different bias.

For each generated subset \( S \), the algorithm evaluates its goodness by applying the learning algorithm to the data with subset \( S \) and evaluating the accuracy. Thus, different learning algorithms may produce different feature selection results. Since learning algorithms are used to control the selection of feature subsets, the wrapper model tends to give superior performance as feature subsets found are better suited to the predetermined learning algorithm. As a result, it’s also more computationally expensive than a filter method. Similar to many wrapper method variations such as the brute force method, branch and bound, sequential backward/forward search, the sequential floating search method, etc.

### 2.2.3 Embedded methods

In embedded methods, see Figure 2.4, the search for an optimal subset of features is built into the classifier construction, and can be seen as a search in the combined space of feature subsets and hypotheses. In other words, the feature selection algorithm is built (embedded) into the classifier model itself rather than using the classifier to evaluate candidate feature sets. Just like wrapper approaches, embedded approaches are thus specific to
Table 2.1: Characteristics of each feature selection category.

<table>
<thead>
<tr>
<th>Category</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>Fast</td>
<td>Ignores feature dependencies</td>
</tr>
<tr>
<td></td>
<td>Scalable</td>
<td>Ignores interaction with classifier</td>
</tr>
<tr>
<td></td>
<td>Independent of the classifier</td>
<td></td>
</tr>
<tr>
<td>Wrapper</td>
<td>Simple</td>
<td>Risk of over fitting</td>
</tr>
<tr>
<td></td>
<td>Interacts with the classifier</td>
<td>Prone to get stuck in a local optimum</td>
</tr>
<tr>
<td></td>
<td>Models feature dependencies</td>
<td>Classifier dependent selection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Computationally intensive</td>
</tr>
<tr>
<td>Embedded</td>
<td>Interacts with the classifier</td>
<td>Classifier dependent selection</td>
</tr>
<tr>
<td></td>
<td>Better computationally intensive than wrapper</td>
<td></td>
</tr>
<tr>
<td></td>
<td>methods</td>
<td></td>
</tr>
</tbody>
</table>

a given learning algorithm. Embedded methods have the advantage that they include the interaction with the classification model, while at the same time being far less computationally intensive than wrapper methods [54].

Table 2.1 highlights a set of characteristics of each feature selection category.

### 2.3 Subset Search Methods

#### Exhaustive search

An exhaustive search for an optimal feature subset of a fixed size $d \in N$ requires to examine many possible combinations. This number grows, however, exponentially with the dimensionality. If the subset size $d$ has to be optimized as well, the search need to explore even up to $2^n$ possible combinations. Notice that problems with exponential dependency are known to be NP-hard. This makes the exhaustive search method computationally prohibitive even for problems of a small dimensionality. The main research has been therefore directed to development of suboptimal search strategies.

#### Suboptimal search

An overview of several traditional subset search methods, their properties, and other feature selection issues are summarized by Kittler [53]. Search algorithms construct candidate subsets of features by adding or removing a small number of measurements. The initial point in the feature space specifies the search direction. According to [72], the following basic search strategies are used for selecting feature subsets:
CHAPTER 2. FEATURE SELECTION IN MACHINE LEARNING

21

(a) Sequential forward selection (SFS)  (b) Sequential backward selection (SBS)

Figure 2.5: Subset search methods.

• Starting with an empty set and subsequent addition of features is referred to as a bottom-up approach or sequential forward selection (SFS).

• Using the full set of features at the beginning and subsequent features removal is called a top-down approach or sequential backward elimination (SBS).

• Starting with a randomly selected feature set and adds randomly selected features or removes them from the set. This strategy is called random mutation.

Plot (a) in Figure 2.5 shows an example of SFS. The top node, where the search starts, contains the empty set. In the first step, we can choose between all features and select the best feature. Afterwards, we check, if the selection of an additional feature will lead to a better subset, and so on. Plot (b) in Figure 2.5, on the other hand, shows an example of SBS, in which the process would take the inverted paths and starts with the complete set of features. Obviously, both strategies can miss the global optimum, if a wrong path has been taken in an earlier step.

2.4 Feature Selection with Hypothesis Margin

Margins are very important in the statistical pattern recognition, because they measure confidence of a classifier with respect to its predictions. Crammer et al. [28] discuss two ways of describing a margin. First considered is a sample margin which is defined as a distance between a sample and a decision boundary created by a classifier. This approach is known from Support Vector Machines, for instance. Secondly, Crammer et al. [28] introduced a hypothesis margin which is defined as a
distance measure between possible decision boundaries, i.e., a distance that a decision boundary can travel without changing labels of the sample points. A good generalization can be expected if many samples have a large margin. In this work we focus on algorithms developed under the hypothesis margin concept.

2.4.1 RELIEF algorithm

The most discussed hypothesis margin method is the RELIEF algorithm presented by Kira and Rendell [51] in 1992. The method is based on the hypothesis margin of a 1-NN classifier. RELIEF evaluates features using the nearest neighbor rule based on how well a feature’s values differentiate themselves from nearby points. When RELIEF selects any specific instance, it searches for two nearest neighbors: one from the same class (the nearest hit \textit{nearhit}), and one from the other class (the nearest miss \textit{nearmiss}). Then RELIEF calculates the relevance of each feature \(v\) by the rule found in Equation 2.1.

\[
W(v) = P(\text{different value of } v|\text{nearmiss}) - P(\text{different value of } v|\text{nearhit}) \tag{2.1}
\]

The justification for this rule is that if a feature is a good predictor, instances from different classes should have vastly different values, and instances of the same class should have very similar values. Unfortunately, the true probabilities cannot be calculated, so we must estimate the difference in Equation 2.1. This is done by calculating the distance between random instances and their nearest hits and misses. All distances are determined in one-dimensional subspaces hence the algorithm cannot cope with multidimensional statistical dependencies. Also, RELIEF is a randomized technique imposing ranking of features and thus it cannot handle correlated features very well. Pseudo code of RELIEF is shown in Algorithm 1.

\textbf{Algorithm 1} RELIEF (Kira & Rendell, 1992)

\begin{verbatim}
Initiate the weights vector to zero: \(w = 0\)
for \(t = 1, \ldots, T,\)
    Pick randomly an instance \(x\) from \(S\)
    for \(i = 1, \ldots, N\)
        \(i.w_i = w_i + (x_i - \text{nearmiss}(x)_i)^2 - (x_i - \text{nearhit}(x)_i)^2\)

The chosen feature set is \(\{i|w_i > \tau\}\) where \(\tau\) is a threshold
\end{verbatim}
2.4.2 Simba algorithm

In 2004, Gilad-Bachrach et al. [42] presented the algorithm Simba which embeds stochastic gradient ascent into slightly modified RELIEF. The difference distances between samples and their nearest neighbors are weighted by coefficients linked to the quality of features. Weights are found by maximizing the margin. Gilad Bachrach et al. show in experiments with synthetic and real-world datasets that Simba handles correlated features better than RELIEF. All distances are, however, determined only in one-dimensional subspaces again. Pseudo code of Simba is shown in Algorithm 2.

Algorithm 2 Simba (Gilad-Bachrach et al., 2004)

\[
\text{Initiate} \quad w = (1, 1, \cdots, 1) \\
\text{for } t = 1, \cdots, T, \\
\quad \text{Pick randomly an instance } x \text{ from } S \\
\quad \text{Calculate } \text{nearmiss}(x) \text{ and } \text{nearhit}(x) \text{ with respect to } S / \{x\} \text{ and the weight vector } w \\
\quad \text{for } i = 1, \cdots, N \\
\quad \quad \Delta_i = \frac{1}{2} \left( \frac{(x_i - \text{nearmiss}(x_i))^2}{\|x - \text{nearmiss}(x)\|_w} - \frac{(x_i - \text{nearhit}(x_i))^2}{\|x - \text{nearhit}(x)\|_w} \right) w_i \\
\quad \quad w = w + \Delta \\
\quad w \leftarrow w^2 / \|w^2\|_\infty \text{ where } (w^2)_i := (w_i)^2
\]

2.5 Feature Selection in Ensemble Methods

Feature selection methods discussed up to this point employ the use of a single classifier. An ensemble system, on the other hand, is composed of a set of multiple classifiers and performs classification by selecting from the predictions made by each of the classifiers [96]. Since wide research has shown that ensemble systems are often more accurate than any of the individual classifiers of the system alone [15, 56], it is only natural that ensemble systems and feature selection would be combined at some point.

2.5.1 Ensemble methods

The main goal of an ensemble is to construct multiple classifiers from the original data and then aggregate their predictions when classifying unknown instances. Figure 2.6 shows a basic view of an ensemble method [15]. As depicted, three main steps exist: training set generation, learning, and integration [96]. **Step 1** begins with the original training set \( D \). From this training set, \( t \) data subsets are created \( (D_1, D_2, \cdots, D_t) \). Then in **Step 2**, \( t \) base classifiers are generated \( (C_1, C_2, \cdots, C_t) \). These
classifiers may all be the same, all different, or contain any combination of the same or different classifiers. Each classifier $C_i$ is trained using the subset $D_i$. Finally in Step 3, the prediction of each classifier is combined in a predetermined way to produce the resulting classification. Bagging and Boosting are two common ensemble methods.

Two primary approaches exist to the integration phase: combination and selection. In the combination approach, the base classifiers produce their class predictions and the final outcome is composed using those predictions. In the selection approach, one of the classifiers is selected and the final prediction is the one produced by it [99]. The simplest and most common combination method is voting, also known as majority voting. In voting, the classification predicted by a base classifier is counted as a vote for that particular class value. The class value with the most votes becomes the final classification [99]. Although research is always ongoing on new approaches to increase diversity, some methods include training on different subsets of the training set, using different learning algorithms, injecting randomness, and training on different sets of input features. The latter is where ensemble feature selection can be successfully applied.

**Ensemble Feature Selection**

An effective way of generating a diverse, yet accurate, ensemble of base classifiers is to use ensemble feature selection [99]. To recall, theoretical and empirical research has shown that an efficient ensemble should consist not only of high accuracy classifiers, but classifiers which also errors in different parts of the input space [15]. Providing different feature subsets allow base classifiers to make their classification errors in different subareas of the instance space. While feature selection algorithms attempt to find an optimal feature subset for the learning algorithm, ensemble feature selection has the additional goal of finding a set of feature subsets which will model dependencies among the base classifiers [99]. This is perhaps the most important point in understanding the motivation and goals of the research presented in this work. Various successful attempts have been made at implementing ensemble feature selection. Here we focus on boosting based feature selection approaches.
2.5.2 Boosting-based feature selection techniques

Boosting is a general term for the performance enhancement. The principle of boosting is to effectively combine simple decision rules (so called weak classifiers) into strong ensembles of classifiers. This allows capturing even quite complex decision boundaries. The Adaboost algorithm (from adaptive boosting) is a powerful learning method with a built in feature selection mechanism. Adaboost was presented by Freund and Schapire in the original paper [37]. Lately, it has been shown effective in many applications like face recognition [90, 32], text mining [105] and intrusion detection [11, 44]. Adaboost is mostly used for selecting a small number of important features from a very high number of potential candidates and for building a final strong composite decision rule.

In the view of boosting as a feature selection, an interesting approach to use is boosting with decision stumps as the learning algorithm [106]. Constructing such a learner involves selecting a single feature, based on its ability to discriminate between classes [2]. In 2001 Das [30] proposed the BDSFS (Boosted Decision Stump Feature Selection) algorithm. Another boosting-based with decision stumps feature selection with greedy selection search strategy, termed Adaboost-CR, was recently proposed by Tsuchiya and Fujiyoshi [98].

We first give a short introduction about the decision stumps concept, then we describe in details the two boosting-based feature selection approaches mentioned above, since they are the most related studies to our work.

2.5.3 Decision stumps

A decision stump is a machine learning model consisting of a one-level decision tree [106] (see example in Figure 2.7). That is, it is a decision tree with one internal node (the root) which is immediately connected to the terminal nodes. A decision stump makes a prediction based on the value of just a single input feature. Depending on the type of the input feature, several variations are possible. For nominal features, one may build a stump which contains a leaf for each possible feature value or a stump with the two leaves, one of which corresponds to some chosen category, and the other leaf to all the other categories. For binary features these two schemes are identical. A missing value may be treated as a yet another category.

For continuous features, usually, some threshold feature value is selected, and the stump contains two leaves - for values below and above the threshold. However, rarely, multiple thresholds may be chosen and the stump therefore contains three or more leaves. Decision stumps are often used as

Figure 2.7: An example of a decision stump that discriminates between two of three classes of Iris flower data set: Iris versicolor and Iris virginica.
components (called “weak learners” or “base learners”) in machine learning ensemble techniques such as bagging and boosting. The term “decision stump” has been coined in a 1992 ICML paper by Wayne Iba and Pat Langley [2].

2.5.4 BDSFS algorithm

The algorithm uses boosted decision stumps as the weak learners. It uses the information gain criterion (IG) described in Equation 2.2 for deciding which feature to choose. The IG measures the amount of information in bits about the class prediction, if the only information available is the presence of a feature and the corresponding class distribution. Concretely, it measures the expected reduction in entropy (uncertainty associated with a random feature) [70]. Given $S$ the set of training examples, $x_i$ the vector of $i$th variables in this set, $|S_{x_i=v}|/|S|$ the fraction of examples of the $i$th variable having value $v$:

$$IG(S, x_i) = H(S) - \sum_{v=\text{values}(x_i)} |S_{x_i=v}| H(S_{x_i=v})$$

(2.2)

with entropy

$$H(S) = -p_+(S) \log_2 p_+(S) - p_-(S) \log_2 p_-(S)$$

(2.3)

Where $p_\pm(S)$ is the probability of a training example in the set $S$ to be of the positive/negative class. The boosting algorithm is run for $T$ rounds, and at each round all features that have previously been selected are ignored. Hence at each round it looks for the previously unselected feature with the highest information gain on the weighted distribution of training examples.

The search for features is greedy. If a feature is selected in any round, it becomes part of the set that will be returned. BDSFS is shown in Algorithm 3.

Algorithm 3 BDSFS (Das, 2001)

**Input:** $S$, Training dataset $(x_i, \ell_i)$

**Initialize:** Subset of surviving features $s = \{\}$

**Do for** Until all features are added

1. **Train** AdaBoost classifier with all the training examples
2. **Compute** IG
3. **Find** the best feature $\text{best} = \arg \max(\text{IG})$
4. **Add** the best feature $i$ that maximize IG

**Rank** features
2.5.5 AdaBoost-CR algorithm

Tsuchiya and Fujiyoshi has proposed a feature evaluation method based on AdaBoost with decision stumps [98]. They introduced a metric, called contribution ratio CR, that indicates how well the features “contribute” to the classification performance based on the performance weight $\alpha$ of the weak hypothesis $h_t$. We will refer to this method as AdaBoost-CR.

A contribution ratio $CR_v$ for each feature $v$ is defined by:

$$CR_v = \frac{1}{T} \sum_{t=1}^{T} \bar{\alpha}_t \delta_K[P(h_t) - v]$$  \hspace{1cm} (2.4)

where $\bar{\alpha}_t$ is the average confidence assigned to feature $v$, $\delta_K$ is the Kronecker delta; which is a function of two variables that is 1 if they are equal and 0 otherwise, and $P()$ is a function for outputting the feature chosen at round $t$ in the AdaBoost training process. In other words, the $CR_v$ is equal to the fraction of the absolute “confidence” weight associated with any feature $v$. Pseudo code of AdaBoost-CR is shown in Algorithm 4.

### Algorithm 4 AdaBoost-CR (Tsuchiya & Fujiyoshi, 2008)

**Input:** $S$, Training dataset $(x_i, \ell_i)$

**Initialize:** Subset of surviving features $s = [1, 2, \ldots, k]$

**Do for** Until $s$ is empty

- **Train** AdaBoost classifier with all the training examples
- **Compute** Contribution Rate $CR_1, CR_2, \ldots, CR_k$
- **Find** the worst feature $\text{worst} = \arg \min(CR_i)$
- **Remove** the worst feature $i$ that minimum $CR_i$

2.6 Limitations of Existing Methods

Utilizing the concept of hypothesis margins to measure the quality of a set of features is a growing line of research. The most discussed algorithms developed under the large hypothesis margin principles of the 1-NN algorithm are RELIEF and Simba.

One common problem with these previous studies is that they are filter based methods; they do not consider the interactions between the selected features— they evaluate each single feature once, independently of other features, and they ignore the features interaction with the classifier.
Clearly, any feature selection method that would attempt to consider all possible subsets of features would have an exponential run-time and would be intractable. Thus, alternative greedy subset search methods have been proposed, such as sequential forward/backward search which is still computationally prohibitive with $O(d^2)$, where $d$ is the number of features.

Moreover, to our knowledge, almost no previous work has exploited the characteristics of the hypothesis margins of boosting to determine the quality of features. The two main related works are BDSFS and AdaBoost-CR. In BDSFS, the original Adaboost is modified in such a way that features attached to weak classifiers are gradually removed from the set of all available measurements. Apparently, the removed features cannot be used for further learning which limits the flexibility to create new weak classifiers. The final composite decision rule is therefore not as strong as it would be in the original Adaboost classifier. The resulting feature subset is formed from the features picked for the output decision rule. Das [30] claims that features selected in this way are effective with other types of learning algorithms. However, this is not generally true since the variant of the Adaboost algorithm is very likely to select highly correlated features due to its greedy strategy and thus it is not very effective. In the AdaBoost-CR algorithm the features are evaluated based on the CR criterion. In our work, however, we show that the dynamics inherent to boosting offer ideal means to evaluate features utilizing the training examples’ margins distribution, and while a feature may have a large CR; it will not contribute to a good overall margin unless its “conditional” margin is also large. Thus, a better indicator is its fraction of the overall margin.

### 2.7 Learning from Imbalanced Data

In a number of instances, there are applications possessing imbalanced data sets – this class of imbalance presents problems for classical machine learning and data mining algorithms. The data imbalance problem has begun to attract significant attention over the last ten years. The class imbalance problem typically occurs when there are many more instances of a subset of classes than others. In many applications, the ratio of the small classes to the larger classes can be dramatic (e.g., 1-to-100, 1-to-1,000, or 1-to-10,000), and sometimes even larger. In such cases, standard classifier performance can be hindered, since classification tends to be overwhelmed by the majority classes and ignores minority classes. Accordingly, the overall classification rate is not appropriate to evaluate the performance. The class imbalance problem is prevalent in many applications, including: fraud/intrusion detection, risk management, text classification, and medical diagnosis/monitoring.

The current approaches to handle class imbalance have been proposed at either the algorithm level or at the data level as discussed in two workshops at the AAAI conference [50] and the ICML conference [23], and later in the sixth issue of SIGKDD Exploration (see, for example, a review by Weiss [104]). When working at an algorithm level, solutions involve introducing a cost function associated with each class. This can begin to resolve some of the problems encountered with class
imbalance [33], such as adjusting the probabilistic estimate at the tree leaf when working with decision trees.

At the data level, these solutions include many different forms of re-sampling, which involve normalizing the classes so that they are no longer imbalanced. Typically, by under-sampling the larger class [57], by over-sampling the smaller class [58], or by a combination of these techniques [35], the data imbalance problem can be overcome.

Many real-world imbalanced data sets are often accompanied by another problem: high dimensionality. For example, in molecular classification of cancer [109], the number of genes (corresponding to the features of instances) under consideration is extremely large (e.g., over 10,000), while only a small subset of genes is found to be responsible for causing cancer. Finding features that capture the high skew in the class distribution is important. Indeed, solutions proposed for dealing with imbalanced data may not work well for high dimensional data. Forman observed that in highly imbalanced data classification problems, feature selection is even more important than classification algorithms [36].

2.7.1 Feature selection on imbalanced data

Feature selection’s importance to resolving the class imbalance problem is a recent development with most research appearing in the previous seven years. In this time period, a number of researchers have conducted research on using feature selection to combat the class imbalance problem. Mladenić and Grobelnik [72] studied the performance of feature selection metrics in classifying text data drawn from the Yahoo web hierarchy. They applied nine different metrics to the data set and measured the power of the best features using the naive Bayes classifier. Their results showed that the odds ratio (OR) was nearly universally the best performing metric on the F-measure with $\beta = 1$ and $2$, precision, and recall measures; in general, they concluded that the best metrics choose common features and consider the domain and learning method’s characteristics. This explains why OR would be very good for the naive Bayes classifier: it tries to maximize the posterior probability of a sample being in a class. Forman [36] examined the classification power of feature selection metrics on a number of text sets drawn from the TREC competitions, MEDLINE, OHSUMED, and Reuters databases. He used the features selected by different metrics to train linear SVMs and evaluated their performance using accuracy, precision, recall, and the F1-measure. While a small number of metrics saw mild improvements over the performance of an SVM using all of the available features, the best
overall performance came from features selected by bi-normal separation (BNS). Forman’s general finding is that the metrics that performed best selected features that separated the minority class from the majority class well. He also concluded that the evenly balanced form of BNS that selected features with equal weight to true positive rate and false positive rate gave the best performance. Zheng, Wu, and Srihari [112] analyzed how the types of selected features affected classification performance. Positive features are those that indicate membership in a class, and negative features are those that indicate lack of membership in a class. They used the chi-square statistic (CHI), IG, and OR algorithms as a starting point to create one-sided and two-sided metrics; one-sided metrics only select positive features on their score, and two-sided metrics select both positive and negative features based on the absolute value of their score. They then compared the performance of the one and two-sided metrics with different ratios of the best positive and negative features. The ratio selection method resulted in improved performance on the break-even point between precision and recall compared to general one and two-sided metrics. Thus, both positive and negative features are important to obtain the best possible power from a classifier. Recently, Chen [24] proposed the FAST metric, which evaluates features using an approximation to the area under the ROC curve (AUC). Both linear SVM and nearest neighbor classifiers using FAST-selected features saw improved AUC and balanced error rates over the baseline score using all features and the scores achieved using RELIEF and Pearson correlation coefficient (PCC)-selected features [24].

2.7.2 FAST algorithm

FAST (Feature Assessment by Sliding Thresholds) is a recent feature selection metric proposed by Chen [24]. This method is based on computing the area under a ROC curve, which is determined by training a simple linear classifier on each feature and sliding the decision boundary for optimal classification. The samples are classified on multiple thresholds. Statistics are gathered to characterize a classifier’s performance at each boundary. If the true positive rate and false positive rate are calculated at each threshold, then an ROC curve can be built, and thus, the AUC can be computed. Since the area under the ROC curve is a strong predictor of performance (especially for imbalanced data classification problems), this score can be used rank features. To determine where best to place thresholds: a modified histogram or an even-bin distribution is used. One problem with this approach is that it evaluates features independently, missing an intercorrelation between features. Pseudocode for the FAST is described in Algorithm 5.

2.7.3 Limitations of existing methods

One common problem with many of the standard feature selection metrics used in the previous studies is that they operate on Boolean data. Some metrics, such as IG and CHI, generalize well to nominal data, but they break down when handling continuous data. Thus, when one uses a discrete or Boolean feature selection metric on continuous data, its performance is entirely dependent on the
Algorithm 5 FAST (Chen & Wasikowski, 2008)

Input: \( k \): number of bins, \( m \): number of samples, \( n \): number of features, \( p \) is number of positive, \( q \) is number of negative

Split: 0 to \( m \) with \( s \) step size \( m/k \)

\begin{algorithm}
\begin{algorithmic}
\For {\( i = 1 \) to \( n \)}
  \State \( X \) = Vector of samples’ values for feature \( j \)
  \State Sort \( X \)
  \For {\( j = 1 \) to \( k \)}
    \State Bottom = \text{round}(\text{Split}(j))+1
    \State Top = \text{round}(\text{Split}(j+1))
    \State M = \text{mean}(X(\text{bottom to Top}))
    \State Classify \( X \) using \( M \) as threshold
    \State tpr(i,j) = \( tp/p \)
    \State fpr(i,j) = \( fp/q \)
  \EndFor
\EndFor

\textbf{Calculate} the area under ROC by tpr and fpr
\end{algorithmic}
\end{algorithm}

choice of the preset threshold used to binarize the data. This threshold determines the confusion matrix’s true positive (tp), false negative (fn), false positive (fp), and true negative (tn) counts. Consider a classification problem using two different features sets such that a classifier using the first feature set yields a higher tp count and lower tn count than the second set. However, if we bias the classifier to change its decision threshold, that classifier may instead have a higher tp count and lower tn count for the second feature set. Thus, it is impossible to tell which feature set is better because one threshold will not give us adequate information about the classifier’s performance. This happens because the confusion matrix is an evaluation statistic of a classifier’s predictive power based solely on this one threshold.

This can be seen if we plot the receiver operating characteristic (ROC) associated with each feature set.

It may be advantageous to use a feature selection metric that is non-parametric and could use all possible confusion matrices when using ordinal or ratio data. Thus, it would be possible to find the threshold that result in the highest performance possible. One possibility is the ROC. This is a non-parametric measure of a system’s power that compares the true positive rate with the false positive rate. We can then quantify the ROC using the area under the ROC curve (AUC) and apply this score as our feature ranking. The FAST metric evaluates features using an approximation to
the AUC. However, FAST is a filter based method, and thus it does not consider the interactions between the selected features and the features interaction with the classifier.

2.8 Comparison of Feature Selection Algorithms

Ground truth regarding the quality of the features is not available in practice. Feature selection algorithms are therefore assessed based on their execution time and on the size and quality of the selected feature subsets. Experimental methodology for the results evaluation is however not standardized in the literature and differs from paper to paper. Moreover, the results are often provided without information about their reliability. It is therefore very difficult to draw any conclusions or to make any comparison between feature selection methods proposed or examined over various research papers.

2.8.1 Performance evaluation of the selected feature subsets

Feature selection is typically a single purpose task performed in an on-line manner. It has been argued that the execution time is therefore not as critical property as the optimality the solution. While this is true for data of moderate dimensionality, several recent applications (e.g., microarray) involve several thousands of features. In such cases, the computational requirements of feature selection methods may become very important.

The most common performance measure in the statistical pattern recognition is the probability of the classification error of a decision rule, called also the error rate. However, the error rate is only a single measure of the performance. Furthermore, the error rate assumes an equal misclassification cost (i.e., zero-one loss function) and a relatively uniform class distribution (i.e., balanced a priori class probabilities). Nevertheless, it is a simple and straightforward criterion. There are several more measures of discriminability, e.g., ROC curve and its area under the curve, AUC, and the balanced error (BER).

Also, to judge the significance of the differences found in the results, a pair t-test can be run on the output accuracies both before feature selection and then again applying each feature selection algorithm. The following sections give brief reviews of ROC, BER and t-test.

Receiver Operating Characteristic (ROC)

It’s a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (tp = true positive rate) vs. the fraction of false positives out of the negatives (fp = false positive rate), at various threshold settings. tp is also known as sensitivity, and fp is one minus the specificity or true negative rate.
ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. Example of ROC curve is shown in Figure 2.9.

One ROC is said to dominate another if the second ROC is less than or equal to the first for every point; a dominating ROC is better than the other in all cases, and we would prefer the use of the dominating classifier.

BER

On extremely imbalanced data sets, algorithms will be hard pressed to classify test samples as members of the minority class because the discriminant scores given by the classifier are often biased toward the majority class. Standard accuracy and error statistics are clearly poor measures of the performance of a classifier on imbalanced data, since a trivial majority classifier can give good results [36]. An ideal classifier should perform well on both the minority and the majority class. One of the statistics that researchers commonly use is BER which assesses balanced error rate with respect to both positive and negative examples [73]. BER is defined as the average of the errors on each class, and is computed in Equation 2.5.

\[
BER = \frac{1}{2} \left( \frac{fp}{fp + tp} + \frac{fn}{fn + tn} \right)
\]  

(2.5)

t-test

is any statistical hypothesis test in which the test statistic follows a Student’s t distribution if the null hypothesis is supported. It is most commonly applied when the test statistic would follow a normal distribution if the value of a scaling term in the test statistic were known. When the scaling term is unknown and is replaced by an estimate based on the data, the test statistic (under certain conditions) follows a Student’s t distribution.

2.8.2 Re-sampling techniques for the performance evaluation

Here we briefly discuss only the more popular data re-sampling approaches for the performance evaluation. The major problem is how to make an efficient use of the data as only a limited sample size is available. Of course, a sufficient number of samples have to be employed in a classifier design.
and accuracy. Moreover, the sampling procedure should keep the probability distributions of the training and test data close to the original problem. A common way is to run Cross Validation.

Cross validation

Cross-validation, sometimes called rotation estimation, is a technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds. Cross-validation is important in guarding against testing hypotheses suggested by the data (called “Type III errors”), especially where further samples are hazardous, costly or impossible to collect (see uncomfortable science).

One type of cross validation is \(k\)-fold cross validation— in which the data set is divided into \(k\) subsets. Each time, one of the \(k\) subsets is used as the test set and the other \(k-1\) subsets are put together to form a training set, see Figure 2.10. Then the average error across all \(k\) trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set \(k-1\) times. The variance of the resulting estimate is reduced as \(k\) is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch \(k\) times, which means it takes \(k\) times as much computation to make an evaluation. A variant of this method is to randomly divide the data into a test and training set \(k\) different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over.

Data split stratification

Stratification is a supporting data sampling procedure that tries to keep approximately the same portions of class labels in the training and test data as in the original data set. An argument for doing so is such that training and test data keep (hopefully) similar statistical properties like the original problem in this way (e.g., class priors). A classifier designed and tested on stratified data is therefore expected to produce more accurate results. Kohavi [54] experimentally showed that a
stratified $k$-fold cross-validation seems to have a lower variance and to be less biased compared to standard randomized approaches. For demonstration, he chose several problems originating from the UCI [77]. Because there are no ground truth results about the real performance, Kohavi utilized repeated 50/50 holdout validation with 500 trials as a reference standard. Notice that although such an estimate has a low variance, the achieved performance might still be biased. Thus, it cannot tell us much about the real bias from the ground truth. Nevertheless, Kohavi recommended to use stratified ten-fold cross-validation for the error rate estimation and model selection but hypothesized that repeated cross-validation might be a better estimation scheme.

2.8.3 Benchmark data sets

Majority of publicly available data repositories are not very well documented and there is also no feedback from domain experts on the achieved results. Data often include multi-class problems, mixed definition domains of individual features, missing values, and usually need some preprocessing. Ground truth regarding the quality of features is of course not available in practice. Further problem is mainly the small sample size. Some data may also contain a very few if any redundant or irrelevant features, which leaves a limited space for improvement using the feature selection. All these issues make the eventual design and further analysis of feature selection methods rather difficult. An introduction to feature selection issues and some useful internet links to publicly available benchmark data can be found in [107].

The most widely used benchmark data in the feature selection research are several artificial and real-world problems originating from the University of California at Irvine, known as the UCI data repository [77]. Original data are text files of a different format. We recommend to use their WEKA [107] version, where the data format is standardized.

2.9 Chapter Conclusions

In this chapter, we organized the state-of-the-art in feature selection in order to provide a quick and compact overview for the reader. We discussed basic search strategies applied in feature selection as well as methods which are independent or dependent on a classifier (i.e., filters, wrappers, and embedded methods). Next, we provided in depth description of the use of feature selection in combination with boosting algorithm, as well as, a review of the most discussed hypothesis margin based methods. We then, provided a detailed description of the class imbalance problem, and the issues arising when underlying model is trained with high-dimensional imbalanced data. Finally, the evaluation methodology applied in feature selection was briefly described including the most common benchmark data.

Utilizing the concept of hypothesis margins to measure the quality of a set of features is a growing line of research. The most discussed algorithms developed under the large hypothesis
margin principles of the 1-NN algorithm are RELIEF and Simba. These algorithms are filter based methods, and thus, they ignore features dependencies and their interaction with the classifier.

To our knowledge, almost no previous work has exploited the characteristics of the hypothesis margins of boosting to determine the quality of features. The two main related work, particularly in boosting with decision stumps with a greedy search i.e., embedded based methods, are: 1) BDSFS algorithm by Das, in which he applies a forward selection search strategy. The selection of the next feature to be considered is based on the information gain criterion and takes into consideration the weight of each dataset instance, and 2) Tsuchiya and Fujiyoshi [98] algorithm. In their work the features are evaluated based on the contribution ratio (CR) criterion. The CR is defined as the relative importance of features based on the confidence ratio of the learned base hypothesis. Tsuchiya and Fujiyoshi algorithm starts with a given set of features, the CR of each features from a feature set is estimated, and features that are not contributing and that have a low CR are removed.

Moreover, the class imbalance problem is a recent development in machine learning. It is frequently encountered when using a classifier to generalize on real-world application data sets, and it causes a classifier to perform sub-optimally. Researchers have rigorously studied re-sampling methods, new algorithms, and feature selection methods, but no studies have been conducted to understand how well these methods combat the class imbalance problem. In particular, feature selection has been rarely studied outside of text classification problems. Additionally, no studies have looked at the additional problem of learning from small samples.

In this work, however, we show that the dynamics inherent to boosting offer ideal means to evaluate features utilizing the training examples’ margins distribution. Our algorithm is an embedded based approach, which is capable of modeling features dependencies in a reasonable computational complexity. In addition, we show how the proposed method can be extended to effectively select informative features from small data possessing skewed class distributions.
Chapter 3

Feature Weighting and Selection Using Hypothesis Margin of Boosting

In this chapter, we propose two embedded-based feature selection techniques. The feature selection methods introduced in this chapter are inspired by the boosting algorithm. In particular, our approaches are based on the concept of the hypothesis margin induced by boosting. In the first algorithm, a new weight metric, termed Margin Fraction (MF), is proposed. It defines the quality of a set of features based on their contribution to the average margin produced. In the second algorithm, we extend this methodology to define another weight metric, to handle the presence of data with imbalance class distributions, by considering the quality of features based on the maximized AUC margin induced.

Our algorithms have the advantage that they include the interaction with the classification model, i.e., with boosting algorithm, while at the same time being far less computationally intensive. In addition, they complement current boosting-based feature selection techniques by measuring the relative importance of features based on their contribution to the overall margin induced, rather than to their contribution to the confidence ratio of the learned base hypothesis.

This chapter is organized as follows. We first explain the concept of margin and its importance to feature selection. We then, review the idea of AdaBoost algorithm and its hypothesis margin notion. Next, we present the mathematical theory of the proposed feature selection algorithms. Finally, we summarize our work.
CHAPTER 3. HYPOTHESIS WEIGHTING FEATURE SELECTION

3.1 Margin Concept

Margin based feature selection is a growing line of research. There are two main ways to define margins [28]: the sample-margin (SM), and the hypothesis-margin (HM).

The SM measures the distance between the instance and the decision boundary induced by the classifier. Support Vector Machine algorithm (SVM), for example, finds the separating hyper-plane with the largest SM. On the other hand, the HM requires the existence of a distance measure on the hypothesis class. Boosting and 1-NN use the concept of HM. Figure 3.1 shows examples of SM and HM margins.

Various feature selection algorithms have been developed under the large margin (SM or HM) principles such as SVM-based feature selection [43] and RELIEF family (1-NN based) algorithms, such as Simba [42]. However, although boosting algorithm has attracted much attention in the machine learning community, almost no previous work has exploited the characteristics of the hypothesis margins of boosting to determine the quality of features.

The idea of AdaBoost, Freund and Schapire’s most successful boosting algorithm [37, 38], is to linearly combine a number of simple weak classifiers into a single strong classifier. The major advantage of AdaBoost algorithm is the adaptive selection of discriminative and complementary features during the training process which most often yields to better feature or variable selection while keeping or even increasing the prediction accuracy.

The feature selection methods proposed in the next sections use the concept of the Adaboost
CHAPTER 3. HYPOTHESIS WEIGHTING FEATURE SELECTION

algorithm. We therefore consider it appropriate to introduce the Adaboost algorithm first and show how it can be applied for the feature selection task.

3.2 Notation

We first briefly recall the notation that will be used. Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a definition domain of the input set of features, where $n \in \mathbb{N}$ indicates the dimensionality of the problem. Let $\mathcal{L} = \{1, -1\}$ be a set of admissible class labels. Suppose that the training data is given as a finite set $S = \{(x_i, \ell_i)\}_{i=1}^m$ of $m \in \mathbb{N}$ labelled examples, where $i = 1, \ldots, m$ lists the available examples, and $m = p + q$, where and $p$ and $q$ are numbers of negative and positive samples, respectively. Values $x_i \in \mathcal{X}$ are individual sample vectors with corresponding class labels $\ell_i \in \mathcal{L}$. Data for processing is stored in an $n \times m$ pattern matrix $X = [x_1, x_2, \ldots, x_m]$, where $i = 1, \ldots, m$ are rows and $j = 1, \ldots, n$ are columns of the matrix. Each column represents an individual sample vector and each row corresponds to a particular measurement. The labels of the samples are stored in the row vector $\ell = [\ell_1, \ell_2, \ldots, \ell_m]$. The components of a sample vector $x_i$ are denoted as $v_i$ and we refer to them as features. Let $\Upsilon = \{v_1, v_2, \ldots, v_n\}$ be the feature set.

3.3 AdaBoost Algorithm

Definition 3.3.1 Let $S$ be a set of $m$ instances drawn i.i.d. from $\mathcal{D}$, and let $\mathcal{H}$ be a hypothesis space (in this paper we constrain $\mathcal{H}$ to be finite).

AdaBoost calls a given weak or base learning algorithm repeatedly in a series of rounds $t = 1, \ldots, T$, where $T \in \mathbb{N}$. A base learner $h \in \mathcal{H}$ job is to find a weak hypothesis $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ appropriate for the distribution $\mathcal{D}_t$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on training example $i$ on round $t$ is denoted $\mathcal{D}_t(i)$. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. The strength of the weak hypothesis is measured by the formula

$$\epsilon_t = \sum_{i: h_t(x_i) \neq \ell_i} \mathcal{D}_t(i)$$

(3.1)

which stands for the probability of misclassification of samples from the training set. Weak hypotheses are constructed in such a way that the misclassification error in Equation 3.1 is gradually minimized.

After a hypothesis is received, the algorithm updates the weights of the distributions $\mathcal{D}_t$ by this formula:
CHAPTER 3. HYPOTHESIS WEIGHTING FEATURE SELECTION

Algorithm 6 AdaBoost Algorithm (Freund & Schapire, 1996)

**Input:** Given a labeled training set \( S = \{(x_i, \ell_i)\}_{i=1}^m \), where \( x_i \in \mathcal{X} \) are samples, \( \ell_i \in \{1, -1\} \) are labels. Let \( T \in \mathbb{N} \) be the number of weak classifiers to combine.

**Initialize:** \( D_1(i) = 1/m \) for all \( i = 1, \ldots, m \)

for \( t = 1, \ldots, T \) do

\( \mathcal{H} = h_t : \mathcal{X} \to \{-1, 1\} \)

\( h_t^* = \arg\min_{h_t \in \mathcal{H}} \epsilon_t \)

\( \epsilon_t^* = \min_{h_t \in \mathcal{H}} \epsilon_t \) where \( \epsilon_t = \sum_{i : h_t(x_i) \neq \ell_i} D_t(i) \)

\( \alpha_t^* = \frac{1}{2} \log \left( \frac{1 - \epsilon_t^*}{\epsilon_t^*} \right) \)

\( D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t^* \ell_i h_t^*(x_i))}{Z_t} \)

where \( Z_t \) is a normalization constant, and \( \sum_{i=1}^m D_{t+1}(i) = 1 \)

**Output:** \( H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t^*(x) \right) \)

\[
\mathcal{D}_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t \ell_i h_t(x_i))}{Z_t} \tag{3.2}
\]

and

\[
\alpha_t = \frac{1}{2} \log \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{3.3}
\]

Where \( \alpha_t \) is the voting coefficient (confidence) assigned to the hypothesis \( h_t \). Note that as \( \epsilon \) decreases; alpha increases, and as such, one can view \( \alpha \) a measure of how much one should “trust” the corresponding weak hypothesis \( h_t \). As \( \alpha \) increases, misclassified examples have their weights increased; correctly classified examples have their weights decreased. Thus, boosting concentrates on the “difficult” examples in subsequent rounds.

Combining the hypotheses generated by these weak learners will collectively produce the following weighted linear classifier:

\[
H(x_i) = \sum_{t=1}^T \alpha_t h_t(x_i) \tag{3.4}
\]

A pseudo-code for the discrete version of the Adaboost algorithm is shown in Algorithm 6.

In boosting the weak classifiers are constructed in a greedy way by minimizing an exponential loss function of a certain type of a margin. In each training step, weights of samples that were incorrectly classified by the generated weak classifier are increased. Hence, the Adaboost algorithm concentrates on the samples that are difficult to discriminate. Such learning process leads to a
gradual minimization of an upper bound on the training error. Deriving weak classifiers in high
dimensional spaces is a non-trivial task. Therefore, weak learning algorithms are usually based on
scalar features, i.e., projections of sample vectors into their respective coordinates. This suggests
that Adaboost could be applied jointly for the feature selection and classifier design as it is shown,
for example, in [11, 30, 98].

3.3.1 AdaBoost margin

Boosting is particularly good at finding hypotheses with large margins, in which it concentrates on
those examples whose margins are small (or negative) and forces the base learning algorithm to
generate good classifications for those examples [37].

The margin of AdaBoost at $T$ rounds associated with any instance $i$ is defined as

$$
\rho(x_i) = \frac{\ell_i H(x_i)}{\omega} = \frac{\ell_i \sum_{t=1}^{T} \alpha_t h_t(x_i)}{\omega}
$$

while $\omega$ served as a normalization factor and is defined as $\omega = \sum_{t=1}^{T} \alpha_t$.

It is easy to see that the margin is a number in the range $[-1, 1]$, and that an example is classified
correctly if and only if its margin is positive i.e., $H(x)$ classifies the example correctly as shown in
Figures 3.2, reffig:AdaboostHxT10, reffig:AdaboostHxT50 and reffig:AdaboostHxT100. A large
positive margin can be interpreted as a “confident” correct classification. The distribution of the
margin can be visualized by plotting the cumulative distribution function (CDF) of margins, i.e.,
the fraction of examples whose margin is at most $c$ as a function of $c \in [-1, 1]$. Figure 3.6 shows
how the CDF of margins of the training instances changes quite significantly after 5, 10, 50 and 100
rounds.

3.3.2 AdaBoost and overfitting

AdaBoost has the property that it does not often seem to suffer from overfitting, even after a large
number of iterations [20, 84]. To understand this lack of overfitting, Breiman [20] first used the notion
of variance and bias for classification to argue that AdaBoost could avoid overfitting by reducing
variance, since it is in ways similar to bagging [19]. Schapire et. al. [87], however, explained to some
extent a reasonable explanation to the success of AdaBoost by the margin theory.

The margin of a boosted classifier is a number between $-1$ and $1$ as shown in Figure 3.7, that
according to the margin theory, can be thought of as a confidence measure of a classifier’s predictive
ability, or as a guarantee on the generalization performance. If the margin of a classifier is large,
then it tends to perform well on test data. If the margin is small, then the classifier tends not to
perform so well. Furthermore, Schapire et al. showed that AdaBoost has a tendency to increase the margins on the training examples. Thus, though not entirely complete, their theory and experiments strongly supported the notion that margins are highly relevant to the behavior and generalization performance of AdaBoost.

Breiman [20], however, soon thereafter raised serious doubts on the margin theory by designing a boosting-type algorithm called arc-gv. Breiman’s experiments indicated that his algorithm achieved higher margins than AdaBoost, and yet performed worse on test data.

Reyzin and Schapire [85] reproduced Breiman’s experiments and were able to reconcile his results with the margins explanation, noting that the weak classifiers found by arc-gv are more complex than those found by AdaBoost. Although the empirical success of a AdaBoost algorithm depends on many factors (e.g., the type of data and how noisy it is, the capacity of the weak learning algorithm, the number of boosting iterations, regularization and the entire margin distribution over the training examples), it is well accepted that margin distribution is crucial to relate margin to the generalization of AdaBoost.

Previously the minimum margin bound was established for AdaBoost, however, researchers believe that this is far from sufficient. Intuitively, Reyzin and Schapire [85] suggested to use the average margin as a measure to compare margin distributions, but provided no bound prove for it; this was the focus of Gao and Zhou [40] work who recently proved the average margin bound for AdaBoost and showed that a larger average margin implies stronger generalization.

### 3.3.3 Average margin bound

AdaBoost effectively maximizes the minimum margin:

$$\rho^* = \max_\alpha \min_{i \in [1, \ldots, m]} \ell H(x)$$  \hspace{1cm} (3.7)

Which leads to good generalization ability as Schapire et al., proved in Theorem 1 [87].

However, recently, Gao and Zhou [40] show that compared to previous statistics on margin theory (i.e., minimum margin), the average margin is one of the statistics that considers the whole margin distribution and thus includes more information.

The average margin, $E_S[\ell H(x)]$ across $m$ examples, can be defined as:
\[ \rho = \frac{1}{m} \sum_{i=1}^{m} \rho(x_i) \]  
\[ \rho = \frac{\sum_{i=1}^{m} \ell_i H(x_i)}{m \omega} \]  
\[ \rho = \frac{\sum_{i=1}^{m} \ell_i \left( \sum_{t=1}^{T} \alpha_t h_t(x_i) \right)}{m \sum_{t=1}^{T} |\alpha_t|} \]  
\[ \rho = \frac{\sum_{i=1}^{m} \ell_i \alpha_t h_t(x_i)}{m \sum_{i=1}^{m} |\alpha_t|} \]

Gao and Zhou [40] provided an upper bound for the generalization error of AdaBoost in term of average margin by Theorem 6. Which we briefly describe as follows.

**Theorem 3.3.1** For constant \( \gamma > 0 \), suppose base learner \( h_t \) in each iteration has edge \( \gamma_t \geq \gamma \) and set

\[ \tau = \frac{-0.99 \ln(1 - \gamma^2)}{\ln(1 + \gamma) - \ln(1 - \gamma)} \]

For any \( \delta > 0 \), if \( \theta = E_S[\ell H(x)] > \sqrt{8/|H|} \) and the iteration number

\[ T \geq \left\lceil \frac{100}{\ln(1 - \gamma^2)} \ln \left( \frac{1}{m} \left( \frac{16 \ln 2 |H|}{\tau^2 \theta^2} \ln \frac{2m^2}{\ln |H|} + \ln \frac{|H|}{\delta} \right) \right) \right\rceil \]

then with probability at least \( 1 - \delta \) over the random choice of sample \( S \) with size \( m \), every voting classifier \( H(x) \) by AdaBoost satisfies the following bound:

\[ \text{Pr}_D[\ell H(x) < 0] \leq \left( \frac{\ln |H|}{m} \right) \]

\[ + \sqrt{\frac{8}{m} \left( \frac{8 \ln 2 |H|}{\tau^2 \theta^2} \ln \frac{2m^2}{\ln |H|} + \ln \frac{|H|}{\delta} \right)} \]

Here \( \gamma_t = \sum_{i=1}^{m} D_t(i) \ell_i h_t(x_i) \) is called the edge of \( h_t \), which is an affine transformation of the error rate \( \epsilon_t \) of \( h_t(x_i) \).

The Theorem states that the generalization of AdaBoost depends on: 1) sample size, 2) complexity of the base learner, 3) average margin, and 4) number of iterations. All these factors could affect the generalization error, and thus, completely explaining AdaBoost’s resistance to overfitting is more difficult than what has been disclosed by previous theoretical results. In this sense, we would rather use average margin than other statistics to evaluate the quality of features.
Figure 3.2: Boosting $H(x)$, $\rho(x)$ and CDF values for training instances for Prostate dataset after 5 rounds.
Figure 3.3: Boosting $H(x)$, $\rho(x)$ and CDF values for training instances for Prostate dataset after 10 rounds.
Figure 3.4: Boosting $H(x)$, $\rho(x)$ and CDF values for training instances for Prostate dataset after 50 rounds.
Figure 3.5: Boosting $H(x)$, $\rho(x)$ and CDF values for training instances for Prostate dataset after 100 rounds.
CHAPTER 3. HYPOTHESIS WEIGHTING FEATURE SELECTION

Figure 3.6: The cumulative distribution function (CDF) of margins with different rounds using Prostate dataset.

Figure 3.7: The margin of a boosted classifier is a number between $-1$ and 1.
3.4 Margin Fraction for Feature Weighting and Selection

In this section, we present our first feature selection method based on the concept of the average margin induced by boosting. Our method observes the training examples’ mean margins to evaluate the quality of features. A weight criterion, termed Margin Fraction (MF), is assigned to each feature that contributes to the average margin combined in the final output produced by boosting.

To perform feature selection we consider features as the weak learners for boosting. The choice of the weak learner is usually driven by optimizing the prediction performance. In addition, some structural properties can be another useful criterion as well. AdaBoost estimator is a linear combination of weak learners. Therefore, structural properties of the boosting function estimator are given by linear combination of structural characteristics of the weak learner.

3.4.1 Stumps and larger trees as weak learners

Trees are among the most popular base procedures in machine learning. They have the advantage to be invariant under monotone transformations of predictor variables, i.e., we do not need to search for good data transformations. When using stumps [37, 106], i.e., a tree with two terminal nodes, the boosting estimate will be an additive model in the original predictor variables, because every stump-estimate is a function of a single predictor variable only. Similarly, boosting trees with (at most) \( d + 1 \) terminal nodes results in a nonparametric model having at most interactions of order \( d - 1 \): e.g., for \( d = 2 \), we would pick up interaction terms between pairs of predictor variables. Thus, if we want to constrain the degree of interactions, we can easily do this by constraining the (maximal) number of nodes in the tree learner. For many real datasets, it seems that low-order interaction (or even additive) models are sufficiently rich for good prediction and interpretation. For example, the naive Bayes classifier works surprisingly well in many applications [47]. Also boosting with stumps, yielding an additive model, has proven to be successful in many areas, e.g. winning the performance prediction challenge of the IEEE World Congress on Computational Intelligence 2006 [66]. Thus, we often get good performance with trees having 2 or 3 terminal nodes (\( d = 1 \) or 2, respectively). With such small values of \( d \), our proposed feature selection is computationally fast.

Therefore, we construct a set of weak classifiers by considering decision stumps. The properties of individual features are assessed indirectly by simple thresholding in one-dimensional subspace. Let

\[
h_{\theta}(x) = \begin{cases} 
+1, & \text{if } x \geq \theta \\
-1, & \text{otherwise}
\end{cases}
\]

where \( \theta \in \mathbb{R} \) is a threshold and \( x \in \mathbb{R} \) is a sample value corresponding to one of the features.
3.4.2 Margin fraction weight (MF)

**Definition 3.4.1** Let \( F \) be the total number of unique features used across all \( T \) rounds, i.e., decision stumps, and for any chosen feature \( v \), let \( h_{(v,j)} \) be the decision stump corresponds to the \( j \)-th use of feature \( v \), and let \( \alpha_{(v,j)} \) be the associated voting confidence. Let \( N_v \) be the total number of times that feature \( v \) is used.

We then have

\[
\sum_{v=1}^{F} N_v = T \tag{3.10}
\]

We can now redefine \( H(x) \) and \( \rho(x) \) as follows:

\[
H(x_i) = \sum_{i=1}^{F} \sum_{v=1}^{N_v} \alpha_{(v,j)} h_{(v,j)}(x_i) \tag{3.11}
\]

\[
\rho(x_i) = \frac{\ell_i \sum_{v=1}^{F} \sum_{j=1}^{N_v} \alpha_{(v,j)} h_{(v,j)}(x_i)}{\sum_{t=1}^{T} |\alpha_t|} \tag{3.12}
\]

Now for any individual feature \( v \), one can consider the weighted linear combination associated with that feature and the “conditional” margin associated with just that weighted linear combination for any instance \( i \).

\[
H_v(x_i) = \sum_{j=1}^{N_v} \alpha_{(v,j)} h_{(v,j)}(x_i) \tag{3.13}
\]

\[
\rho_v(x_i) = \frac{\ell_i \sum_{j=1}^{N_v} \alpha_{(v,j)} h_{(v,j)}(x_i)}{\sum_{j=1}^{N_v} |\alpha_{(v,j)}|} \tag{3.14}
\]

Consider the fraction of the absolute “confidence” weight associated with any feature \( v \), defined as follows:

\[
\psi_v = \frac{\sum_{j=1}^{N_v} \alpha_{(v,j)}}{\sum_{t=1}^{T} |\alpha_t|} = \frac{\sum_{j=1}^{N_v} \alpha_{(v,j)}}{\sum_{v=1}^{F} \sum_{j=1}^{N_v} |\alpha_{(v,j)}|} \tag{3.15}
\]

We then have the following theorem.

**Theorem 3.4.1** The overall margin associated with any instance “i” is the weighted linear combination of conditional margins, where \( \psi_v \) are the weights.

\[
\sum_{v=1}^{F} \psi_v \rho_v(x_i) = \rho(x_i)
\]
Proof

\[
\sum_{\nu=1}^{F} \psi_{\nu} \rho_{\nu}(x_{i}) = \sum_{\nu=1}^{F} \left( \frac{\sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)}}{\sum_{t=1}^{T} |\alpha_{t}|} \right) \rho_{\nu}(x_{i}) \\
= \sum_{\nu=1}^{F} \frac{\left( \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} \right) \rho_{\nu}(x_{i})}{\sum_{t=1}^{T} |\alpha_{t}|} \\
= \frac{\sum_{\nu=1}^{F} \left( \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} \right) \rho_{\nu}(x_{i})}{\sum_{t=1}^{T} |\alpha_{t}|} \\
= \frac{\ell_{i} \sum_{\nu=1}^{F} \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} h_{(\nu,j)}(x_{i})}{\sum_{t=1}^{T} |\alpha_{t}|} \\
= \rho(x_{i})
\]

This helps to support the use of \( \psi_{\nu} \) as an indicator of the utility of a given feature \( \nu \), which is basically the contribution ratio (CR) weight criterion that Tsuchiya and Fujiiyoshi [98] and Alshawabkeh et al. [11] proposed. However, while a feature \( \nu \) may have a large \( \psi_{\nu} \), it will not contribute to a good overall margin unless \( \rho_{\nu} \) is also large. A better indicator is the fraction of the overall margin that is due to \( \nu \):

\[
\rho_{\nu}(x_{i}) = \frac{\psi_{\nu} \rho_{\nu}(x_{i})}{\rho(x_{i})} \tag{3.16}
\]

Note, however, that this only deals with a single instance \( i \). To consider the margin across all \( m \) instances we use the average margin, which can be redefined as:

\[
\bar{\rho} = \frac{1}{m} \sum_{i=1}^{m} \sum_{\nu=1}^{F} \psi_{\nu} \rho_{\nu}(x_{i}) \tag{3.17}
\]

Thus, the Margin Fraction (MF) due to feature \( \nu \) is computed as:

\[
MF_{\nu} = \frac{\psi_{\nu} \frac{1}{m} \sum_{i=1}^{m} \rho_{\nu}(x_{i})}{\frac{1}{m} \sum_{i=1}^{m} \rho(x_{i})} \\
= \frac{\psi_{\nu} \sum_{i=1}^{m} \rho_{\nu}(x_{i})}{\sum_{i=1}^{m} \rho(x_{i})} \\
= \frac{\sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} \sum_{i=1}^{m} \ell_{i} \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} h_{(\nu,j)}(x_{i})}{\sum_{t=1}^{T} |\alpha_{t}| \sum_{i=1}^{m} \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} h_{(\nu,j)}(x_{i})} \\
= \frac{\sum_{i=1}^{m} \sum_{j=1}^{N_{\nu}} \ell_{i} \alpha_{(\nu,j)} h_{(\nu,j)}(x_{i})}{\sum_{i=1}^{m} \sum_{j=1}^{N_{\nu}} \ell_{i} \alpha_{t} h_{t}(x_{i})} \tag{3.18}
\]

We can use MF as an indicator of the utility of a given feature \( \nu \). Typically, the higher the value of MF, the better the feature \( \nu \).
CHAPTER 3. HYPOTHESIS WEIGHTING FEATURE SELECTION

3.5 AUC Margin Fraction Weighting and Selection

In this section, we show how to extend the margin fraction weighting concept to select informative features when the underlying model is trained using data with skewed class distributions. Our algorithm accounts for the class distribution by optimizing boosting for the AUC. A new weight metric is defined, we refer to it as AUC Margin Fraction (AMF).

3.5.1 Preliminaries

Definition 3.5.1 Let $S^+$ be the subset of positive instances, and $S^-$ be the subset of negative instances. We say that a distribution $D$ over $S$ is nontrivial if $D$ has non-zero probability over both positive and negative instances. Given a non-trivial distribution $D$, we denote $D^+$ and $D^-$ as the marginal distribution of $D$ over positive and negative instances, respectively.

In this study, the AUCBoost algorithm proposed by Long and Servedio [65] is used. Next, we briefly describe the algorithm.

AUCBoost algorithm

The algorithm takes as input the labeled training examples and a set of weak classifiers as defined above. The approach will force a weak classifier that achieves accuracy (slightly better than random guessing) on the positive and negative distributions, to generate a two-sided weak hypothesis $h_t : X \rightarrow \{-1, 1\}$. For any $\theta \in \mathbb{R}$ that thresholds $h$, a confusion matrix can be constructed where

- $\lambda_\theta$ is the false positive rate obtained when $D^- [h(x) > \theta]$ and is equal to
  \[ \lambda_\theta = \Pr_{x \in D^-} [h(x) = 1] \] (3.19)

- $\beta_\theta$ is the true positive rate obtained when $D^+[h(x) > \theta]$, and is defined as
  \[ \beta_\theta = \Pr_{x \in D^+} [h(x) = 1] \] (3.20)

- $\mu_\theta$ is the false negative rate obtained when $D^- [h(x) \leq \theta]$ and is defined as
  \[ \mu_\theta = \Pr_{x \in D^+} [h(x) = -1] \] (3.21)

- $\zeta_\theta$ is the true negative rate obtained when $D^+[h(x) \leq \theta]$, and is defined as
  \[ \zeta_\theta = \Pr_{x \in D^-} [h(x) = -1] \] (3.22)

As in any boosting algorithm, AUCBoost calls a given weak or base learning algorithm repeatedly in a series of rounds $t = 1, \cdots, T$. Three different distributions $D$, $D^-$, and $D^+$ need to be maintained at each round of boosting. Initially, for each $x_i \in X$, $D_1(i) = 1/m$, $D_1^+(i) = 1/2p$, and $D_1^-(i) = 1/2q$. At each iteration $t$, a two-sided hypothesis $h_t$ is chosen that maximizes the AUC (i.e., performs best
on both weighted positive and negative examples). The AUC of \( h \) over the distribution \( D \) is defined as

\[
AUC(h; D) = \Pr_{x_i \in D^+, x_j \in D^-} \{ h(x_i) > h(x_j) \} + \frac{1}{2} \Pr_{x_i \in D^+, x_j \in D^-} \{ h(x_i) = h(x_j) \},
\]

(3.23)

The higher the AUC, the more useful is the hypothesis for classifying the training examples.

After a hypothesis is received, the algorithm then reweights the training examples by setting:

\[
D_{t+1} = \frac{D_t \exp(-\alpha_t h_t(x_i))}{Z_t}
\]

(3.24)

\[
D^+_t = \frac{D_t^+ \exp(-\alpha_t h_t(x_i))}{Z_t^+}
\]

(3.25)

\[
D^-_t = \frac{D_t^- \exp(\alpha_t h_t(x_i))}{Z_t^-}
\]

(3.26)

Here \( Z_t, Z_t^+ \) and \( Z_t^- \) are normalization factors chosen so that \( D_{t+1}, D_{t+1}^+, D_{t+1}^- \) are distributions.

Combining the hypotheses generated by these weak learners will collectively produce the following weighted linear classifier:

\[
H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)
\]

(3.27)

A pseudo-code for the AUCboost algorithm is shown in Algorithm 7.

### 3.5.2 AUC margin fraction weight (AMF)

In this section, we show how the concept of boosting hypothesis margin can be extended and used to select informative features when the underlying model is trained using data with skewed class distributions. Our algorithm accounts for the class distribution by optimizing boosting for the AUC, and weights features by measuring their effect on the average margin combined in the final output produced by boosting. For simplicity, we refer to the new weight criterion as AUC Margin Fraction (AMF).

**Definition 3.5.2** Let \( F \) be the total number of unique features used across all \( T \) rounds, i.e., decision stumps, and for any chosen feature \( v \), let \( h_{(v,j)} \) be the decision stump corresponds to the \( j \)-th use of feature \( v \), and let \( \alpha_{(v,j)} \) be the associated voting confidence. Let \( N_v \) be the total number of times that feature \( v \) is used. We then have \( \sum_{v=1}^F N_v = T \).

We assume that:

\[
Pr_{x_i \in D^+} [h(x) = 1] = 1 - \epsilon^+
\]

(3.28)

\[
Pr_{x_i \in D^-} [h(x) = -1] = 1 - \epsilon^-
\]

(3.29)
Algorithm 7 AUCBoost Algorithm (Long & Servedio, 2007)

Input: Given a labeled training set $S = \{(x_i, \ell_i)\}_{i=1}^m$, where $x_i \in \mathcal{X}$ are samples, $\ell_i \in \{1, -1\}$ are labels, $m = q + p$, and $q$ and $p$ are numbers of negative and positive samples, respectively. Let $T \in \mathbb{N}$ be the number of weak classifiers to combine.

Initialize: $D_1(i) = 1/m, D_1^+(i) = 1/2p, D_1^-(i) = 1/2q$ for all $i = 1, \cdots, m$

for $t = 1, \cdots, T$ do

$\mathcal{H} = h_t : \mathcal{X} \to \{-1, 1\}$

$h_t^* = \underset{h_t \in \mathcal{H}}{\text{argmax}} \text{AUC}_t$, where $\text{AUC}_t = 1 - \frac{1}{2} (e_t^+ + e_t^-)$

$\alpha_t^* = \frac{1}{2} \log \left( \frac{1 - e_t^- - e_t^+}{e_t^+ + e_t^-} \right)$

$D_{t+1} = \frac{D_t \exp(-\epsilon_t h_t^*(x_i))}{Z_t}$

$D_{t+1}^+ = \frac{D_t^+ \exp(-\alpha_t h_t^*(x_i))}{Z_t^+}$

$D_{t+1}^- = \frac{D_t^- \exp(\alpha_t h_t^*(x_i))}{Z_t^-}$

where $Z_t, Z_t^+, Z_t^-$ are normalization constants, and $\sum_i^m D_{t+1}(i) = 1, \sum_i^p D_{t+1}^+(i) = 1, \sum_i^q D_{t+1}^-(i) = 1$

Output: $H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t^* h_t^*(x) \right)$

where $e_t^+$ and $e_t^-$ are the error rates with respect to $D^+$ and $D^-$ over $S^+$ and $S^-$, respectively, which are defined as follows:

$$e_t^+ = \frac{\lambda_\theta}{\lambda_\theta + \beta_\theta}$$  \hspace{1cm} (3.30)

$$e_t^- = \frac{\mu_\theta}{\mu_\theta + \zeta_\theta}$$  \hspace{1cm} (3.31)

We then can redefine the AUC as:

$$\text{AUC}(h; D) = 1 - \frac{1}{2} \left( e_t^+ + e_t^- \right)$$  \hspace{1cm} (3.32)

Thus, the associated confidence assigned to $h_t$ can be obtained as

$$\alpha_t = \frac{1}{2} \log \left( \frac{1 - e_t^- - e_t^+}{e_t^+ + e_t^-} \right)$$  \hspace{1cm} (3.33)

For each feature $v$, the weighted linear combination associated with it that feature, as well as the “conditional” margin associated with just that weighted linear combination.
CHAPTER 3. HYPOTHESIS WEIGHTING FEATURE SELECTION

\[ H(x_i) = \sum_{\nu=1}^{F} \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} h_{(\nu,j)}(x_i) \] (3.34)

\[ \rho(x_i) = \frac{\ell_i \sum_{\nu=1}^{F} \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} h_{(\nu,j)}(x_i)}{\sum_{i=1}^{T} |\alpha_i|} \] (3.35)

Now for any individual feature \( \nu \), one can consider the weighted linear combination associated with that feature and the “conditional” margin associated with just that weighted linear combination for any instance \( i \).

\[ H_{\nu}(x_i) = \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} h_{(\nu,j)}(x_i) \] (3.36)

\[ \rho_{\nu}(x_i) = \frac{\ell_i \sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)} h_{(\nu,j)}(x_i)}{\sum_{j=1}^{N_{\nu}} |\alpha_{(\nu,j)}|} \] (3.37)

Consider the fraction of the absolute “confidence” weight associated with any feature \( \nu \), defined as follows:

\[ \psi_{\nu} = \frac{\sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)}}{\sum_{i=1}^{T} |\alpha_i|} = \frac{\sum_{j=1}^{N_{\nu}} \alpha_{(\nu,j)}}{\sum_{i=1}^{T} \sum_{j=1}^{N_{\nu}} |\alpha_{(\nu,j)}|} \] (3.38)

Based on Theorem 3.4.1, the overall margin associated with any instance “\( i \)” is the weighted linear combination of conditional margins, where \( \psi_{\nu} \) are the weights.

\[ \sum_{\nu=1}^{F} \psi_{\nu} \rho_{\nu}(x_i) = \rho(x_i) \]

This only deals with a single instance \( i \). To consider the entire margin we again consider the average margin \( E_S[\rho(x)] \) of both positive and negative examples, and the effect of a feature on this value.

\[ E_{catS}[\rho(x)] = \frac{1}{m} \sum_{i=1}^{m} \rho(x_i) \]

\[ = \frac{1}{(q + p)} \sum_{j=1}^{q} \sum_{\nu=1}^{F} \psi_{\nu} \rho_{\nu}(x_j) + \]

\[ \frac{1}{(q + p)} \sum_{k=1}^{p} \sum_{\nu=1}^{F} \psi_{\nu} \rho_{\nu}(x_k) \] (3.39)
Thus, the AUC Margin Fraction (AMF) due to feature $\nu$ is computed as:

$$AMF_\nu = \psi_\nu \left( \sum_{j=1}^{q} \psi_\nu \rho_\nu(x_j) + \sum_{k=1}^{p} \rho(x_k) \right) + \sum_{i=1}^{n} \psi_i \rho_i(x_i) + \sum_{k=1}^{n} \rho(x_k)$$

(3.40)

$$= \psi_\nu \left( \sum_{j=1}^{q} \psi_\nu \rho_\nu(x_j) + \sum_{k=1}^{p} \rho(x_k) \right)$$

(3.41)

Now we can use $AMF_\nu$ as an indicator of the utility of a given feature $\nu$. The higher the $AMF_\nu$, the better is the feature $\nu$ for classifying the positive instances (i.e., minority class) and the negative instances (i.e., majority class).

### 3.6 Boosting for Feature Selection

There are basically two approaches how Adaboost type algorithms can be applied as a feature selection method:

#### 3.6.1 Standard boosting

The Adaboost algorithm is employed mainly in its standard mode [32], i.e., all available features are used during the whole learning process. Only a unique set of features linked to all generated weak classifiers is selected once the training is finished. Other features can be removed, because they do not influence the final strong classifier ensemble (they are redundant for the classification). The measurement cost in the target application (classification phase) is reduced by generating, or measuring; only the selected features. The same feature used by several weak classifiers (i.e., feature selected more than once) is not a problem, because complexity of the final strong classifier ensemble does not increase in the dimensionality.

#### 3.6.2 Boosting with feature removal

A variant of the Adaboost algorithm applied for the feature selection task was introduced by Das [30] in 2001. The original Adaboost is modified in such a way that features attached to weak classifiers are gradually removed from the set of all available measurements based on a sequential forward search. Apparently, the removed features cannot be used for further learning which limits the flexibility to create new weak classifiers. The final composite decision rule is therefore not as strong as it would be in the original Adaboost classifier. The resulting feature subset is formed from the features picked for the output decision rule. Das claims that features selected in this way are effective with other types of learning algorithms. However, this variant of the Adaboost algorithm is very likely to select highly correlated features due to its greedy strategy and thus it is not very effective.
3.7 Design and Implementation

In our proposed feature selection algorithms the problem of searching the “best” subset of features is solved by means of a greedy algorithm based on backward selection [54]. A backward sequential selection is used because of its lower computational complexity compared to randomized or exponential algorithms and its optimality in the subset selection problem [27].

3.7.1 SBS-MF algorithm

Here the MF weight used as a ranking criterion for our proposed algorithm. The selection process starts from a full set of features then removes sequentially the most irrelevant ones. To find the most irrelevant feature of the current surviving subset, AdaBoost algorithm is trained on the training set with the current surviving features subset. The classification results of AdaBoost are then used to obtain the MF weights for each feature.

Algorithm 8 The SBS-MF algorithm, (Alshawabkeh et al., 2012)

\begin{verbatim}
Input: Given a labeled training set \( S = \{(x_i, \ell_i)\}_{i=1}^m \) where \( x_i \in \mathcal{X} \) are samples, \( \ell_i \in \{1, -1\} \) are labels. Let \( T \in \mathbb{N} \) be the number of weak classifiers to combine.
\( \Upsilon = \{v_1, v_2, \cdots, v_n\} \) // subset of surviving features
\( F = \emptyset \) // feature ranked list

while \((\Upsilon \neq \emptyset)\) do
    \( \rho_i = 0 \) for \( i : v_i \in \Upsilon \) // vector of contributed margin of features
    \( \psi_i = 0 \) for \( i : v_i \in \Upsilon \) // vector of weights of features
    AdaBoost\((S, \Upsilon, \rho, \psi)\) // train AdaBoost classifier using \( \Upsilon \), cf. Algorithm 6
    for \( j : v_j \in \Upsilon \) do
        \( MF_{v_j} = \psi_j \frac{\sum_{i=1}^m \rho_i(x_i)}{\sum_{i=1}^m \rho_i(x_i)} \)
    endfor
    \( j^* = \arg\min_{j : v_j \in \Upsilon} (MF_{v_j}) \) // find the worst feature with smallest MF weight
    \( F = F \cup v_{j^*} \) // update feature ranked list
    \( \Upsilon = \Upsilon \setminus v_{j^*} \) // reduce the set of available features
endwhile

Output: feature ranked list \( F \).
\end{verbatim}
The features are then ranked based on the MF weight criterion. Finally, the most irrelevant feature, which its MF weight is the smallest, is eliminated. The procedure is repeated until \( r \) features are removed or all of the features are ranked. Algorithm 8 summarizes the SBS-MF method and a high level design is shown in Figure 3.8.

### 3.7.2 SBS-AMF algorithm

Here the AMF weight is used as a ranking criterion for our proposed algorithm. Again the sequential backward selection search algorithm is used. The selection process starts from a full set of features then removes sequentially the most irrelevant ones. To find the most irrelevant feature of the current surviving subset, AUCBoost algorithm is trained on the training set with the current surviving features subset. The classification results of AUCBoost are then used to obtain the AMF weights for each feature. The features are then ranked based on the AMF weight criterion. Finally, the most irrelevant feature, which its MF weight is the smallest, is eliminated. The procedure is repeated until \( r \) features are removed or all of the features are ranked. Algorithm 9 summarizes the SBS-AMF method and a high level design is shown in Figure 3.8.

![Figure 3.8: High level design of SBS-MF/SBS-AMF algorithms.](image)

### 3.7.3 Computational complexity analysis

In the training stage, the computational complexity of the SBS-MF arises from the construction of the decision stumps and strong classifiers. For the construction of the decision stumps, all samples should be searched for each feature; thus, the computational complexity for constructing the decision stumps is \( O(nF) \), where \( n \) is the number of samples, and \( F \) is the number of features, i.e., the number
Algorithm 9 The SBS-AMF algorithm, (Alshawabkeh et al., 2011)

**Input**: Given a labeled training set \( S = \{(x_i, \ell_i)\}_{i=1}^m \), where \( x_i \in \mathcal{X} \) are samples, \( \ell_i \in \{1, -1\} \) are labels, \( m = q + p \), and \( q \) and \( p \) are numbers of negative and positive samples, respectively. Let \( T \in \mathbb{N} \) be the number of weak classifiers to combine.

\[ Y = \{v_1, v_2, \cdots, v_n\} \quad \text{// subset of surviving features} \]

\[ \mathcal{F} = \phi \quad \text{// feature ranked list} \]

**while** \((Y \neq \phi)\) **do**

\[ \rho_i = 0 \quad \text{for } i : v_i \in Y \quad \text{// vector of contributed margin of features} \]

\[ \psi_i = 0 \quad \text{for } i : v_i \in Y \quad \text{// vector of weights of features} \]

\[ \text{AUCBoost}(S, Y, \rho, \psi) \quad \text{// train AdaBoost classifier using } Y, \text{ cf. Algorithm 7} \]

**for** \(( j : v_j \in Y)\)** do**

\[ AMF_{v_j} = \psi_i \left( \frac{\sum_{j=1}^{q} \rho_i(x_j) + \sum_{k=1}^{p} \rho_k(x_k)}{\sum_{j=1}^{q} \rho(x_j) + \sum_{k=1}^{p} \rho(x_k)} \right) \]

**endfor**

\[ j^* = \arg\min_{j:v_j \in Y} (AMF_{v_j}) \quad \text{// find the worst feature with smallest AMF weight} \]

\[ \mathcal{F} = \mathcal{F} \cup v_{j^*} \quad \text{// update feature ranked list} \]

\[ Y = Y \setminus v_{j^*} \quad \text{// reduce the set of available features} \]

**endwhile**

**Output**: feature ranked list \( \mathcal{F} \).

of decision stumps. There are \( T \) iterations for constructing the strong classifier. Therefore, in the training stage of the AdaBoost algorithm, the computational complexity is only \( O(nTF) \). Since the sequential backward search is used, this process is repeated \( \mathcal{F} \) number of times. Therefore, the computational complexity of SBS-MF and SBS-AMF is \( O(nTF^2) \). In short, our feature selection boosting-based algorithm possesses the lowest computational complexity in the published learning algorithms for feature selection. This property is very attractive and promising.

### 3.8 Chapter Conclusions

We proposed two embedded-based feature selection techniques based on the boosting algorithm. In particular, our approaches are based on the concept of the hypothesis margin induced by boosting. In the first algorithm, a new weight metric, termed Margin Fraction (MF), is proposed. It defines the quality of a set of features based on their contribution to the average margin produced. In the second algorithm, we extend this methodology to define another weight metric, to handle the presence of data with imbalance class distributions, by considering the quality of features based
on the maximized AUC margin induced. Our algorithms have the advantage that they include the interaction with the classification model, while at the same time being far less computationally intensive. In addition, they complement current boosting-based feature selection techniques by measuring the relative importance of features based on their contribution to the overall margin induced, rather than to their contribution to the confidence ratio of the learned base hypothesis.
Chapter 4

Experimental Evaluation of The Proposed Feature Selection Algorithms

This chapter discusses the experimental methodology applied for performance evaluation of the proposed feature selection algorithms. We create grounds for a fair comparison of the proposed feature selection algorithms through extensive comparisons with other state-of-the-art methods, including boosting-based and margin-based feature selection algorithms using real-world high-dimensional data from UCI repository [77] and datasets that contain a large number of features with a small number of samples and significant imbalance between the two classes. We also try to address potential issues in feature selection, such as the impact of the existence of correlated features, and exploring differences in performance on highly imbalanced classes versus balanced classes.

4.1 Data sets

To validate performance fairly and to provide a comprehensive testing suite for feature selection methods under different conditions, two groups of benchmark datasets are adopted in our simulation experiments.

4.1.1 UCI datasets

The first group includes datasets with large number of samples. These datasets are all available from the UCI Machine Learning Repository [77] and most of them are frequently used in the literatures. Some of these datasets may embody missing values or continuous features, and so they would be
processed during the preprocessing phases. For missing values, we replaced them with the most frequently used values and means for nominal and numeric features, respectively.

Table 4.1 summarizes some general information about these datasets. Their full documentation is shown below:

**Chess Dataset**: Chess End-Game – King+Rook versus King+Pawn on a7 (usually abbreviated KRKPA7). The pawn on a7 means it is one square away from queening. It is the King+Rook’s side (white) to move. The dataset has 3196 instances with 36 attributes. It has 2 classes: White-can-win ("won") and White-cannot-win ("nowin"). The class distribution is as follows, in 1669 of the positions (52%), White can win, and in 1527 of the positions (48%), White cannot win. The format for instances in this database is a sequence of 37 attribute values. Each instance is a board-descriptions for this chess endgame. The first 36 attributes describe the board. The last (37th) attribute is the classification: “win” or “nowin”. There are 0 missing values. The names of the features do not appear in the board-descriptions. Instead, each feature corresponds to a particular position in the feature-value list. For example, the head of this list is the value for the feature “bkblk”.

**Ionosphere Dataset**: Johns Hopkins University Ionosphere database. This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. “Good” radar returns are those showing evidence of some type of structure in the ionosphere. “Bad” returns are those that do not; their signals pass through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal. There are 351 instances and 34 attributes plus the class attribute, i.e., “good” or “bad”. All 34 predictor attributes are continuous.

**Mushroom Dataset**: This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like “leaflets three, let it be” for Poisonous Oak and Ivy. It has 8124 instances and 22 attributes: 22 (all nominally valued). 2 classes in the data: edible=e, poisonous=p, with class distribution: edible: 4208 (51.8%), poisonous: 3916 (48.2%).

**MUSK Clean1 Dataset**: This dataset describes a set of 92 molecules of which 47 are judged by human experts to be musks and the remaining 45 molecules are judged to be non-musks. The
CHAPTER 4. EXPERIMENTAL EVALUATION

63

goal is to learn to predict whether new molecules will be musks or non-musks. However, the
166 features that describe these molecules depend upon the exact shape, or conformation, of
the molecule. Because bonds can rotate, a single molecule can adopt many different shapes.
To generate this data set, the low-energy conformations of the molecules were generated and
then filtered to remove highly similar conformations. This left 476 conformations. Then, a
feature vector was extracted that describes each conformation. This many-to-one relationship
between feature vectors and molecules is called the “multiple instance problem”. When learn-
ing a classifier for this data, the classifier should classify a molecule as “musk” if ANY of its
conformations is classified as a musk. A molecule should be classified as “non-musk” if NONE
of its conformations is classified as a musk. The dataset has 476 instances with 168 attributes
plus the class. The class distribution is: 47 musks and 45 non-musks.

Spambase Dataset: This is a spam e-mail database, with 48 continuous real [0,100] attributes
of type word \_freq.WORD = percentage of words in the e-mail that match WORD, i.e. 100
\* (number of times the WORD appears in the e-mail) / total number of words in e-mail. A
“word” in this case is any string of alphanumeric characters bounded by non-alphanumeric
characters or end-of-string. 6 continuous real [0,100] attributes of type char \_freq.CHAR =
percentage of characters in the e-mail that match CHAR, i.e., 100 \* (number of CHAR occur-
rences) / total characters in e-mail.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Features</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>32</td>
<td>3196</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>34</td>
<td>351</td>
</tr>
<tr>
<td>Mushroom</td>
<td>22</td>
<td>8124</td>
</tr>
<tr>
<td>Musk clean1</td>
<td>166</td>
<td>476</td>
</tr>
<tr>
<td>Spambase</td>
<td>57</td>
<td>4601</td>
</tr>
</tbody>
</table>

4.1.2 Imbalanced datasets

The second group includes data sets selected from different application domains to show the effec-
tiveness and robustness of our new feature selection technique on different kinds of imbalanced data
sets. The range of application domains, include: microarray, mass spectrometry, and text mining.
The data sets we examine are summarized in Table 4.2. While there is only a single data set for
the text mining domain, it is rebalanced into five separate class ratios (1 : 1, 1 : 2, 1 : 4, 1 : 8, 1 : 16)
to explore differences in performance on highly imbalanced classes versus balanced classes. Each of
the data sets has a relatively small number of samples, a large number of features, and significant imbalance between the two classes.

The mass spectrometry sets were minimally preprocessed by trimming the range of inspected mass/charge ratios, normalizing and reducing the amount of noise. The text mining set was constructed using [68] to extract the word counts from text documents.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Source</th>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVARY</td>
<td>Ovarian Cancer Data [80]</td>
<td>6000</td>
<td>This Mass Spectrometry data set contains 66 samples: 50 are benign tumors, and 16 are malignant tumors</td>
</tr>
<tr>
<td>PROST</td>
<td>Prostate Cancer Data [81]</td>
<td>6000</td>
<td>This Mass Spectrometry data set contains 89 samples: 63 have no evidence of cancer, and 26 have prostate cancer</td>
</tr>
<tr>
<td>LYMPH</td>
<td>Lymphoma Data [89]</td>
<td>7129</td>
<td>This Microarray data set contains 77 samples: 58 are diffuse large B-cell lymphoma, and 19 are follicular lymphomas</td>
</tr>
<tr>
<td>CNS</td>
<td>Central Nervous System Embryonal Tumor Data [82]</td>
<td>7129</td>
<td>This Microarray data set contains 90 samples: 60 have medulloblastomas and 30 have other types of tumors or no cancer</td>
</tr>
<tr>
<td>NIPS</td>
<td>Bag-of-Words Data(NIPS) [86]</td>
<td>13649</td>
<td>This data set contains 320 documents: 160 cover neurobiology topics, and 160 cover various applications topics. The set was rebalanced for five separate class ratios: 1:1, 1:2, 1:4, 1:8, and 1:16. The neurobiology class was the class shrunk to account for these imbalances</td>
</tr>
</tbody>
</table>

4.2 Performance Assessment

The performance of SBS-MF was assessed using the UCI benchmark data sets and was compared against, AdaBoost-CR, BDSFS and Simba, and the performance of SBS-AMF was assessed using the imbalanced benchmark data sets and was compared against, AdaBoost-CR, BDSFS, Simba and FAST. See chapter2 for more details about these algorithms.

In our simulation experiments, datasets are initially fed into these four feature selectors, which will generate different feature subsets from each full dataset. Since the number of features chosen by these four selectors is different, we chose the same quantity of features for the sake of impartiality and the selected features were considered in descending order according to their priorities. After this initial step, datasets with newly selected features were passed to the learning algorithms to assess classification performance.
4.2.1 Learning algorithms

Currently, there are a number of powerful learning algorithms available. In our experiments we selected two different classifiers, namely, 1-Nearest Neighbor (1-NN) and SVM, to evaluate the prediction capabilities of the selected subset. The reason we selected these classifiers is because they represent significantly different approaches to learning and are commonly used by the data mining community. We briefly describe these classification techniques.

1-Nearest Neighbor (NN) classifier

The 1-NN classifier is one of the oldest methods known. The idea is extremely simple: to classify \( x \) find its closest neighbor among the training points (call it \( \hat{x} \)) and assign to \( x \) the label of \( \hat{x} \).

Support vector machine (SVM)

In machine learning, support vector machines (SVMs) [26] are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. The basic SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the output, making it a non-probabilistic binary linear classifier. Given a set of training examples, each marked as belonging to one of two categories, a SVM training algorithm builds a model that assigns new examples into one category or the other. A SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. In addition to performing linear classification, SVMs can efficiently perform non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

4.2.2 Results evaluation

To achieve objective results, a \( k \)-fold cross validation and stratified cross validation were employed for each algorithm-dataset combination to evaluate classification accuracy. For each dataset we ran each classification algorithm before and after feature selection. We ran each classification algorithm 10 times and used 10-fold cross validation. The results are presented as averages over all of these runs.

To judge the significance of the differences found in our results, we ran pair t-tests on the output accuracies both before feature selection and then again applying each feature selection algorithm. Throughout this paper, a difference in accuracy is considered significant if its \( p \)-value is less than 0.05 (i.e., confidence level greater than 95%) when applying the paired t-test.
4.3 Performance Evaluation of SBS-MF Algorithm

In this section, we report the experimental results on classification performance for SBS-MF algorithm on the UCI datasets. We also perform a comparison study between the proposed margin fraction weight and the contribution ratio criterion defined in AdaBoost-CR algorithm proposed by Tsuchiya and Fujiyoshi [98]. Where in their work the features are evaluated based on the contribution ratio (CR), which is defined as the relative importance of features based on the confidence ratio of the learned base hypothesis. Tsuchiya and Fujiyoshi algorithm starts with a given set of features, the CR of each features from a feature set is estimated, and features that are not contributing and that have a low CR are removed.

4.3.1 Performance results

The results for applying the four feature selection algorithms are presented in Table 4.4. The accuracy of each classifier for the different datasets using only the original features is shown in Table 4.3. Notation “●” (or “○”) in Table 4.4 denotes that the performance of the classifier with the current feature selector is significantly better (or worse) than the result that does not use the selector (i.e., the results in Table 4.3), based on the outcome of the t-test. Also, entries denoted in bold show the best results from the four different feature selection methods all using the same classifier. We present the average accuracy assuming the same feature selector in the “Ave.” row.

The results in Table 4.4 show that the performance using SBS-MF is better than other approaches for the 1-NN classifier. SBS-MF achieves the best classification accuracy in all results over 5 datasets. In terms of average performance, SBS-MF is superior to other selectors.

For the SVM classifier, we can see that our proposed method clearly surpasses others in most cases. As an illustration, for SBS-MF, we find significantly better accuracy in two cases and slightly lower performance in only a single case. In addition, SBS-MF produces the best classification accuracy for all datasets, which is higher than any of the other selection algorithms. Correspondingly, the average accuracy ratios for SBS-MF are the best for both 1-NN and SVM.

Table 4.3: Classification accuracy rates obtained by 1-NN and SVM without feature selection.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1-NN (%)</th>
<th>SVM(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chess</td>
<td>89.73 ± 0.38</td>
<td>87.43 ± 1.42</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>85.80 ± 0.14</td>
<td>86.04 ± 1.13</td>
</tr>
<tr>
<td>Mushroom</td>
<td>97.84 ± 0.05</td>
<td>97.97 ± 0.26</td>
</tr>
<tr>
<td>Musk clean1</td>
<td>89.28 ± 4.41</td>
<td>86.27 ± 1.76</td>
</tr>
<tr>
<td>Spambase</td>
<td>91.58 ± 6.21</td>
<td>90.76 ± 4.11</td>
</tr>
<tr>
<td>Ave.</td>
<td>90.85 ±2.24</td>
<td>89.64 ±1.74</td>
</tr>
</tbody>
</table>
Table 4.4: A comparison of classification accuracies of classifiers using four feature selection algorithms on UCI datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1-NN (%)</th>
<th>SVM(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBS-MF</td>
<td>AdaBoost-CR</td>
</tr>
<tr>
<td>Chess</td>
<td>94.20 ± 0.49*</td>
<td>91.79 ± 4.41*</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>92.73 ± 0.38*</td>
<td>87.71 ± 0.63*</td>
</tr>
<tr>
<td>Mushroom</td>
<td>99.91 ± 0.12</td>
<td>93.1 ± 0.38</td>
</tr>
<tr>
<td>Musk clean1</td>
<td>94.64 ± 1.30*</td>
<td>83.62 ± 2.13°</td>
</tr>
<tr>
<td>Spambase</td>
<td>94.56 ± 0.61</td>
<td>91.97 ± 0.78</td>
</tr>
<tr>
<td>Ave.</td>
<td>95.21 ± 0.58</td>
<td>89.64± 1.66</td>
</tr>
</tbody>
</table>

4.3.2 Comparison between margin fraction and contribution ratio

Tsuchiya and Fujiyoshi [98] and Alshawabkeh et al., [11] have proposed a feature evaluation method based on AdaBoost with decision stumps. They introduced a metric, called contribution ratio CR, that indicates how well the features “contribute” to the classification performance based on the performance weight $\alpha$ of the weak hypothesis $h_t$. A contribution ratio $CR_v$ for each feature $v$ is
defined by:

\[ CR_\upsilon = \sum_{t=1}^{T} \bar{\alpha}_t \delta_K[P(h_t) - \upsilon] \tag{4.1} \]

where \( \bar{\alpha}_t \) is the average confidence assigned to feature \( \upsilon \), \( \delta_K \) is the Kronecker delta; which is a function of two variables that is 1 if they are equal and 0 otherwise, and \( P() \) is a function for outputting the feature chosen at round \( t \) in the AdaBoost training process.

In other words, the \( CR_\upsilon \) is equal to the fraction of the absolute “confidence” weight associated with any feature \( \upsilon \), thus, it is equal to \( \psi_\upsilon \) that we defined in Chapter 3 as:

\[ \psi_\upsilon = \frac{\sum_{j=1}^{N_\upsilon} \alpha_{(\upsilon,j)}}{\sum_{t=1}^{T} |\alpha_t|} \tag{4.2} \]

\[ = \frac{\sum_{j=1}^{N_\upsilon} \alpha_{(\upsilon,j)}}{\sum_{u=1}^{U} \sum_{j=1}^{N_{(u,j)}} |\alpha_{(u,j)}|} \]

However, while \( CR_\upsilon \) is a helpful evaluation metric of the quality of a given feature \( \upsilon \), it is not an accurate indicator of its performance. A feature \( \upsilon \) may have a large \( CR_\upsilon \) value but will not contribute to a good overall margin. A better indicator is the margin fraction MF that is due to \( \upsilon \).

To better explain this, we generated scatter plots, shown in Figures 4.1a and 4.1b, of the two best selected features for SBS-MF and AdaBoost-CR, using the Lymphoma data set for training. SBS-MF features are able to separate the two classes and group them into smaller clusters better than the AdaBoost-CR.

To understand the relationship between the MF and CR weights, we train boosting for \( T = 100 \) using UCI dataset and then report the MF and CR weights assigned to the selected features. The plots of Figure 4.2 show the relationship between CR and MF. As shown those weights are not correlated, meaning a feature might have a high CR weight but not a high MF weight.

Figure 4.1: Training data distribution of Ionosphere with the two best selected features by each approach.
Figure 4.2: $\psi$ or CR weights vs. MF weights generated by boosting with $T = 100$ when trained with each UCI data set.
4.4 Performance Evaluation of SBS-AMF Algorithm

We use real data sets selected from different application domains including: microarray, mass spectrometry and textmining. The data sets we examine are summarized in Table 4.2. While there is only a single data set for the text mining domain, it is rebalanced into five separate class ratios (1:1, 1:2, 1:4, 1:8, and 1:16) to explore differences in the performance when encountering highly imbalanced classes versus balanced classes. Each of the data sets has a large number of features and significant imbalance between the two classes.

4.4.1 Experimental setup

We evaluate our results with three different measures:

i) AUC.

ii) BER – which assesses the balance error rate with respect to both positive and negative examples [24], and is defined as the average of the errors in each class (this statistic is similar in nature to the AUC in that it weights errors differently in the two classes).

iii) F$_1$-measure [36]– which is the harmonic mean of the precision $p$ and recall $r$. $p$ is the number of correct results divided by the number of all returned results and $r$ is the number of correct results divided by the number of results that should have been returned. An F$_1$ score reaches its best value at 1.0 and worst score at 0.0 as shown in Equation 4.3.

$$F_1 = \frac{2pr}{p + r} \quad (4.3)$$

where

$$p = \frac{tp}{tp + fp} \quad , \quad r = \frac{tp}{tp + Ffn} \quad (4.4)$$

We compare the performance of our approach against four feature selection algorithms: Simba (a margin-based feature selection method), FAST (an AUC based feature selection method), BDSFS (Boosting decision stumps with information-gain), and AdaBoost-CR ((Boosting decision stumps).

We applied stratified 10-fold cross-validation to evaluate the methods. We repeated the stratified 10-fold cross-validation 20 times and report the averaged results as we vary the number of features selected by each method. To determine whether the difference is significant or not, pair t-tests between accuracies without feature selector and with each selector at a time had been performed. Throughout this paper, the difference of accuracies is considered significantly different if its $p$-value is less than 0.05 (i.e., confidence level greater than 95%) according to a paired t-test. We tested the effectiveness of the selected feature sets produced by the four different feature selection methods by evaluating their use in 1-NN and SVM classifiers.
4.4.2 Performance results

Sub-figures in Figures 4.3 and 4.4 show the classification results in terms of AUC versus the number of features selected using an 1-NN and SVM classifiers, and Sub-figures in Figures 4.5 and 4.6 show the classification results in terms of BER versus the number of features selected using an 1-NN and SVM classifiers.

Lines with □ markers indicate classifiers using, lines with ◆ markers indicate classifiers using FAST-selected features, lines with ★ markers indicate classifiers using BDSFS-selected features, lines with + markers indicate classifiers using AdaBoost-CR-selected features, and lines with × markers indicate classifiers using SBS-MF-selected features and the solid black line indicates the performance where all the features are used for classification.

The AUC results in Figures 4.3 and 4.4 show that the features selected by AMF outperform those selected by other approaches for both 1-NN and SVM classifiers. Note that AMF is able to achieve this with as few as 20 features. What this indicates is that AMF is more effective in selecting fewer high quality features, such that after keeping 60 features, adding more features may not improve and may even degrade classifier performance. We observe the same trend with the BER results in Figures 4.5 and 4.6 when using 1-NN and SVM that SBS-MF achieves a lower BER value compared to the other methods with fewer features. What this indicates is that SBS-MF is more effective in selecting fewer high quality features, such that after keeping 70 features (with 1-NN), adding more features may not improve and may even degrade classifier performance. Further, note that SBS-AMF, FAST and Simba –AUC and margin based methods, perform better than BDSF and AdaBoost-CR in both 1-NN and SVM.

Besides AUC, and BER we also calculated the $F_1$ measure resulting from these methods. We report the results in Table 4.6. Higher $F_1$ values are desired. The best results are highlighted in bold font. Notation “★” (or “◆”) denotes that the performance of the classifier with current selector is significantly better (or worse) than those without using selector, i.e., results in Table 4.5 in statistical test. In addition, the bold value in entries means that it is the largest one among these four feature selectors in the same classifier. The average value of accuracies with the same selector is given in the “Ave.” row.

The results in Table 4.4 are generated using the top ranked 20 features by each selector. As we can see, the results show that the performance using SBS-AMF are better than other approaches in the 1-NN classifier, except for PROST data set whose accuracy on Simba is higher. However, SBS-AMF has three maximal values of classification accuracy and one worse than the results over the four datasets. From the view of average performance SBS-AMF is still relatively superior to other selectors.

For SVM classifier, one may also observe that our proposed method clearly surpasses others in many cases, except for OVARY data set whose $F_1$ measure is higher on FAST. However, for SBS-AMF the quantities of cases with significantly better and worse performance are five and zero,
respectively. In addition, SBS-AMF has the highest classification performance over three-quarter datasets, which is higher than other selection algorithms. Correspondingly, the average value of accurate ratios in SBS-AMF is also the largest one.

![Figure 4.3: AUC vs. the number of ranked features used for training 1-NN classifier, with the five feature selection methods.](image)
CHAPTER 4. EXPERIMENTAL EVALUATION

Figure 4.4: AUC vs. the number of ranked features used for training SVM classifier, with the five feature selection methods.
Figure 4.5: BER vs. the number of ranked features used for training 1-NN classifier, with the five feature selection methods.
Figure 4.6: BER vs. the number of ranked features used for training SVM classifier, with the five feature selection methods.
Table 4.5: $F_1$ measure classification rates obtained by 1-NN and SVM without feature selection.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1-NN (%)</th>
<th>SVM(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVARY</td>
<td>82.2 ± 1.58</td>
<td>82.7 ± 3.10</td>
</tr>
<tr>
<td>PROST</td>
<td>82.6 ± 0.26</td>
<td>83.9 ± 1.36</td>
</tr>
<tr>
<td>LYMPH</td>
<td>81.4 ± 1.69</td>
<td>84.5 ± 1.01</td>
</tr>
<tr>
<td>CNS</td>
<td>81.1 ± 2.34</td>
<td>80.7 ± 1.90</td>
</tr>
<tr>
<td>Ave.</td>
<td>81.8 ± 1.46</td>
<td>82.9 ± 1.84</td>
</tr>
</tbody>
</table>

Table 4.6: A comparison of classification accuracies using $F_1$ measure of classifiers using four feature selection algorithms on microarray datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>1-NN (%)</th>
<th>SVM(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SBS-AMF</td>
<td>FAST</td>
</tr>
<tr>
<td>OVARY</td>
<td>84.8 ± 0.62*</td>
<td>83.3 ± 2.23</td>
</tr>
<tr>
<td>PROST</td>
<td>82.7 ± 1.09</td>
<td>82.4 ± 1.67</td>
</tr>
<tr>
<td>LYMPH</td>
<td>84.0 ± 2.08*</td>
<td>80.9 ± 3.44</td>
</tr>
<tr>
<td>CNS</td>
<td>83.0 ± 0.61*</td>
<td>81.0 ± 0.78</td>
</tr>
<tr>
<td>Ave.</td>
<td>83.63 ± 1.10</td>
<td>81.90 ± 2.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SBS-MF</th>
<th>FAST</th>
<th>BDSFS</th>
<th>Simba</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVARY</td>
<td>83.2 ± 0.93</td>
<td>84.4 ± 2.31*</td>
<td>83.0 ± 1.53</td>
<td>83.1 ± 0.95</td>
</tr>
<tr>
<td>PROST</td>
<td>83.4 ± 0.81</td>
<td>83.3 ± 0.92</td>
<td>80.5 ± 0.97°</td>
<td>82.5 ± 0.84</td>
</tr>
<tr>
<td>LYMPH</td>
<td>84.3 ± 1.12</td>
<td>82.5 ± 2.42</td>
<td>81.90 ± 1.77°</td>
<td>81.12 ± 1.79°</td>
</tr>
<tr>
<td>CNS</td>
<td>86.1 ± 0.31*</td>
<td>84.9 ± 0.48*</td>
<td>79.0 ± 1.02</td>
<td>80.3 ± 0.73</td>
</tr>
<tr>
<td>Ave.</td>
<td>84.25 ± 0.79</td>
<td>83.78 ± 1.53</td>
<td>81.10 ± 1.32</td>
<td>81.76 ± 1.08</td>
</tr>
</tbody>
</table>
4.4.3 The effect of different class ratios

We also studied the effect of different class ratios on the performance of each feature selection technique for the NIPS data. The data is for the text mining domain, and it is rebalanced into five separate class ratios (1 : 1, 1 : 2, 1 : 4, 1 : 8, 1 : 16) to explore differences in performance on highly imbalanced classes versus balanced classes.

Figures 4.7 and 4.8 show the AUC and BER versus class ratios for the NIPS data using the 1-NN classifiers and SVM, respectively. Expectedly, as the class ratio increases, the BER tends to increase accordingly. For the 1-NN and SVM classifiers, SBS-AMF produces features that perform significantly better than AdaBoost-CR, BDSFS and Simba. As for FAST, SBS-AMF is better for the 1-NN classifier. For SVM, SBS-AMF is still better with all ratios except for ratio 1:8, where the achieve similar performance.

4.5 Discussion

One conclusion that can be drawn from the above results is that SBS-MF and SBS-AMF features are doing better than all methods by a significant margin for both 1-NN and SVM classifiers. We next explain why this behavior of using the concept of the margin fraction is effective in selecting informative features. We also study the impact of the existence of correlated features and how our algorithm handles this issue. Finally, we lay out the limitations of our proposed methods and how these potential weaknesses can be addressed.

4.5.1 Analyzing performance

To better explain why the SBS-MF and SBS-AMF features are doing better than all methods, we generated scatter plots of the two best selected features for each method, using the LYMPHOMA data set for training as shown in Figure 4.9. Plot (a) show the data using the best two SBS-AMF features, plot (b) show the best the best two AdaBoost-CR features, plot (c) show the best two BDSFS features, plot (d) show the best two Simba features, and plot (e) show the best two FAST features. As we can see, SBS-AMF features are able to separate the two classes and group them into smaller clusters better than the other methods.

This may explain why SBS-AMF features perform better using both the SVM and 1-NN classifiers; SVM’s try to maximize the distance between two classes, and 1-NN classifiers give the best results when similar samples are clustered close together. This also explains the poor performance obtained for BDSFS and Simba. BDSFS takes a global view as to whether a feature accurately predicts the target; in contrast, Simba, especially when the number of nearest hits and misses selected is small, considers only a local view of a feature’s relevancy to predicting the target.
Figure 4.7: The effect of different class ratios on the AUC performance of each feature selection technique for the NIPS data using 1-NN and SVM.
Figure 4.8: The effect of different class ratios on the BER performance of each feature selection technique for the NIPS data using 1-NN and SVM.
Figure 4.9: Training data distribution of LYMPHOMA with the two best selected features by each feature selection method.
4.5.2 Impact of correlated features

Replicated and highly correlated features contain more or less the same amount of information (apart from modulated noise of a small magnitude). No significant improvement in the class discriminability is therefore gained by adding them. Such features originate from statistically too close measurements. Their use can negatively influence a classifier design by increasing the number of its degrees of freedom. A typical sign is a very high correlation coefficient between features.

In order to demonstrate how SBS-MF algorithm handle correlated features and how they are being weighted and ranked, we designed the following simulation. We use *Iris* data set from UCI [77] for the experimental study, since it only has small number of features (easy to track), and the features are correlated with each other. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. It has 4 numeric features and the class label Iris Setosa, Iris Versicolour, Iris Virginica. The features and statistics about the data are described in Table 4.7.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>SD</th>
<th>Class-Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sepal Length (SL)</td>
<td>4.3</td>
<td>7.9</td>
<td>5.84</td>
<td>0.83</td>
<td>0.7826</td>
</tr>
<tr>
<td>Sepal Width (SW)</td>
<td>2</td>
<td>4.4</td>
<td>3.05</td>
<td>0.43</td>
<td>-0.4194</td>
</tr>
<tr>
<td>Petal Length (PL)</td>
<td>1</td>
<td>6.9</td>
<td>3.76</td>
<td>1.76</td>
<td>0.9490</td>
</tr>
<tr>
<td>Petal Width (PW)</td>
<td>0.1</td>
<td>2.5</td>
<td>1.2</td>
<td>0.76</td>
<td>0.9565</td>
</tr>
</tbody>
</table>

In order to understand the behavior of SBS-MF algorithm in weighting correlated features, we first calculate the correlation for the Iris features using the *Pearson’s correlation coefficient*, which is defined as

\[
r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}
\] (4.5)

The correlation values of Iris features are shown in Table 4.8. As we can see that the highly correlated features are: SL and PL (r = 0.871), SL and PW (r = 0.818), and PL and PW (r = 0.967). Scatter plots in Figure 4.10 show the distribution of the Iris data using every pair of the features to visualize the correlation between them.

We then, modify the Iris dataset by adding three more new features that are redundant and correlated with feature PW at different correlation coefficients \( r = 1, r = 0.85, r = 0.65 \), named: PWcorr1, PWcorr0.85 and PWcorr0.65, respectively. We use the concept of linear models in order to generate these three new correlated features. The idea is that given a variable \( x \), the goal is to find that one variable \( y \) is varying as a straight-line function of the variable \( x \). Using linear models for prediction it turns out very conveniently that the only statistics of interest (at least for purposes of estimating coefficients) are the: mean, variance of each variable and the correlation coefficient
between each pair of variables $r$. The correlation coefficient is not only the average product of the standardized values, but also: the correlation coefficient is the “best” coefficient for multiplying the standardized value of one variable in order to predict the standardized value of the other. That is, the “best” linear model (in a minimum-squared-error sense).

$$\frac{\hat{y}_i - \bar{y}}{\sigma(y)} = r \frac{x_i - \bar{x}}{\sigma(x)} \quad (4.6)$$

$$\hat{y}^*_i = rx^* \quad (4.7)$$

If $x$ is observed to be 1 standard deviation above its own mean, then we should predict that $y$ will be $r$ standard deviations above its own mean; if $x$ is 2 standard deviations below its own mean, then we should be predict that $y$ will be $2r$ standard deviations below its own mean, and so on. In graphical terms, this means that, on a scatter plot of $y^*$ versus $x^*$, the best-fit line for predicting $y^*$ from $x^*$ is the line that passes through the origin and has slope $r$.

We run SBS-MF algorithm with the 7 features for 7 rounds. Histogram plots in Figure 4.11 show the MF weights assigned to each feature at each round. In round 1 the algorithm ranks PL as the best feature (highly correlated with the class label see Table 4.7), and feature PWcorr0.65 as the worst. Notice that although PW is a good feature (highly correlated with the class label see Table 4.7), since this feature has other highly correlated features (PWcorr1, PWcorr0.85) the MF weight was distributed among them, and thus, the PW was ranked fourth. In round 2, PWcorr0.65 was removed, and PW is getting higher MF weight. As we continue we will see removing correlated features helps SBS-MF to find informative features and assign higher weights to them.

Nevertheless, correlation does not always imply absence of information. Consider the case of feature PL and PW. Better class separation and noise reduction may be obtained by selecting both features even though observations are correlated but under presence of noise.

### 4.5.3 Addressing potential weaknesses

While SBS-MF is an effective method, it also introduces a few limitations. In particular the following issues:

i) Our method is not generally recognized that functional values of the criterion guiding the search for the best feature set are random variables. The results of our SBS-MF may therefore be heavily dependent on the sampled training data. The inherent instability often leads to incorrect conclusions regarding the features effectiveness. Hence, it is important to be able to measure and optimize the stability of resulting feature sets.

ii) The sequential greedy search approach is used, and this means adding or removing features sequentially, but have a tendency to become trapped in local minima.
Table 4.8: Correlation coefficient for features of Iris data.

<table>
<thead>
<tr>
<th></th>
<th>SL</th>
<th>SW</th>
<th>PL</th>
<th>PW</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>1</td>
<td>-0.117</td>
<td>0.871</td>
<td>0.818</td>
</tr>
<tr>
<td>SW</td>
<td>1</td>
<td>-0.428</td>
<td>-0.366</td>
<td></td>
</tr>
<tr>
<td>PL</td>
<td>1</td>
<td>0.967</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PW</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.10: Correlation between Iris data set features.
Figure 4.11: MF weights assigned at each round by the SBS-MF algorithm for Iris data set features.
Our weighting method MF might not be robust to noise since it is based on the hard margins of boosting. Part of the future work is to investigate how to overcome this issue by utilizing the soft margins of boosting instead.

4.6 Chapter Conclusions

In this chapter, we discussed the experimental methodology applied for performance evaluation of the proposed feature selection algorithms described in Chapter 3. We created grounds for a fair comparison of the proposed feature selection algorithms through extensive comparisons with other state-of-the-art methods, including boosting-based such as BDSFS and AdaBoost-CR, and margin-based feature selection algorithms such as Sima using real-world high-dimensional data from UCI repository and datasets that contain a large number of features with a small number of samples and significant imbalance between the two classes from different domains including: microarray, spectrometry and textmining.

We have shown that the dynamics inherent to boosting offer ideal means to evaluate features utilizing the training examples’ margins distribution using the margin fraction weight, and while a feature may have a large contribution ratio; it will not contribute to a good overall margin unless its conditional margin is also large.

Our experimental results showed that our methods, SBS-MF and SBS-AMF achieve equal or significantly better performance to other approaches on the data sets tested.
Chapter 5

Intrusion Detection System for Virtualization: Framework and Data

In this chapter, we discuss how to design a real intrusion detection system (IDS) of virtual server environments utilizing only information available from the perspective of the virtual machine monitor (VMM). We begin with a short overview of the concept of IDS and virtualization technology, and the related work. We then describe the main goals and advantages of designing such VMM IDS. Finally, we provide details about the proposed VMM IDS framework, including: data gathering feature extraction, feature selection and anomaly detection.

5.1 Intrusion Detection System

Intrusion Detection Systems are programs that look for suspicious or anomalous behavior on the computer system. When such behavior is found whether it is a real attack, unauthorized behavior or otherwise, it is usually flagged for closer inspection.

Traditionally, IDS can be categorized as one of two types: a host-based intrusion detection system (HIDS) or network-based intrusion detection system (NIDS). An HIDS resides on the system that is being monitored and thus has the advantage of a rich view of the internal workings of the system. The disadvantage with this approach is that a malware can determine the existence of the HIDS and subsequently compromise it or attempt to evade detection. For example, they may monitor system logs, system calls, user commands, and file accesses. An NIDS, on the other hand, performs intrusion detection from outside the target system, using information from the network flow. This makes it more resistant to attacks and evasion, but at the cost of poor visibility of the system. For
example, they may monitor incoming and outgoing packets' content, rate, and targets.

There are two main approaches to intrusion detection [59]:

i) Misuse detection – The behavior of the system is compared to patterns of known malicious behavior, or attack signatures. A weakness of this approach is its inability to detect new and previously unseen attacks, known as zero-day attacks.

ii) Anomaly detection – A profile of normal behavior is built and any deviation from this normal profile is flagged as a potential attack. While anomaly detection has the ability to detect zero-day attacks, it is also prone to false alarms, i.e., previously unseen normal behavior may incorrectly be identified as an attack.

Today, intrusion detection is one of the high priority and challenging tasks in many technologies, particularly, in virtualization technology. A short overview of virtualization is given next.

5.2 Virtualization

Virtualization technology allows us to create a range of interfaces at different levels of abstraction. For example, it allows us to execute two operating systems concurrently, sharing the same hardware, or execute one OS (e.g., Linux/Windows) created for a particular architecture (X86) on top of another OS (e.g., Sun Solaris) running on a completely different Instruction Set Architecture (ISA) (e.g., SPARC). A good example of a popular virtual machine environment is VMware’s ESX [102].

In their recent text, Smith and Nair define a taxonomy for virtual machines [92] (as shown in Table 5.1). The taxonomy differentiates VMs according to the scope being virtualized, (i.e., process VMs vs. system VMs), as well as the ISA presented by the VM (i.e., the virtual ISA).

We elaborate on the benefits of the various VM approaches and discuss associated tradeoffs in the following sections.

5.2.1 Process VM

A process VM creates a virtualized application binary interface (ABI) environment which can be used by user applications [92]. Process VMs are very commonly used and are supported in most current multiprogrammed systems. These systems actually create a process VM for each application. Each application is given its own memory, an illusion of its own CPU (running the same ISA) and a set of system calls (an ABI) which allow it to access hardware resources such as disks and networks. The OS effectively creates a private virtual environment for each application on the system.

Binary optimizers are also a form of a process VM. The Dynamo system [14, 22], which is used in this thesis, allows us not only to virtualizes a process but also to optimize its execution. Binary optimizers are able to profile the execution of an application during run-time and optimize the binary
on-the-fly. In Dynamo’s case, the source ISA and target ISA are the same and its main objective is to optimize execution.

Another class of process VMs are Dynamic binary translators such as Digital’s FX!32 [25]. These translators allow applications compiled for one ISA to execute on a different one. FX!32 allows applications compiled for Windows NT on X86 to run on an Alpha Windows NT system transparently. The beauty of utilizing the same operating system was that all system calls and operating system system execution could be run natively on the Alpha. This provided for more competitive performance and lower overhead due to translation.

The Java VM and Microsoft .Net frameworks can also be classified as process VMs that execute binaries compiled for one ISA on a different ISA. In both cases a new Virtual ISA is introduced. The Virtual ISA provides for many advantages and enables the model of compile once, run anywhere. Applications are compiled into a binary form (bytecode) targeting the virtual ISA, and are then executed (interpreted or just-in-time compiled or jitted) by the VM. Java and .Net runtime systems provide a lot of support for the application including threading, security, garbage collection, and more. This support allows the VM to present an application binary interface that is independent from the underlying OS and ISA, and in turn increases the portability of the binaries - the binary can execute wherever the VM is installed, regardless of the underlying OS and ISA.

5.2.2 System VM

System VMs create a virtualized system (architectural interface, IO, etc.) environment that is used by operating systems. For some system virtual machines, the goal is to create a virtualized system with an ISA that differs from the underlying ISA. In those systems, hardware emulation or binary translation is necessary. Bochs [18] and Simics [91] are examples of this type of system VM. The System VM creates a virtual system by emulating each one of the system components (i.e., CPU,
memory, network adapter, etc.). For example, to effectively virtualizes a hard disk the VM can provide an API identical to that of a real hard disk. Instead of accessing a real hard disk the VM can use a large file to mimic the behavior of the hard disk and map each virtual sector to a unique location in the file. This method is very powerful since it allows us to emulate hardware which is not available. However, it does come with a price: lower performance.

It is also possible to create system VMs that are running on the same ISA. Although these virtual machines are restricted to the underlying physical hardware, they are usually very efficient. Since the underlying hardware is identical to the virtualized architecture, much of the virtualized user and system code can run directly on the underlying hardware. This significantly reduces the overhead introduced by the VM. A major component in these VMs is the Virtual Machine Monitor (VMM). This component is responsible for managing the hardware resources across the different VMs, maintaining strict isolation, and enforcing fair resource allocation and usage.

5.3 Related work

Much work has been done in the area of host-based IDSs. We organize our discussion here according to the information, or *semantics*, utilized by the IDS:

i) Program-level IDS – An IDS that uses information available at the program/application abstraction level. This includes source code, static or dynamic information flow, and application execution state.

ii) OS-level IDS – An IDS that utilizes information available at the OS level such as system calls and system state.

iii) VMM-level IDS – An IDS that uses semantics and information available at the VMM-level. This includes architectural information.

A related characterization of IDSs can be found in the work done by Gao et al. [39]. They use the terms *white box*, *gray box*, and *black box* to refer to the class information available to the IDS. Black box systems only use system call information, white box systems include all information available including high program-level source or binary analysis, and gray box lies in between.

In our classification criteria, we consider a broader range of semantics available to the IDS. Program-level IDSs use information similar to that available in white or gray box systems. OS-level IDSs can use all system-level information available including (but not limited to) system calls (i.e., a black box system). VMM IDSs extend the characterization even further to include VMM-level information. We use this classification to contrast and compare current IDSs in the next sections, and to highlight the novelty of our own work.
5.3.1 Program-level IDS

Program-level IDS has all high level information available to it including source code, application data, application control flow, and more.

Wagner et al. [103] show how static analysis can be used to thwart attacks that change the runtime behavior of a program. They build a static model of the expected behavior (using system calls, call graph, etc.), and compare it to the runtime program behavior. In the work done by Kirda et al. [52], both static and dynamic analysis (including information leakage) is used to determine if behavior is malicious.

There has been a number of information flow tracking systems that fall into this category. These systems include static [31, 75] and dynamic [101] data flow analysis to extract program-level information available to the application only.

5.3.2 Operating system-level IDS

System calls have been used extensively to distinguish normal from abnormal behavior. One example is the work done by Kosoresow et al. [55]. In this work they use system call traces to find repeated system calls and common patterns, and store them in an efficient deterministic finite automaton. Then, during execution they compare and verify that all system call traces have been seen before.

Many other intrusion detection systems have used system call profiles to successfully detect malicious code. System call tracing can be done very efficiently and can provide much insight into program activities.

Stolfo et al. [94] use Windows registry accesses to detect anomalous behavior. The underlying idea is that while registry activity is regular in time and space, attacks tend to launch programs never launched before and change keys not modified since OS installation.

In the work by Oliveira et al. [29], a virtual machine is used to provide recovery from zero-day control-flow hijacking attacks. The attack detection mechanism involves augmenting every 32-bit word of memory and general purpose register with an integrity bit, used to determine when a vulnerability is being exploited.

5.3.3 Virtual machine monitor-level IDS

To clarify the novelty of our work and distinguish it from previous approaches that incorporate the VMM for intrusion detection, we define two classes of VMM-level intrusion detection systems:

- Hybrid VMM/OS IDS
- Pure VMM IDS

Hybrid VMM/OS intrusion detection systems utilize the VMM as a means to isolate and secure the IDS. However, they rely on OS-level information and therefore are not pure VMM IDSs. Pure
VMM intrusion detection systems, on the other hand, only use semantics visible to the VMM to perform intrusion detection. This limits the amount of information available to the IDS and poses a greater challenge.

Work on hybrid VMM/OS intrusion detection systems include the efforts of Laureano et al. [62]. In their work, a VM is used to isolate and secure the IDS outside the guest OS. The guest OS is a User Mode Linux [100] that is modified to extract system calls. Then, system call sequence analysis is used to perform anomaly detection. Zhang et al. [111] use a Xen VMM to intercept sequences of system calls that are analyzed to detect intrusions. Intel and Symantec present how a VMM can be used to isolate anti-virus software [46].

Garfinkel et al. present a Virtual Machine Introspection architecture [41] which is used to create a set of policies for intrusion detection. These policies include a lie detector (verify that the kernel state is consistent with user level view), a signature detector (scan guest memory), a memory access enforcer (mark code sections in the kernel as read-only through the VMM), and more. They use a special OS interface library (built on top of a Linux crash dump examination tool) to access information available at the OS level. Similarly, the VMwatcher system uses a VMM to export the guest OS raw disk and memory to a separate VM. A driver and symbols are used to compare memory views to detect rootkits and run an anti-virus software on the (exported) disk.

### 5.3.4 IDS comparison

In Table 5.2, we present our view of the trade-offs associated with the different IDS types according to the semantics available to them.

<table>
<thead>
<tr>
<th></th>
<th>Program &amp; IDS</th>
<th>OS &amp; Hybrid IDS</th>
<th>Pure VMM &amp; IDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantics</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Applicability</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Performance</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Ease of deployment</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Attack Resistance</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 5.2: IDS level comparison

We identify the following trends in IDS design. First, the more program-level semantics there are available to the IDS, the more accurately it is able to identify and classify the malware. Using fewer semantics (as in the case of a VMM-level IDS) can limit the effectiveness of an IDS and may restrict the guest environments that can be effectively secured. In this sense, the applicability of a VMM IDS to more general computing environments is lower than that of IDSs which have more information available to them.

Second, more semantics may also introduce more overhead (i.e., impacting performance, as extracting this information (if available) can introduce significant overhead (e.g., information flow...
A VMM IDS only requires that information be extracted from the VMM layer and therefore results in significantly less overhead. Since the VMM is the only element that needs to be modified to extract this information, it also has the advantage of a greater ease of deployment. In most VMMs, a majority of the information is already available through standard profiling interfaces (e.g., Vcenter Server in VMware’s ESX).

Third, extracting program-level semantics may be done by modifying or monitoring the execution of the application. Frequently, this is done at runtime in the application address space (e.g., using binary instrumentation). While these techniques are able to protect the IDS from the application they monitor, they are still vulnerable to malware running at a higher privilege level. For example, a root-kit may thwart both application-level and OS-level IDSs. In contrast, malware would first have to detect that an application or operating system is virtualized before a successful corruption could be launched against a VMM-level IDS. Hence, a VMM-level IDS has a higher resistance to a malicious attack.

5.4 Proposed VMM IDS

There is a need to safeguard these VMMs from known vulnerabilities and at the same time take steps to detect new and unseen, but possible, system abuses by developing more reliable and efficient intrusion detection systems (IDSs). A VMM IDS has some inherent requirements. Its prime purpose is to detect most of the attacks, give very few false alarms, copes with large amount of data, and is fast enough to make real-time decisions.

Goals

In this work we propose VMM-based IDS, a variant of host-based intrusion detection systems wherein the IDS resides on the physical host machine, yet remains outside of the virtual machine being monitored. As such, a VMM IDS is able to enjoy the advantages offered by both HIDSs and NIDSs: a rich view of the target system (the VM) combined with a greater resistance to attacks and evasion by the malware. The latter is one of the benefits of isolation provided by the VMM.

The VMM IDS only uses information available at the VMM-level to detect intrusions. There exists a large semantic gap between this low-level architectural data and the actual program behavior. Consequently, we utilize sophisticated data mining algorithms to extract meaningful and useful information to distinguish normal (non-malicious) from abnormal (malicious) behavior.

Advantages

A key advantage that a pure VMM-level IDS provides is ease of deployment. Only the VMM needs to be modified to extract low-level architectural events during runtime. This ties the IDS to a particular VMM and instruction set architecture (ISA). No modification to the operating system is
required. Hence, it can be deployed in any virtualized computing environment with minimal effort. In our work, we focus on virtualized server applications. These applications are combined with a customized commodity operating system to run optimally in a virtual environment. As there are no login operations and typical execution consists of one main process running alongside background processes, we expect the normal behavior of these workloads to be fairly stable in time and space. Our IDS uses data mining algorithms to characterize the normal behavior of the workload. A malicious attack would introduce deviations from the normal behavior, which should be identified by the data mining algorithms and flagged by the IDS. Along these lines, a VMM IDS has the advantage of being able to detect zero-day attacks, in addition to previously known malware.

Potential issues

Data for online VMM IDS analysis requires collecting an enormous amount of traffic records with number of various features. Among the large amount of features, some of the features may be irrelevant with poor prediction ability to the target patterns, and some others may be redundant due to the high inter-correlation with one of more of the other features. If irrelevant and redundant features are involved in the analysis, not only the detection speed becomes slow but also the detection accuracy possibly decreases. Moreover, we are commonly faced with highly imbalanced intrusion data. For intrusion data, the proportion of traffic that corresponds to an attack is considerably smaller than the proportion of normal traffic. When the underlying model is trained with unbalanced data, existing feature selection techniques in general perform poorly because they tend to concentrate on the larger classes and disregard the ones with few samples [36].

The ability to detect unknown attacks makes anomaly detection systems advantageous over signature based detection systems. However, anomaly detection systems tend to be inefficient (i.e., computationally expensive) during both training and evaluation. This prevents them from being able to process huge amount of traffic and detect intrusions in real time.

Framework

The problems mentioned above not only hinder the detection speed but also have significant impacts on the detection performance of an IDS. Here, we propose to develop a novel VMM IDS framework in order to successfully solve these problems mentioned above. The framework shown in Figure 5.1 consists of three phases:

i) **Pattern Builder phase** – we first gather enormous monitored raw data at the VMM layer. However, these data are in a form that is not suitable for mining processing. Therefore, in the Pattern Builder we preprocess and convert them to features vectors.

ii) **Pattern Selector phase** – here we apply the SBS-AMF algorithm to select informative features.
iii) **Detector phase** – having reduced the complexity of the original data set, the compact data set is fed into the Detector phase for the task of identifying intrusions. In this phase, we propose to use the Graphics Processing Unit (GPU) platform in our intrusion detection design to accelerate the process of analyzing huge data size, and thus speed up the VMM IDS detection process. Given the recent popularization of GPUs, and the increased flexibility of the most recent generation of GPU hardware combined with high-level GPU programming languages such as CUDA [78], we incorporate a GPU-enabled CUDA anomaly detection implementation into our intrusion detection design for the LOF algorithm [21]. This algorithm is chosen for its merit of identifying attacks based on the deviations from the established profiles of normal activities, and the assumption that attacks deviate from normal behavior. The system then raises an alert when it detects any intrusion using the patterns built in the Pattern Selector phase.

### 5.5 The Pattern Builder Phase

A typical virtualization architecture consists of different levels, as shown in Figure 5.2. These levels include: the guest OS where the guest software runs within a VM and the VMM.

We collect enormous amount of raw and low-level architectural data from the perspective of the VMM layer. Such IDS can harness the ability of the VMM to isolate and manage several virtual machines, making it possible to provide monitoring of intrusions at a common level across VMs.

It also offers unique advantages over recent advances in intrusion detection for virtual machine environments [62, 41]. By working purely at the VMM-level, the IDS does not depend on structures or abstractions visible to the OS (e.g., file systems), which are susceptible to attacks and can be modified by malware to contain corrupted information (e.g., the Windows registry). In addition, being situated within the VMM provides ease of deployment as the IDS is not tied to a specific operating system (i.e., supports different versions of both Windows and Linux without any modifications).

As part of our contributions, we have implemented a prototype of our VMM IDS using VirtualBox [45], an open-source full-virtualization VMM.

#### 5.5.1 Raw data

We use the term *events* to describe the raw data and information extracted from the VMM during execution. These events are *architectural events* that the VMM must intercept to guarantee correct
Figure 5.1: VMM IDS high-level design.
Table 5.3: Summary of the Virtual events, the additional information extracted for the events, and what they mean for the OS.

<table>
<thead>
<tr>
<th>Virtual Event</th>
<th>Additional Information</th>
<th>High-Level Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk IO</td>
<td>Disk sector and Number of bytes</td>
<td>Disk read/write event</td>
</tr>
<tr>
<td>Network IO</td>
<td>Number of bytes</td>
<td>Network read/write event</td>
</tr>
<tr>
<td>Control Registers</td>
<td>Register number and Value</td>
<td>Control register read/write event</td>
</tr>
<tr>
<td>Page Faults</td>
<td>Error code and Virtual EIP register</td>
<td>Hints at memory usage and application behavior</td>
</tr>
<tr>
<td>TLB Flush</td>
<td>Global flag and New CR3 register value</td>
<td>Occurs during a context switch</td>
</tr>
<tr>
<td>Invalidate Page</td>
<td>Page to invalidate</td>
<td>Hints at memory and application behavior</td>
</tr>
<tr>
<td>Current Privilege Level</td>
<td>Privilege level</td>
<td>Hints at the type of code running; whether in user mode or supervisor mode</td>
</tr>
<tr>
<td>Load Segment Descriptor</td>
<td>Segment register, base and limit</td>
<td>Hints at application startup</td>
</tr>
<tr>
<td>Programmable Interrupt Timer</td>
<td></td>
<td>Means by which the OS can track time or Real Time Clock</td>
</tr>
</tbody>
</table>

execution. They include execution of privileged instructions, accesses to shared resources (memory), and IO (disk, network, devices). For example, when a disk IO event occurs, the VMM intercepts the request and we record the disk sector accessed, the number of bytes accessed, and read/write status.

The data available to the VMM dictate the types of events that we are able to monitor. The richness of the events, in terms of the wealth of information they provide to the VMM, is critical to the success of a VMM-based IDS. Consequently, a vast array of events must be collected in order to reconstruct a more accurate picture of what occurs at the application level.

Events can be categorized into three main types:

i) **Virtual or VM events**: architectural-level and system events related to the virtualized guest OS executing inside the VM. For example, a guest modifying control registers, flushing the TLB, or writing to the disk.

ii) **VMM events**: these events are extracted from the VMM and relate to the state of the VMM itself.
iii) **Real events**: these events can be extracted from within the VMM or from the host OS. The semantics of these events are related to the host OS. The real time clock is an example of such an event.

For some events, in addition to determining when they occur, we can also extract useful information about them. For example, during a disk IO event, we can determine information such as the disk sector, number of bytes accessed, and whether the disk access was a read or write event. A summary of the Virtual and VMM events that we extract is provided in Table 5.3.

In the table, we also list additional useful information that can be extracted for the events. The events we are interested in gathering should be able to hint at the underlying behavior of the system to allow us to distinguish changes in the system behavior. These *high-level semantics* of the events are provided in the last column of the tables.

### 5.5.2 Feature generation

We use the extracted events as guidelines to construct temporal statistical features for building classification models. Raw events output are first summarized into records using pre-processing programs, where each record has a set of features. This is done by dividing the stream of collected events into consecutive windows of equal (virtual) time, i.e., by using the event *Timer*. The length of the window introduces a trade-off: longer windows capture more behavior while shorter ones reduce the time-to-detection. We have found that windows of approximately 2 seconds long provide a good balance between these trade-offs. This time quantum provides the back-end with sufficient information to make accurate classifications on a per-window basis, while also allowing it to identify any malicious activity within seconds of its execution.

Each window $x$ (i.e., record or sample) is then represented by a vector of features $(v_1, \ldots, v_n, \ell)$. The $v_i$’s are input features and $\ell$ is the label $\ell \in \{-1, 1\}$. If a window contains *any event generated by any malicious process*, then the entire window is tagged as “malicious” i.e., “1”; only if a window contains no event generated by any malicious process it is tagged “non malicious” i.e., “-1”.

We consider constructing two types of features: 1) basic features, and 2) complex features.

#### Basic features

These features are simply constructed by counting the frequency of each event in each window $x$. We build multiple frequency features such as page-fault count, control register modifications, disk and network IO accesses, etc. These features can be constructed efficiently and provide rich information about the execution behavior of the system. Figure 5.3 illustrates an example of transforming events into the basic features.
Complex features

These features are built with the intention of capturing information not covered by the basic features. This may happen, for example, when different events have the same rate across windows. We are able to differentiate these windows by accounting for the order that particular events took place. For example, we are able to detect that a sequence of disk IO read events which is followed by a sequence of disk IO write events is different than a sequence of interleaving read and write disk IO events. This type of features can be constructed using the event occurrence, as well as event value(s). In Figure 5.4 we show some of the information we extract from event occurrences to build a complex feature. We plot a line corresponding to the order of occurrences of two events: for each event $A$ we add 1 and for each event $B$ we add $-1$. In this Figure the $x$-axis plots the events $A$ and $B$ in order, while $y$-axis corresponds to the difference between the number of occurrences of events $A$ and $B$ (Number of events $A$ - Number of events $B$). Different statistical approaches are used to generate the complex features, such as:

Figure 5.3: Example of constructing basic features.
i) **Max Discrepancy**: the largest difference between the respective occurrences of events $A$ and $B$ within any sub-window. Example shown in Figure 5.4.

ii) **Average run value**: the average length.

iii) **Max run value**: the maximum value in the window.

iv) **Min run value**: the minimum value in the window.

v) **Wilcoxon-Mann-Whitney** statistic test: is a measure of how randomly interleaved events $A$ and $B$ are within the window. For example, let’s consider two mutually independent pairs of windows $x_m$ and $x_n$ randomly sampled of sizes $m$, and $n$, denoted by $a_1, a_2, \cdots, a_m$ and $b_1, b_2, \cdots, b_n$, drawn from continuous population $D_A$ and $D_B$, respectively. Assume that the two random samples are drawn from the two populations with the same distribution shape but possibly different locations. The Mann-Whitney $U$ statistic is defined as the number of times a $B$ proceeds an $A$ in the combined ordered arrangement of two independent random samples. A symbolic representation of the Mann-Whitney $U$ statistic is

$$U = nm + \frac{m(m + 1)}{2} - \sum_{i=n+1}^{m} R_i$$  \hspace{1cm} (5.1)$$

where $R_i$ are the ranks.
5.5.3 Feature framework design and implementation

Any feature-based IDS should be designed and implemented to allow for efficient construction of features. Our framework was designed with the following goals:

i) Support online construction – We design our framework to support execution driven feature construction.

ii) Minimal storage requirements – To support online construction and minimize performance impact, the framework does not store events or features. Once a feature is completed it is handed over to the user and discarded. The framework includes the minimal state necessary to build features online.

iii) Generic, simple and extendible – The framework is designed to enforce a generic structure (and API) which we expect will simplify feature design. It also enables us to extend the framework with different types of features easily.

iv) Support event and feature labeling – Fine-grained event and feature labeling (described above) allows us support feature selection and detection algorithms training and testing.

5.6 The Pattern Selector Phase

In this phase, we apply the SBS-AMF proposed method (described in Chapter 3) for selecting feature subsets that capture the high skew present in the intrusion data. This gives us the ability to understand which features and consequently, which VMM-level events, are significant in modeling the behavior of normal workload execution and distinguishing irregular, malicious activity. We first discuss related work in feature selection, and then lay out the issues and problems encountered in the VMM-IDS constructed features. Finally, we describe how to employ the SBS-AMF with the VMM-IDS features. proposed method.

5.6.1 Related work to feature selection for IDS

Feature selection allows one to deal with high-dimensions, retaining only the features important for the classification task. By reducing the feature space, the efficiency, accuracy, and comprehensibility of the IDS can be improved significantly. A feature selection technique can be categorized as either being a filter, a wrapper approach or embedded methods [54] . The wrapper approach uses feedback from the final learning algorithm to guide the search for the set of features. Examples of using wrapper method for increasing the detection rate and decreasing the false alarm rate in a network intrusion detection task are, the work of Stein et al. [93], who used genetic algorithm to select a subset of features with C4.5 algorithm, and the work of Mukkamala and Sung [74], they applied both Support Vector Machines (SVM) and Support Vector Decision Function Ranking Method
(SVDFRM) to rank important input features for intrusion detection. Generally, wrapper algorithm ensures selection of a good set of features tailored for the learning algorithm but has the disadvantage of being time consuming because it involves learner accuracy ascertained from cross-validation runs. It thus becomes unpractical to apply a wrapper method to select features from a large data set that contains numerous features and instances [16]. Filters are seen as data pre-processors and generally do not require feedback from the final learner. As a result they tend to be faster, however, because they do not look for interactions between features, they can only select a set of strong features and not an optimal feature set. Example of employing the filter approach to intrusion detection work can be found in the work of Qu et al. [83]. They applied pair-wise correlation analysis to uncover mutual information between each feature and the decision class.

Embedded methods are fairly similar to wrappers in that they choose a subset of features that best predict the outcome. The difference is that while wrappers go around the learning machine and must retrain the method for each potential subset, embedded methods work in conjunction with the learning machine to choose the best feature subset. This helps avoid multiple iterations of retraining. The most popular embedded method is the recursive feature elimination algorithm built into the SVM [43].

Most feature selection algorithms based on boosting methods fall into the category of embedded methods, and have been used with great success in many applications like face recognition [90, 32], text mining [105] and intrusion detection [11, 44]. In the view of boosting as a feature selection, an interesting approach to use is boosting with a one-level decision tree, known as a decision stump, as the learning algorithm [106]. Constructing such a learner involves selecting a single feature, based on its ability to discriminate between classes [2]. The two main related works are the AdaBoost-CR and BDSFS described in Chapter 2.

In this work, however, we show that standard evaluation statistics, such as error rate [98, 90, 32], and information gain [30] are misleading metrics when boosting is trained with highly skewed data, since it tends to be overwhelmed by the majority classes and ignores minority classes. A utility function, such as AUC, provides a better method for classifier evaluation since it assigns different error costs for majority and minority (i.e., positives and negative) examples. ROC evaluation has been widely used as an evaluation measure for imbalanced data sets [50, 23].

While the ROC curve has been extensively used for evaluating IDS performance, since it describes the relationship between two cost metrics, detection rate and false alarm rate, it has not been exploited to evaluate features for intrusion detection domain.

### 5.6.2 VMM IDS features

As described in 5.5.2 we construct two types of features: basic features, and complex features. Tables 5.4 show examples of basic features (rate or count features), and Table 5.5 show examples of the complex features constructed. The total number of features constructed is 300.
Table 5.4: Examples of basic features constructed for the VMM-IDS.

<table>
<thead>
<tr>
<th>Feature Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS-RATE-CR0-WRITE</td>
</tr>
<tr>
<td>SYS-RATE-CR3-WRITE</td>
</tr>
<tr>
<td>SYS-RATE-CR4-WRITE</td>
</tr>
<tr>
<td>SYS-RATE-TRAP-RATE</td>
</tr>
<tr>
<td>SYS-RATE-PAGE-FAULT</td>
</tr>
<tr>
<td>SYS-RATE-PAGE-FAULT-P-0</td>
</tr>
<tr>
<td>SYS-RATE-TLB-FLUSH</td>
</tr>
<tr>
<td>SYS-RATE-INVALIDATE-PAGE</td>
</tr>
<tr>
<td>SYS-RATE-GDT-WRITE</td>
</tr>
<tr>
<td>SYS-RATE-TSS-WRITE</td>
</tr>
<tr>
<td>SYS-RATE-CPL-SET-RATE</td>
</tr>
<tr>
<td>VMM-RATE-SET-GUEST-TRAP-HANDLER-RATE</td>
</tr>
<tr>
<td>VMM-RATE-RESCHEDULE-EXECUTION-MODE-RATE</td>
</tr>
<tr>
<td>:</td>
</tr>
</tbody>
</table>

Table 5.5: Examples of complex features constructed for the VMM-IDS.

<table>
<thead>
<tr>
<th>Feature Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS-REL-CR3-WRITE-VS-CR0-WRITE[minRunValue]</td>
</tr>
<tr>
<td>SYS-REL-TSS-WRITE-VS-TLB-FLUSH[maxDiscrepancy]</td>
</tr>
<tr>
<td>SYS-REL-PAGE-FAULT-P-0-VS-PAGE-FAULT-P-1[minRunValue]</td>
</tr>
<tr>
<td>SYS-REL-PAGE-FAULT-WR-0-VS-PAGE-FAULT-WR-1[maxDiscrepancy]</td>
</tr>
<tr>
<td>SYS-REL-PAGE-FAULT-RSVD-0-VS-PAGE-FAULT-RSVD-1[maxDiscrepancy]</td>
</tr>
<tr>
<td>SYS-REL-PAGE-FAULT-ID-0-VS-PAGE-FAULT-ID-1[maxDiscrepancy]</td>
</tr>
<tr>
<td>SYS-REL-GDT-WRITE-VS-CPL-SET-0[maxDiscrepancy]</td>
</tr>
<tr>
<td>SYS-REL-DISKIO-READ-VS-DISKIO-WRITE[maxDiscrepancy]</td>
</tr>
<tr>
<td>SYS-REL-DISKIO-READ-VS-NETWORKIO-WRITE[MannWhitney-U1]</td>
</tr>
<tr>
<td>SYS-REL-NETWORKIO-READ-VS-NETWORKIO-WRITE[maxRunValue]</td>
</tr>
<tr>
<td>VMM-SYS-REL-EXE-MODE-ENTER-REM-VS-PAGE-FAULT-US-0[minRunValue]</td>
</tr>
<tr>
<td>VMM-SYS-REL-EXE-MODE-ENTER-REM-VS-CPL-SET-0[MannWhitney-U2]</td>
</tr>
<tr>
<td>VMM-REL-EXE-MODE-ENTER-REM-VS-EXE-MODE-ENTER-RAW[MannWhitney-P2]</td>
</tr>
<tr>
<td>:</td>
</tr>
</tbody>
</table>
5.6.3 Applying SBS-AMF to the VMM-IDS data

To perform feature selection we consider features as the weak classifiers for boosting. We construct a set of weak classifiers by considering decision stumps [37, 106]. The properties of individual features are assessed indirectly by simple thresholding in one-dimensional subspace. Let

\[ h_\theta(x) = \begin{cases} 
+1, & \text{if } x \geq \theta \\
-1, & \text{otherwise}
\end{cases} \]

where \( \theta \in \mathbb{R} \) is a threshold and \( x \in \mathbb{R} \) is a sample value corresponding to one of the features.

For example, the decision stump \((\text{PFrate},16)\) would output “malicious” (+1) for any window (sample) \( x \) where the page fault rate (\( \text{PFrate} \)) feature equaled or exceeded 16, and it would output “non-malicious” (−1) otherwise; furthermore, if \( d \) distinct page fault rates were encountered (e.g., 2, 4, and 6), then \( d + 1 \) decision stumps cover all possible threshold classifiers for this feature (e.g., (\( \text{PFrate},1 \)), (\( \text{PFrate},3 \)), (\( \text{PFrate},5 \)), and (\( \text{PFrate},7 \)), where the first and last decision stumps trivially output “malicious” or “non-malicious” for all windows).

To employ SBS-AMF, the following methodology is performed:

I. **Input**: Given a labeled training set \( S = \{ (x_i, \ell_i) \}_{i=1}^m \), where \( x_i \in \mathcal{X} \) are samples, \( \ell_i \in \{1, -1\} \) are labels, \( m = q + p \), and \( q \) and \( p \) are numbers of negative “non-malicious” and positive samples “non-malicious”, respectively. Let \( T \in \mathbb{N} \) be the number of weak classifiers to combine.

\[ \Upsilon = \{ v_1, v_2, \cdots, v_n \} \quad // \text{subset of surviving features} \]

\[ \mathcal{F} = \phi \quad // \text{feature ranked list} \]

II. **Train** AUCBoost algorithm using the subset of surviving features \( \Upsilon \)

while (\( \Upsilon \neq \phi \)) do

(a) **Initialize**: \( D_1(i) = 1/m, D_1^+(i) = 1/2p, D_1^-(i) = 1/2q \) for all \( i = 1, \cdots, m \)

Initial weights have a strong influence on the classification error costs. In real intrusion detection applications, almost all behaviors are normal. A high false-alarm rate wastes resources, as each alarm must be checked. The adjustable initial weights are for making a tradeoff between the false-alarm and detection rates.

(b) for \( t = 1, \cdots, T \) do

\[ \mathcal{H} = h_t : \mathcal{X} \rightarrow \{-1, 1\} \]
\[ h_t^* = \arg\max_{h_t \in H} \text{AUC}_t \], where \( \text{AUC}_t^* = 1 - \frac{1}{2} (\epsilon_t^+ + \epsilon_t^-) \)

The higher the AUC, the more useful is the hypothesis for classifying the training examples. \( \epsilon_t^+ \) and \( \epsilon_t^- \) are the false classification and the false-alarm rates with respect to \( \mathcal{D}_+ \) and \( \mathcal{D}_- \) over \( \mathcal{S}_+ \) and \( \mathcal{S}_- \), respectively and are defined as:

\[
\epsilon_t^+ = \sum_{i=1}^q D_t^+(x_i) I[(h_{(v,j)}(x_i) \neq +1)] \tag{5.2}
\]

\[
\epsilon_t^- = \sum_{i=1}^q D_t^+(x_i) I[(h_{(v,j)}(x_i) \neq -1)]
\]

where

\[
I[\gamma] = \begin{cases} 
1, & \gamma = True \\
0, & \gamma = False
\end{cases}
\]

\[ \alpha_t^* = \frac{1}{2} \log \left( \frac{1-\epsilon_t^+ - \epsilon_t^-}{\epsilon_t^+ + \epsilon_t^-} \right) \]

One can view \( \alpha \) as a measure of one’s “confidence” in the corresponding weak hypothesis. The higher \( \alpha \), the better the corresponding weak hypothesis is on the distribution on which it was trained, and implicitly, the better the underlying feature. By balancing the distributions that are constructed so that both classes are equally likely. Our method tries to select the decision stumps (and associated features) that perform well on both “normal” and “malicious” examples.

\[
\mathcal{D}_{t+1} = \frac{D_t \exp(-\alpha_t h_t^*(x_i))}{Z_t}
\]

\[
\mathcal{D}_{t+1}^+ = \frac{D_t^+ \exp(-\alpha_t h_t^*(x_i))}{Z_t^+}
\]

\[
\mathcal{D}_{t+1}^- = \frac{D_t^- \exp(\alpha_t h_t^*(x_i))}{Z_t^-}
\]

where \( Z_t, Z_t^+, Z_t^- \) are normalization constants, and \( \sum_{i=1}^m \mathcal{D}_{t+1}(i) = 1, \sum_{i=1}^p \mathcal{D}_{t+1}^+(i) = 1, \sum_{i=1}^q \mathcal{D}_{t+1}^-(i) = 1 \)

for ( \( j : v_j \in \mathcal{\Upsilon} \) ) do

\[
\psi_v = \sum_{j=1}^{N_v} \frac{\alpha_{(v,j)}}{\sum_{j=1}^{N_v} \alpha_j}
\]

\[
AMF_v = \psi_v \left( \frac{\sum_{j=1}^{q_v} \rho_v(x_j) + \sum_{j=1}^{p_v} \rho_v(x_j)}{\sum_{j=1}^{q_v} \rho(x_j) + \sum_{j=1}^{p_v} \rho(x_j)} \right)
\]

endfor

\[ j^* = \arg\min_{j : v_j \in \mathcal{\Upsilon}} (AMF_v) \] // find the worst feature with smallest AMF weight
\[ F = F \cup v_j, \quad \text{// update feature ranked list} \]
\[ \Upsilon = \Upsilon \setminus v_j, \quad \text{// reduce the set of available features} \]

```
endwhile
```

III. **Output**: feature ranked list \( F \).

## 5.7 The Detector Phase

After reducing the dimensionality of the data set, we feed it into the Detector phase in order to identify intrusions. Once a subset of the features is selected, the data samples only need to contain feature-values that correspond to the selected features.

The goal of this study is to validate the hypothesis that information sufficient to build an accurate IDS is present in the events we monitor at the VMM layer and in the features we construct, as well as to determine the relative importance of these features. We validate our hypothesis by first testing our detection performance assuming that we have both normal and abnormal training samples—a two-class anomaly detection problem. In the second half of our study, we assume a more realistic scenario where we only have normal training samples—unsupervised learning.

### 5.7.1 Supervised learning

In this study, we propose to apply the boosting algorithm to intrusion detection [87]. The motivation for applying the boosting algorithm includes the following points:

i) It is one of the most popular machine learning algorithms. Its theoretical basis is sound, and its implementation is simple. It has been applied to many pattern recognition problems, such as face recognition [90].

ii) The algorithm corrects the misclassifications made by weak classifiers, and it is less susceptible to overfitting than most learning algorithms. Recognition performances of the Boosting-based classifiers are generally encouraging.

iii) Data sets for intrusion detection are a heterogeneous mixture of categorical and continuous types. The different feature types in such data sets make it difficult to find relations between these features. By combining the weak classifiers for continuous features and the weak classifiers for categorical features into a strong classifier, the relations between these two different types of features are handled naturally, without any forced conversions between continuous and categorical features.

iv) If simple weak classifiers are used, the boosting algorithm is very fast.
We refer to our proposed boosting-based algorithm for intrusion detection as EnhancedBoost, which is based on the AUCBoost (cf. Chapter 3). In our Enhanced boosting-based algorithm for intrusion detection, decision stumps [106] are used as weak classifiers. The decision rules are provided for both categorical and continuous features. Furthermore, we propose to evaluate the performance of the constructed weak classifiers using Receiver Operating Characteristic (ROC) curve. This is very suitable for intrusion detection because it is necessary to reduce the false-alarm rate rather than the mean error: In real applications, almost all behaviors are normal. A high false-alarm rate wastes resources, as each alarm must be checked. Here, we propose to use ROC analysis to adjust initial weights and make a tradeoff between the false-alarm and detection rates.

Computational complexity analysis

In the training stage, the computational complexity of the algorithm arises from the construction of the decision stumps and strong classifier. For the construction of the decision stumps, all samples should be searched for each feature; thus, the computational complexity for constructing the decision stumps is \( O(dM) \), where \( M \) is the number of samples, and \( d \) is the number of features, i.e., the number of decision stumps. There are \( T \) iterations for constructing the strong classifier. Therefore, in the training stage of the Boosting algorithm, the computational complexity is only \( O(dTM) \).

In the testing stage, the computational complexity of our Boosting algorithm is also very low. As discussed in above there is only one comparison operation in each decision stump for testing a sample; thus, the test time for each decision stump is extremely low. The strong classifier is a combination of decision stumps. As there are \( T \) iterations in the construction of the strong classifier, the computational complexity of testing a sample is \( O(T) \). Because \( T \) is commonly of the same order as the number of features, the test time for our algorithm is very low.

In short, our Enhanced Boosting-based algorithm possesses the lowest computational complexity in the published learning algorithms for intrusion detection. This property is very attractive and promising because the classifiers for intrusion detection should be retrained very quickly in practice, and fast detection is essential for an effective defense against intrusions.

5.7.2 Unsupervised learning

While supervised learning techniques have been used to build intrusion detection systems, such a strategy requires positive (malicious) training data, either known or synthetic. While such systems may accurately detect known malicious behavior, they may not adequately identify previously unseen attacks or only do so at high false alarm rates. Motivated by the results of our two-class anomaly detection system (supervised learning) that the data collected at the VMM layer is sufficient to distinguish normal from malicious behavior, we aim to build an IDS that is trained on only normal data. For this we employ a profiling approach that does not require or make assumptions about malicious behavior, other than that it is “different” from normal behavior. Such an approach fits well
with deployment in real production environments. For example, new servers are often “stress tested” for many days or weeks with actual or realistic workloads before they are deployed. During this period, the behavior of the (virtualized) server can be profiled, and substantially different behavior encountered post-deployment can be flagged as potentially malicious.

Various data mining and machine learning techniques have been applied to the design of IDSs. In this study, we employ two well known techniques to build our VMM-based IDS, each taking a different approach to creating a model of normal behavior and performing anomaly detection. They are the distance-based \( K \)-Nearest Neighbor (KNN) and the density-based Local Outlier Factor (LOF) algorithms.

Next, we briefly describe the usage of each technique.

**K-Nearest Neighbor (KNN) algorithm**

The typical method of classifying a test data point with the K-Nearest Neighbors (KNN) algorithm is to find the point’s \( k \)-nearest neighbors and use the labels of those points to assign a label to the test point. In this manner, the learning is performed in a supervised fashion (due to the use of labeled data).

We modify the algorithm to provide unsupervised (one-class) learning by performing two main phases: (1) a model-creation or profiling phase, and (2) an anomaly detection phase. The profiling phase of the algorithm consists of simply storing the vectors of training data points (in our case, timed-windows of normal activity). In the anomaly detection phase, each (validation or test) data point is assigned a score, or decision value, indicating how abnormal it is by calculating the sum of the distances to its \( k \)-nearest neighbors.

The farther a data point is with respect to its \( k \)-nearest neighbors, the more abnormal it is and the larger the decision value assigned to it. The distance between pairs of data points can be measured using different metrics, such as the Euclidean distance. Example of KNN algorithm is shown in Figure 5.5.

![Figure 5.5: Example of KNN.](image)

**Local Outlier Factor (LOF) algorithm**

We propose to use the Local Outlier Factor (LOF) [21] algorithm, which is a powerful outlier detection technique that has been widely applied to anomaly detection and intrusion detection systems. The LOF algorithm defines the notion of a local outlier in which the degree to which an object is an outlier is dependent on the density of its local neighborhood, and each object can be
assigned an LOF which represents the likelihood of that object being an outlier. LOF has been shown to perform well in detecting abnormal behavior in network intrusion detection systems [63], and it was able to identify several novel and previously unseen intrusions in real network data that could not be detected using other state-of-the-art intrusion detection systems such as SNORT [1].

The main goal of the LOF method is to assign to each object a degree of being an outlier. This degree is called the local outlier factor (LOF) of an object. It is local in the sense that the degree depends on how isolated the object is when compared to its surrounding local neighborhood. Intuitively, outliers are the data objects with high LOF values, whereas data objects with low LOF values are likely to be normal with respect to their neighborhood. In our case, outliers represent malicious data that needs to be detected.

Definition 5.7.1 Let $\mathcal{D}$ be a database, $p$, $q$, and $o$ are some objects in $\mathcal{D}$, and $k$ be a positive integer. The distance function (for example Euclidean distance) $d(q,p)$ denotes the distance between objects $p$ and $q$.

The algorithm for computing the LOF value of an object $p$ in a data set $\mathcal{D}$ has several steps.

Step 1 Computing ($k$-distance of $p$):
the $k$-distance of $p$, denoted as $k$-distance($p$) is defined as the distance $d(p,o)$ between $p$ and $o$ such that:
- for at least $k$ objects $o' \in \mathcal{D} \setminus \{p\}$ it holds that $d(p,o') \leq d(p,o')$ and
- for at most $(k-1)$ objects $o' \in \mathcal{D} \setminus \{p\}$ it holds that $d(p,o') < d(p,o)$

$k$-distance($p$), provides a measure of the density around the object $p$, when $k$-distance of $p$ is small that means the area around $p$ is dense and vice versa.

Step 2 Finding ($k$-distance neighborhood of $p$):
the $k$-distance neighborhood of $p$ contains every object whose distance for $p$ is not greater than the $k$-distance, is denoted as $N_k(p) = \{q \in \mathcal{D} | d(p,q) < k$-distance($p$)\}

Step 3 Computing the (reachability distance of $p$ w.r.t object $o$):
the reachability distance of object $p$ with respect to object $o$ is:

$$reach\_dist\_k(p,o) = \max\{k\text{-distance}(o), d(p,o)\}$$ (5.3)

Step 4 Computing (the local reachability density of $p$):
the local reachability density of an object $p$ is the inverse of the average reachability distance
from the k-nearest neighbors of $p$:

$$lrd_k(p) = \left[ \sum_{o \in N_k(p)} \frac{reach_k(p,o) - dist_k(p,o)}{|N_k|(p)} \right]^{-1}$$ (5.4)

**Step 5** Computing the local outlier factor of $p$:

the local outlier factor is a ratio that determines whether or not an object is outlier with respect to its neighborhood. $LOF(p)$ is the average of the ratios of the local reachability density of $p$ and that of $p$’s k-nearest neighbors.

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} lrd_k(o)}{|N_k|(p)}$$ (5.5)

In order to illustrate the idea of the LOF approach, consider the simple two-dimensional data set shown in Figure 5.7.2. There is a much larger number of items in cluster $C_1$ than in cluster $C_2$, and the density of the cluster $C_2$ is significantly higher than the density of cluster $C_1$. With our notion of a “local” outlier, we wish to label both objects $p_1$ and $p_2$ as outliers.

Due to the low density of the cluster $C_1$, it is clear that for every item $q$ inside the cluster $C_1$, the distance between the item $q$ and its nearest neighbor is greater than the distance between the item $p_2$ and the nearest neighbor from the cluster $C_2$, and the item $p_2$ will not be considered as outlier. Therefore, the simple nearest neighbor approaches based on computing the distances fail in these scenarios. However, the item $p_1$ may be detected as an outlier using only the distances to the nearest neighbor. Alternatively, LOF is able to capture both outliers ($p_1$ and $p_2$) due to the fact that it considers the density around the points.

**LOF computational overhead**

Unfortunately, the LOF algorithm’s time complexity is $O(n^2)$, where $n$ is the data size. It is designed to compute the LOF for all objects in the data set, which results in a computationally intensive process, since it requires a large number of k-nearest-neighbors queries. Because of this issue, designing efficient and reliable intrusion detection systems based on the LOF method is challenging.
5.8 Enhance VMM IDS Detection Speed

Encouraged by the recent popularization of Graphics Processing Units (GPUs), and the increased flexibility of the most recent generation of GPU hardware combined with high-level GPU programming languages such as CUDA, we recently developed a CUDA-LOF implementation described in [11]. This has motivated us to use the CUDA-LOF algorithm to accelerate the processing speed of our VMM IDS.

5.8.1 CUDA programming model

Computer unified device architecture (CUDA) [CUD] [79] is a parallel computing architecture developed by NVIDIA that allows programmers to take advantage of the computing capacity of NVIDIA GPUs in a general purpose manner. The CUDA programming model executes kernels as batches of parallel threads in a SIMD programming style. These kernels comprise thousands to millions of lightweight GPU threads per each kernel invocation. CUDA’s threads are organized into a two-level hierarchy represented in Figure 5.8.1 at the higher one, all the threads in a data-parallel execution phase form a grid. Each call to a kernel execution initiates a grid composed of many
thread groupings, called thread blocks. All the blocks in a grid have the same number of threads, with a maximum of 512.

The maximum number of thread blocks is \(65535 \times 65535\), so each device can run up to \(65535 \times 65535 \times 512 = 2 \times 10^{12}\) threads per kernel call. To properly identify threads within the grid, each thread in a thread block has a unique ID in the form of a three-dimensional coordinate, and each block in a grid also has a unique two-dimensional coordinate. Thread blocks are executed in streaming multiprocessors. A stream multiprocessor can perform zero overhead scheduling to interleave warps and hide the overhead of long-latency arithmetic and memory operations.

### 5.8.2 LOF CUDA implementation

#### Algorithm 10 LOF algorithm

**Input:** \(k\) (number of neighbors); \(\mathcal{D}\) (a set of data points)

**Output:** LOF values (a vector with local density factors)

**Assume:**

- \(k\) distance\((\mathcal{D}, p)\): return a matrix that contains the \(k\) – distances neighbors and their \(k\) distances
- \(reach\_dist\_k(p)\): return the local reachability density of each \(p \in \mathcal{D}\)

**Begin**

\[
\text{lof} \leftarrow \text{NULL}
\]

for each data point \(p \in \mathcal{D}\) do

- \(KN\text{Neibors} \leftarrow k\) distance\((\mathcal{D}, k)\)
- \(lrd \leftarrow reach\_dist\_k(\text{KN}\text{Neibors}, k)\)

for each \(p \in \text{KN}\text{Neibors}\) do

- \(temp\text{lof}[i] \leftarrow \sum(lrd[N(p)])/lrd[i])/|N(p)|\)
- \(lof \leftarrow \text{max(lof, templof)}\)

**End**

The pseudo code of the LOF algorithm provided in Algorithm 10 shows that LOF exhibits a high degree of data parallelism. Given this property, LOF seems to be ideally suited for a GPU implementation. The compute-intensive step of the LOF algorithm lies in computing the reachability distances defined as: \(reach\_dist\_k(p, o) = \max(k\_distance(o), d(p, o))\).

Computing reachability for \(p\) involves computing distances of all objects within \(p\)'s neighborhood. The runtime complexity of the algorithm is \(O(n^2)\) time for a KNN query, where \(n\) is the size of the database \(\mathcal{D}\). For the KNN queries, if a brute force search method is used, then that will lead to a complexity of \(O(n^2)\) for LOF. However, since the computation of the distances between any pairs of points in a data set are independent, this can be fully parallelized on a GPU.
Our LOF CUDA implementation consisted of multiple kernels, each of which is data dependent the other; we could have integrated the multiple small kernels into one big kernel, but the sorting step of the algorithm makes this kind of approach inefficient. This series of kernels are invoked inside a loop iteration, with each loop iteration processing a subset of the input data set that fits nicely in GPU memory. It is critical to perform a proper mapping of the data set to the GPU memory subsystem to obtain high performance [49].

To achieve high performance on a GPU using CUDA, we considered a number of optimization factors. During the course of our code development we applied two different classes of optimization techniques. First, we explore the best thread configuration space for maximal hardware utilization, based on the amount of shared memory used and the number of register used. Second, we employ different memory spaces for maximal efficient memory bandwidth, based on data access pattern of each input data. These two optimization techniques have been shown to be the most powerful optimization techniques in current CUDA programming [78, 79, 48]. In our case, we found that the use of shared memory (i.e., heavy data reuse case) and texture memory (i.e., random memory access pattern case) contributes significantly to overall speedup by increasing effective memory bandwidth and reducing the number of expensive off-chip memory accesses.

5.9 Chapter Conclusions

In this chapter, we discussed how to design real VMM-IDS, as virtualization is becoming an increasingly popular service hosting platform, utilizing only information available from the perspective of the VMM. Such IDS increases the ease of deployment across different operating systems and versions, and as part of a VMM, offers high manageability for server appliances. VMM-based IDSs break the boundaries of current state-of-the-art IDSs. They represent a new point in the IDS design space that trades a lack of program semantics for greater malware resistance and ease of deployment. The proposed VMM-IDS framework includes three phases: data gathering feature extraction, feature selection and anomaly detection. In the first phase, we construct different types of features from raw data (events) utilizing information embedded within the VMM level. In the second phase, we applied our proposed feature selection algorithm SBS-AMF to handle the presence of excessive features in an intrusion data whose class distributions are imbalanced. Finally, in the detection phase we implement two unsupervised anomaly detection algorithms, KNN and LOF, by training with normal data. We then, described how to develop a GPU-enabled CUDA implementation for the LOF method to accelerate the IDS framework. This application requires that we can perform both timely and accurate detection of anomalies. LOF is a very powerful outlier detection method; however, it is very computationally expensive. Thus, it becomes a challenging issue when it comes to designing an efficient and reliable IDS.
Chapter 6

Intrusion Detection System for Virtualization: Performance Evaluation

In this chapter, we provide evidence that an effective VMM IDS can be constructed. The chapter is organized in the following way. First we review our experimental setup. Next, we discuss the features reduction study and our observations. We then present the accuracy results we obtain using, first the supervised learning method (boosting), and then with unsupervised learning methods (KNN and LOF). Finally, we show how to accelerate the detection speed performance of the VMM IDS by applying a GPU parallel computing approach to accelerate the LOF algorithm.

6.1 Experimental Setup

As our VMM, we use VirtualBox [45], a hosted virtualization environment (i.e., the VMM runs on a commodity operating system). We use the open source edition 2.2, currently developed by Sun Microsystems as part of its Sun xVM virtualization platform. The target deployment for our VMM-based IDS is to secure virtual machine appliances. Each appliance is usually prepackaged with a commodity OS (Windows or Linux) and software stack that is configured to perform one or more specific tasks [76].

To test the effectiveness and robustness of our proposed VMM IDS, we use different classes of servers, virtual appliances, and workloads, chosen to reflect a broad range of behaviors (CPU intensive workloads, disk accesses, network IO, etc.), as shown in Table 6.1:
Table 6.1: Normal Workload (appliances).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Server</th>
<th>Virtual Appliance</th>
<th>Operating System</th>
<th>Workload(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Database</td>
<td>MySQL Server</td>
<td>Window XP (SP3)</td>
<td>On-line transaction processing benchmark (TPC-C like) [97]</td>
</tr>
<tr>
<td></td>
<td>management</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>system</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web</td>
<td>Web Server</td>
<td>Apache HTTP Server</td>
<td>Windows XP (SP3)</td>
<td>Apache bandwidth benchmark (ab) [12]</td>
</tr>
<tr>
<td>EMail</td>
<td>EMail Server</td>
<td>Exchange Server</td>
<td>Windows Server 2003 (SP2)</td>
<td>Microsoft Exchange Load Simulator (LoadSim) [69]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Malware Categories.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infostealer</td>
<td>a malicious program that steals online game accounts, such as Lineage, Ragnarok online, Rohan, and Rexue Jianghu</td>
</tr>
<tr>
<td>Downloader</td>
<td>a malicious program that connects to the Internet and downloads other components</td>
</tr>
<tr>
<td>Trojan Horse</td>
<td>a malicious program which disguised as something innocuous or desirable, users may be tempted to install it without knowing what it does</td>
</tr>
<tr>
<td>Backdoor</td>
<td>a method of bypassing normal authentication procedures. Once a system has been compromised, backdoors may be installed in order to allow easier access in the future. Backdoors may also be installed prior to malicious software, to allow attackers entry</td>
</tr>
</tbody>
</table>

i) **Database** dataset – consists of a TPC-C like workload, which is an on-line transaction processing benchmark [97]. It generates a pseudo-random sequence of client accesses that create a stream of random reads and writes.

ii) **Web** dataset – consists of **ab** workload, which is the Apache HTTP server benchmarking tool [12]. It is designed to measure performance in http requests per second.

iii) **EMail** dataset – consists of a mail server using Microsoft exchange (LoadSIM [69]), which is a benchmarking tool simulating clients of an Exchange server.
The malicious data are generated from recent malwares taken from Malfease, an online repository [67]. These malwares are categorized into four families based on their behavior as shown in Table 6.2. For each Normal_workload-Malware_family combination we extract 300 features, and the ratio of the two classes in the data (normal: abnormal) is 5 : 1.

For any data mining task, the dataset can be divided into three sets: a training set, a validation set, and a test set. The training set is used to build the data mining model. Since we build a model of normal behavior, our training set is comprised of only normal windows, i.e., those that represent time periods in which there is no malicious activity. Once the model is created, its performance is evaluated on a validation set. Our validation set contains both normal and abnormal data (i.e., windows). We evaluate how well the model of normal behavior is able to distinguish abnormal windows (true detections), while making sure that normal windows from the validation set are not incorrectly identified as malicious (false alarms). By tuning the parameters of the intrusion detection process, it is possible to achieve a reasonable trade-off between the true detections and false alarms. Thus, the validation set can be used to find good values for the unknown parameters. Once these parameters are chosen, they are evaluated on the test set to see how well the IDS (using those parameters) performs on previously unseen normal and abnormal windows.

To demonstrate the scalability and speedups of the GPU implementation, we compare it with a multithreaded CPU implementation developed in C (compiled with gcc 4.2). We run our GPU implementation on a NVIDIA GeForce GTX 285 GPU, using CUDA version 2.3. The CPU host system is equipped with Intel Core 2 Duo running at 2.66 GHz with 2 GB main memory, running Fedora 11 Linux operating system.

6.2 VMM IDS Feature Reduction Study

The main goal of our feature analysis and proposed selection method is to validate that sufficient information exists within the features constructed from events captured at the VMM layer in order to distinguish normal from abnormal behavior, especially, when the class imbalance ratio is high. As explained before, we used the AMF weights assigned by SBS-AMF to rank the importance of features.

6.2.1 VMM IDS features issues

As explained in Chapter 4, we create many basic and complex features. The complex features are designed to capture the interaction between different event types. This may result in many redundant features. Redundancy means that features are strongly correlated so that they do not contain much additional information. Example of such features is shown in plot (a) in Figure 6.1, which shows “SYS-RATE-CR3-WRITE” feature vs. “SYS-RATE-TSS-WRITE” feature. Nevertheless, correlation does not always imply absence of information. Consider an example as
plot (b) in Figure 6.1. Better class separation and noise reduction may be obtained by selecting both variables even though observations are correlated but under presence of noise, such as “SYS-RATE-CR0-WRITE” feature vs. “SYS-RATE-CR4-WRITE” feature.

Another example is when the first principal direction of the covariance matrices of the class conditional densities is the same, however, class centers are shifted similarly as in plot (c) in Figure 6.1. Although such features are correlated, they are not redundant, because selecting both provides more discriminatory information, such as “SYS-RATE-TLB-FLUSH” feature vs. “SYS-RATE-INVALIDATE-PAGE” feature.

Also, we construct complex features for every pair-event exists, such as network IO read Vs. network IO write. Since not all pair-event interaction is important, we may have many noisy and irrelevant features. Irrelevant features have completely overlapping class conditional probability.
density functions which does not bring any useful information. Our feature analysis show that “Load Segment Descriptor” event based features are useless, see plot (d) in Figure 6.1. However, not all features that seem to be irrelevant are completely useless.

6.2.2 Feature analysis

As described in Chapter 4, we apply the SBS-AMF algorithm to perform feature reduction for each of our workloads data. We list the top ranked 5 features along with their AMF weights (i.e., their importance) assigned by the SBS-AMF method. The list of features are described in 6.3, 6.4, and 6.5 for Database, Web and EMail workloads, respectively.

Based on these results we have made several observations:

i) Complex features overshadow basic features. We observed that the combined contribution of all basic features is less than 20%. Also, Complex features contain some of the information described by the basic features, which means that SBS-AMF is able to eliminate redundant features.

ii) We have compared the selected-features produced by SBS-AMF while training on different workloads and families of attacks. We observe similarities in the highly ranked selected features. In particular, we can identify these groups of them:

- GDT features – contains information about the global descriptor table.
- TLB Flush features – including CR3 events.
- Disk I/O, and Network I/O – describes the relationship between disk read and network transmits events.
- Page Fault features - contain information related to page faults and their error codes.
- VMM features – incorporate information related to the state of the VMM. o page faults, task switches, privilege level changes.

Our results show that he best feature is “SYS-REL-PAGE-FAULT-P-0-VS-PAGE-FAULT-P-1[MannWhitney-P1]” for the Database workload, which is disk-intensive, and as it generates accesses to random locations on the disk and that entails page faults due to the frequent swapping between memory and disk. For the Web workload, a network-intensive workload, the best feature is “SYS-REL-NETWORKIO-READ-BYTES-VS-DISKIO-WRITE-BYTES[MannWhitney-P1]” , which is a complex feature which shows the relationship network I/O events and disk I/O events. This is expected as network I/O and disk I/O events play a role in characterizing its normal execution. In the EMail workload, the best feature is “VMM-SYS-REL-EXECUTION-MODE-ENTER-REM-VS-PAGE-FAULT-US-0[MannWhitney-P1]”, which is a complex feature that applies the Mann-Whitney test to evaluate the random interleaving of
the following two events: (1) an event distinguishing that the VMM has entered a different state of execution wherein it intercepts system-level requests and recompiles code, and (2) page faults that occurred while executing in the supervisor mode.

The feature reduction phase show that not all extracted raw events are important and therefore constructing features only on frequently-used events can enhance the detection accuracy and speed of the VMM IDS.

Table 6.3: Top 5 selected features by SBS-AMF for Database workload.

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>AMF Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS-REL-PAGE-FAULT-P-0-VS-PAGE-FAULT-P-1[MannWhitney-P1]</td>
<td>0.348575</td>
</tr>
<tr>
<td>SYS-REL-NETWORKIO-READ-BYTES-VS-DISKIO-WRITE-BYTES[MannWhitney-P2]</td>
<td>0.109586</td>
</tr>
<tr>
<td>SYS-REL-CR3-WRITE-VS-TSS-WRITE[avgRunValue]</td>
<td>0.107570</td>
</tr>
<tr>
<td>VMM-SYS-REL-EXECUTION-MODE-ENTER-REM-VS-PAGE-FAULT-US-0[MannWhitney-U1]</td>
<td>0.100803</td>
</tr>
<tr>
<td>SYS-REL-GDT-WRITE-VS-CPL-SET-3[MannWhitney-U1]</td>
<td>0.086379</td>
</tr>
</tbody>
</table>

Table 6.4: Top 5 selected features by SBS-AMF for Web workload.

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>AMF Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS-REL-NETWORKIO-READ-BYTES-VS-DISKIO-WRITE-BYTES[MannWhitney-P1]</td>
<td>0.421997</td>
</tr>
<tr>
<td>SYS-RATE-DISKIO-READ</td>
<td>0.265953</td>
</tr>
<tr>
<td>SYS-REL-CR3-WRITE-VS-CR4-WRITE[avgRunValue]</td>
<td>0.059801</td>
</tr>
<tr>
<td>SYS-REL-CR3-WRITE-VS-CR0-WRITE[MannWhitney-P2]</td>
<td>0.105268</td>
</tr>
<tr>
<td>SYS-RATE-PAGE-FAULT-WR-1</td>
<td>0.016971</td>
</tr>
</tbody>
</table>

Table 6.5: Top 5 selected features by SBS-AMF for EMail workload.

<table>
<thead>
<tr>
<th>Feature Name</th>
<th>AMF Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMM-SYS-REL-EXECUTION-MODE-ENTER-REM-VS-PAGE-FAULT-US-0[MannWhitney-P1]</td>
<td>0.317701</td>
</tr>
<tr>
<td>SYS-REL-PAGE-FAULT-P-0-VS-PAGE-FAULT-P-1[MannWhitney-P1]</td>
<td>0.202670</td>
</tr>
<tr>
<td>SYS-RATE-DISKIO-WRITE</td>
<td>0.122796</td>
</tr>
<tr>
<td>SYS-RATE-NETIO-WRITE</td>
<td>0.101759</td>
</tr>
<tr>
<td>SYS-REL-TSS-WRITE-VS-TLB-FLUSH[maxRunValue]</td>
<td>0.057609</td>
</tr>
</tbody>
</table>
6.3 VMM IDS Detection Performance Evaluation: Supervised Learning

We compare the performance of EnhancedBoost to AdaBoost algorithm. The performance results shown in Table 6.6 are represented by the detection and the false alarm rates, and AUC. We utilize $k$-fold cross-validation to obtain the results. To determine number of round for EnhancedBoost algorithm, we vary number of round and report the resulted AUC values during training and testing phases for boosting as shown in Figure 6.2. The best number of rounds is when the algorithm starts to overfit (around 50).

From the performance results we can see that EnhancedBoost outperforms AdaBoost on all workloads and all families of attacks. This shows that an intrusion detection algorithm based on error rate, i.e., AdaBoost, is not robust to changes in the cost and class distributions of the underlying application domain. The results also varies across families of attacks; meaning that the distribution of the attacks in our datasets has an impact on the learning process. The results of detecting infrequent attacks (backdoors) are the worst compared to detecting frequent attacks (infostealer) and detecting moderate attacks (downloaders and trojans).

The detection rate seems to be different for the same type of family of attack across workloads. For instance 100% of “Trojan” malwares are detected for Web, while 95% of them are detected for Database with our detection algorithm. This shows that workload characteristics have an impact on the malware behavior.

It is noted that the implementation of our boosting-based intrusion detection algorithm is not amenable to incremental learning. When the system changes over time, the newly produced data should be labeled and merged with the previous sample data, and the classifier is retrained using the merged sample data. Although our algorithm has a very low computational complexity, which makes it possible to frequently retrain the classifier, offline learning is still a limitation when it is necessary to adapt to complicated and changing environments. Therefore, we employ a more realistic scenario where we only have normal training samples.
Table 6.6: Supervised learning results: a comparison of detection accuracy performance for each workload using EnhancedBoost, and AdaBoost.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>DataBase</th>
<th>EnhancedBoost</th>
<th>AdaBoost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>FP(%)</td>
<td>TP(%)</td>
</tr>
<tr>
<td>Downloader</td>
<td>1.24</td>
<td>96.21</td>
<td>94.97</td>
</tr>
<tr>
<td>Infostealer</td>
<td>0.98</td>
<td>98.74</td>
<td>97.76</td>
</tr>
<tr>
<td>Trojan</td>
<td>1.19</td>
<td>95.16</td>
<td>93.97</td>
</tr>
<tr>
<td>Backdoor</td>
<td>2.47</td>
<td>94.35</td>
<td>91.88</td>
</tr>
<tr>
<td>Ave.</td>
<td>1.47</td>
<td>96.12</td>
<td>94.65</td>
</tr>
</tbody>
</table>

| Web            |          | EnhancedBoost | AdaBoost |
|                |          | FP(%) | TP(%) | AUC(%) | FP(%) | TP(%) | AUC(%) |
| Downloader     | 0.94     | 97.89 | 96.95 | 2.98   | 87.51 | 84.53 |
| Infostealer    | 0.74     | 99.17 | 98.43 | 2.56   | 91.31 | 88.75 |
| Trojan         | 0.09     | 100   | 99.91 | 1.79   | 88.52 | 86.73 |
| Backdoor       | 1.86     | 96.74 | 94.88 | 4.55   | 83.19 | 78.64 |
| Ave.           | 0.91     | 98.45 | 97.54 | 2.97   | 87.63 | 84.66 |

| EMail          |          | EnhancedBoost | AdaBoost |
|                |          | FP(%) | TP(%) | AUC(%) | FP(%) | TP(%) | AUC(%) |
| Downloader     | 1.53     | 95.13 | 93.6  | 3.84   | 89.21 | 85.37 |
| Infostealer    | 0.86     | 99.2  | 98.34 | 1.34   | 87.12 | 85.78 |
| Trojan         | 0.75     | 99.31 | 98.56 | 1.82   | 88.97 | 87.15 |
| Backdoor       | 2.13     | 96.62 | 94.49 | 3.55   | 85.35 | 81.8  |
| Ave.           | 1.32     | 97.57 | 96.25 | 2.64   | 87.66 | 85.03 |
6.4 VMM IDS Detection Performance Evaluation: Unsupervised Learning

As described previously, we use SBS-AMF to select features that best determine which VMM features are significant in modeling the behavior of normal workload execution and distinguishing abnormal activity. On extremely imbalanced data sets, algorithms will be hard pressed to classify test samples as members of the minority class because the discriminant scores given by the classifier are often weighted toward the majority class. Accuracy is clearly a poor measure of the performance of a classifier on imbalanced data thanks to the trivial majority classifier. There are a number of statistics that researchers commonly use to focus on the minority class, such as AUC.

We compare the performance of SBS-AMF against AdaBoost-CR [98]. In simulation experiments, datasets were firstly fed into these feature selectors, which will generate different feature subsets from the same dataset. After that, datasets with newly selected features were passed to KNN and LOF algorithm to assess classification performance. Note, however, that the training phase for KNN and LOF consists of only profiling the normal workloads.

To achieve impartial results, stratified 10-fold cross validations had been adopted for each malware family-dataset combination in verifying detection capability. That is to say, for each dataset with a specific malware family we before and after feature selection, we run LOF algorithm on it 10 times and at each time a 10-fold cross validation was used, and the final results were their average and variance values. The results were evaluated using the three evaluation metrics described above.

The ROC results of the VMM IDS detection performance on the three workloads using two feature selection algorithms are shown in Figure 6.3, and are summarized in Tables 6.7 and 6.7 using the TP, FP and AUC.

The AUC results in Table 6.7 show that the performance of the VMM IDS using SBS-AMF-selected features are better than those selected-features of AdaBoost-CR on all datasets and families of attacks. This show that a feature selection based on error rate is not robust to changes in the cost and class distributions of the underlying application domain.

These results indicate that sufficient information exists in features selected by SBS-AMF to build a real IDS that is not susceptible to the characteristics of the attack behavior, or to specific workload.

To better understand the behavior of both feature selection methods, we generate scatter plots (shown in Figure 6.4) of classifying a Downloader malware with LOF using the selected feature of SBS-AMF shown in the top plot, and AdaBoost-CR shown in the bottom plot. Each + in these plots corresponds to “abnormal” sample, and each ◦ corresponds to “normal” sample. The y-axis is the confidence value assigned by LOF. As we can see, the abnormal windows with SBS-AMF features have higher confidence values of being outliers (i.e., marked as malicious) than with AdaBoost-CR features.
Figure 6.3: ROC results for the VMM IDS detection performance for each workload with KNN and LOF using two different feature selection methods.
Table 6.7: Unsupervised learning results: a comparison of detection accuracy performance of KNN for each workload using two feature selection algorithms.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>SBS-AMF</th>
<th>AdaBoost-CR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP(%)</td>
<td>TP(%)</td>
</tr>
<tr>
<td>Downloader</td>
<td>9.00</td>
<td>95.0</td>
</tr>
<tr>
<td>Infostealer</td>
<td>6.00</td>
<td>98.0</td>
</tr>
<tr>
<td>Trojan</td>
<td>8.00</td>
<td>93.0</td>
</tr>
<tr>
<td>Backdoor</td>
<td>4.00</td>
<td>94.0</td>
</tr>
<tr>
<td>Ave.</td>
<td>7.00</td>
<td>95.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Web</th>
<th>SBS-AMF</th>
<th>AdaBoost-CR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP(%)</td>
<td>TP(%)</td>
</tr>
<tr>
<td>Downloader</td>
<td>4.00</td>
<td>97.0</td>
</tr>
<tr>
<td>Infostealer</td>
<td>5.00</td>
<td>98.0</td>
</tr>
<tr>
<td>Trojan</td>
<td>2.00</td>
<td>100</td>
</tr>
<tr>
<td>Backdoor</td>
<td>2.00</td>
<td>96.0</td>
</tr>
<tr>
<td>Ave.</td>
<td>3.00</td>
<td>97.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMail</th>
<th>SBS-AMF</th>
<th>AdaBoost-CR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP(%)</td>
<td>TP(%)</td>
</tr>
<tr>
<td>Downloader</td>
<td>3.00</td>
<td>96.0</td>
</tr>
<tr>
<td>Infostealer</td>
<td>6.00</td>
<td>98.0</td>
</tr>
<tr>
<td>Trojan</td>
<td>4.00</td>
<td>92.0</td>
</tr>
<tr>
<td>Backdoor</td>
<td>2.00</td>
<td>98.0</td>
</tr>
<tr>
<td>Ave.</td>
<td>4.00</td>
<td>96.0</td>
</tr>
</tbody>
</table>
Table 6.8: Unsupervised learning results: a comparison of detection accuracy performance of LOF for each workload using two feature selection algorithms.

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>FP(%)</th>
<th>TP(%)</th>
<th>AUC(%)</th>
<th>FP(%)</th>
<th>TP(%)</th>
<th>AUC(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBS-AMF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Downloader</td>
<td>8.00</td>
<td>93.0</td>
<td>85.0</td>
<td>5.00</td>
<td>81.0</td>
<td>76.0</td>
</tr>
<tr>
<td>Infostealer</td>
<td>5.00</td>
<td>96.0</td>
<td>91.0</td>
<td>6.00</td>
<td>80.0</td>
<td>74.0</td>
</tr>
<tr>
<td>Trojan</td>
<td>7.00</td>
<td>93.0</td>
<td>86.0</td>
<td>9.0</td>
<td>84.0</td>
<td>75.0</td>
</tr>
<tr>
<td>Backdoor</td>
<td>8.00</td>
<td>95.0</td>
<td>87.0</td>
<td>8.00</td>
<td>85.0</td>
<td>77.0</td>
</tr>
<tr>
<td>Ave.</td>
<td>7.00</td>
<td>94.0</td>
<td>87.0</td>
<td>7.00</td>
<td>83.0</td>
<td>76.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Web</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SBS-AMF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Downloader</td>
<td>6.0</td>
<td>96.0</td>
<td>90.0</td>
<td>3.0</td>
<td>81.0</td>
<td>78.0</td>
</tr>
<tr>
<td>Infostealer</td>
<td>5.0</td>
<td>100</td>
<td>95.0</td>
<td>3.0</td>
<td>80.0</td>
<td>77.0</td>
</tr>
<tr>
<td>Trojan</td>
<td>3.00</td>
<td>98.0</td>
<td>95.0</td>
<td>4.00</td>
<td>83.0</td>
<td>79.0</td>
</tr>
<tr>
<td>Backdoor</td>
<td>3.70</td>
<td>94.0</td>
<td>91.0</td>
<td>2.00</td>
<td>80.0</td>
<td>78.0</td>
</tr>
<tr>
<td>Ave.</td>
<td>4.30</td>
<td>97.0</td>
<td>93.0</td>
<td>3.00</td>
<td>81.0</td>
<td>78.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMail</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SBS-AMF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Downloader</td>
<td>6.00</td>
<td>90.0</td>
<td>84.0</td>
<td>4.00</td>
<td>77.0</td>
<td>73.0</td>
</tr>
<tr>
<td>Infostealer</td>
<td>4.00</td>
<td>91.0</td>
<td>87.0</td>
<td>5.00</td>
<td>75.0</td>
<td>70.0</td>
</tr>
<tr>
<td>Trojan</td>
<td>8.00</td>
<td>89.0</td>
<td>81.0</td>
<td>3.00</td>
<td>71.0</td>
<td>68.0</td>
</tr>
<tr>
<td>Backdoor</td>
<td>8.00</td>
<td>92.0</td>
<td>84.0</td>
<td>5.00</td>
<td>72.0</td>
<td>67.0</td>
</tr>
<tr>
<td>Ave.</td>
<td>7.00</td>
<td>91.0</td>
<td>84.0</td>
<td>4.00</td>
<td>74.0</td>
<td>70.0</td>
</tr>
</tbody>
</table>
Figure 6.4: Example of detecting a Downloader Malware with LOF using the selected features of SBS-AMF (Top), and AdaBoost-CR (Bottom).
6.5 VMM IDS Performance Detection Speed Results

We report our experimental results focusing on the VMM IDS detection speed using the LOF GPU implementation (LOF-GPU) as compared to a multithreaded CPU implementation (LOF-C) developed in C (compiled with gcc 4.2), in terms of three factors: (1) input data size, (2) neighborhood size \((k)\), and (3) feature set size.

Results in Figure 6.5 show the sensitivity of the two different implementations to these factors. Sub-figure(a) and (b) in Figure 6.5 show the performance of the two implementations in execution time with respect to input data size. As the slopes in graph indicate, our GPU implementation scales very favorably as input data size increases, making it very suitable for large-scale data processing.

In Sub-figures (c) and (d) in Figure 6.5 we plot the execution time vs. \(k\) (i.e., the size of neighborhood). The computation time seems to increase linearly with the size of \(k\). The major difference between these implementations is the slope of the increase. Let us consider the case where \(k = 20\), the slope of LOF-C is 0.52, and almost 0.001 for the LOF-CUDA implementation. The same trend can be seen in Sub-figures (e) and (f) in Figure 6.5, which shows the impact of changing the feature set size on execution time.

In other words, our VMM IDS with CPU implementation (LOF-C) is sensitive to data size, size of the neighborhood, and the feature set size in terms of computation time. On the other hand, the impact of these factors on the performance of our VMM IDS with GPU implementation (LOF-CUDA) is very small.

Tables 6.9 and 6.10, show the VMM IDS computation time in seconds for our two implementations, LOF-CUDA and LOF-C. Here, \(n\) corresponds to the input data size and \(No.\text{feat}\) corresponds to the number of features. Note that results in Table 6.10 are generated using the top 5 features ranked by SBS-AMF, and for Table 6.10 we set \(k = 20\).

The speed-up results obtained in Figure 6.6 shows that by mapping the LOF algorithm to a GPU, we can greatly reduce the execution time needed for the VMM IDS to detect intrusions. According to plots in Figure 6.6 the execution time of the VMM IDS on GPU is up to 100 times faster than CPU. For instance, with 3620 data points and with \(k = 40\), the computation time is 24 seconds on CPU, whereas the execution is less than half a second on GPU.

Exploiting the fact that our GPU implementation is fairly insensitive to changes in the input parameters chosen is very attractive to our intrusion detection application as near real-time performance is needed in order to detect any malicious activity before the system becomes fully compromised.

The desirable insensitivity observed from the experimental result benefits from the fact that GPU becomes more efficient as data size increases thanks to better memory latency hiding and hardware utilization. Note that the execution time of CPU implementation follows algorithm complexity as input size increase. These two benefits make GPU a very suitable platform for our intrusion detection application.
Figure 6.5: Impact of each factor on execution time of the VMM-IDS using LOF algorithm (right plots in log-scale).
Table 6.9: Comparison of the VMM IDS computation time, with respect to the size of data set (n) and the number of neighbors (k). All numbers in seconds.

<table>
<thead>
<tr>
<th>k</th>
<th>LOF implementation</th>
<th>n</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1810</td>
<td>3620</td>
<td>5430</td>
<td>10860</td>
</tr>
<tr>
<td>1</td>
<td>LOF-C</td>
<td>1.0</td>
<td>1.0</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.08</td>
<td>0.10</td>
<td>0.08</td>
<td>0.09</td>
</tr>
<tr>
<td>20</td>
<td>LOF-C</td>
<td>7.0</td>
<td>12.0</td>
<td>19.0</td>
<td>36.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.12</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
</tr>
<tr>
<td>40</td>
<td>LOF-C</td>
<td>12.0</td>
<td>24.0</td>
<td>36.0</td>
<td>72.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.18</td>
<td>0.21</td>
<td>0.22</td>
<td>0.26</td>
</tr>
<tr>
<td>80</td>
<td>LOF-C</td>
<td>24.0</td>
<td>48.0</td>
<td>72.0</td>
<td>94.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.41</td>
<td>0.50</td>
<td>0.55</td>
<td>0.63</td>
</tr>
<tr>
<td>100</td>
<td>LOF-C</td>
<td>31.0</td>
<td>60.0</td>
<td>90.0</td>
<td>123.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.58</td>
<td>0.68</td>
<td>0.76</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Table 6.10: Comparison of the VMM IDS computation time, with respect to the size of data set (n) and the feature set size (No.features). All numbers in seconds.

<table>
<thead>
<tr>
<th>No.features</th>
<th>LOF implementation</th>
<th>n</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1810</td>
<td>3620</td>
<td>5430</td>
<td>10860</td>
</tr>
<tr>
<td>5</td>
<td>LOF-C</td>
<td>7.0</td>
<td>12.0</td>
<td>19.0</td>
<td>36.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.12</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
</tr>
<tr>
<td>25</td>
<td>LOF-C</td>
<td>16.0</td>
<td>33.0</td>
<td>49.0</td>
<td>99.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.13</td>
<td>0.14</td>
<td>0.14</td>
<td>0.16</td>
</tr>
<tr>
<td>50</td>
<td>LOF-C</td>
<td>29.0</td>
<td>59.0</td>
<td>88.0</td>
<td>177.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.13</td>
<td>0.14</td>
<td>0.15</td>
<td>0.17</td>
</tr>
<tr>
<td>75</td>
<td>LOF-C</td>
<td>55.0</td>
<td>112.0</td>
<td>168.0</td>
<td>342.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.14</td>
<td>0.15</td>
<td>0.16</td>
<td>0.20</td>
</tr>
<tr>
<td>100</td>
<td>LOF-C</td>
<td>107.0</td>
<td>215.0</td>
<td>323.0</td>
<td>726.0</td>
</tr>
<tr>
<td></td>
<td>LOF-CUDA</td>
<td>0.18</td>
<td>0.17</td>
<td>0.19</td>
<td>0.24</td>
</tr>
</tbody>
</table>
Figure 6.6: VMM IDS speed-up. (Top) impact of feature set size. (Bottom) impact of neighborhood size.
6.6 Discussion

In this section we address the following challenges: 1) impact of window size on classification results, 2) threshold selection, and 3) the alarm mechanism.

6.6.1 Impact of sliding window size

Our study shows that window size parameter has a crucial impact on the classification results. As described in Chapter 5, we use the extracted events as guidelines to construct temporal statistical features for building classification models, by summarizing the raw events output into records using pre-processing programs, where each record has a set of features (see 6.7 as an example).

This is done by dividing the stream of collected events into consecutive windows of equal (virtual) time, i.e., by using the event \( \text{Timer} \). The length of the window introduces a trade-off: longer windows capture more behavior while shorter ones reduce the time-to-detection. i.e., when the window size is very small everything will be classified abnormal (noise), and as this parameter gets very large everything seems to look normal. Figure 6.8 shows the time series plots results of classifying abnormal behavior, with different window sizes (10, 100, 1000).

We have found that windows of approximately 100 timers (2 seconds) long provide a good balance between these trade-offs. This time quantum provides the back-end with sufficient information to make accurate classifications on a per-window basis, while also allowing it to identify any malicious activity within seconds of its execution.

6.6.2 Threshold selection

Defining a representative normal region is challenging, and the boundary between normal and outlying behavior is often not precise. There several graphical-based techniques used for threshold selection and based on the distribution of normal data, such as:

**Histogram** — the purpose of a histogram is to graphically summarize the distribution of an univariate data set. The histogram graphically shows the following: center (i.e., the location) of the data; spread (i.e., the scale) of the data; skewness of the data; and presence of outliers. Recall that a histogram is a probability distribution:
CHAPTER 6. VMM IDS EVALUATION

Figure 6.8: Impact of window size.
\[ p(u_i) = \frac{D_{u_i}}{D} \]  
(6.1)

If the histogram shows the presence of outliers, the recommended next step is to graphically check for outliers (in the commonly encountered normal case) by generating a box plot. In general, box plots are a much better graphical tool for detecting outliers than are histograms.

**Box-plot** — The box plot is a useful graphical display for describing the behavior of the data in the middle as well as at the ends of the distributions. The box plot uses the median and the lower and upper quartiles (defined as the 25\(^{th}\) and 75\(^{th}\) percentiles). If the lower quartile is \(Q_1\) and the upper quartile is \(Q_3\), then the difference \((Q_3 - Q_1)\) is called the *interquartile range* or IQR. A box plot is constructed by drawing a box between the upper and lower quartiles with a solid line drawn across the box to locate the median. The following quantities (called fences) are needed for identifying extreme values in the tails of the distribution:

- lower inner fence: \(Q_1 - 1.5 \times \text{IQR}\)
- upper inner fence: \(Q_3 + 1.5 \times \text{IQR}\)
- lower outer fence: \(Q_1 - 3 \times \text{IQR}\)
- upper outer fence: \(Q_3 + 3 \times \text{IQR}\)

A point beyond an inner fence on either side is considered a *mild outlier*. A point beyond an outer fence is considered as *extreme outlier*. In our design we set initial value of the threshold \(\theta\) based on the *mild outlier*.

Therefore, we set:

\[ \theta_0 = Q_3 + 1.5 \times \text{IQR} \]  
(6.2)

### 6.6.3 Alarming mechanism

Once initial thresholds are set up from training we can do detection. The alarm is raised when any violation for the threshold occur. However, to reduce number of false alarms we modify the alarming condition to be raise alarm only if \(k\) consecutive violations occur. The alarming condition is as follows:

\[
\sum_{i=n-k+1}^{n} x_n \geq \theta \geq k
\]  
(6.3)
Figure 6.10: The boxplot for each workload based on normal data.

where $\theta$ is the threshold

The alarming mechanism is based on a sequential detection procedure. At time $t$, we are in one of these regions:

- Acceptance state (A): The system is behaving as expected, no anomalies, start a new run
- Rejection state (R): An anomaly is generated which is reported to the user and/or the forecasting model is reset. Start a new run.
- Continue (C): Don’t have enough data to reach a decision, keep accumulating evidence by taking another sample.

6.7 Chapter Conclusions

Intrusion detection system is one of the high priority and challenging task, particularly, in virtualization technology. Such IDS prime purpose is to detect most of the attacks, give very few false alarms, copes with large amount of data, and is fast enough to make real-time decisions. However, there still exist several open issues regarding these systems. Some of the most significant challenges are: 1) the poor detection rate, due to the presence of excessive features in a data set whose class distributions
are imbalanced, and 2) the slow detection process, mainly due to the expensive computation time of the underlying detection algorithms.

In this chapter we validated that sufficient information exists within the features constructed from events captured at the VMM layer in order to distinguish normal from abnormal behavior, especially, when the class imbalance ratio is high using our proposed method, the SBS-AMF. We showed that not all extracted raw events are important, and therefore constructing features only on frequently-used events can enhance the detection accuracy and speed of the VMM IDS.

We evaluated the performance of the VMM IDS on different commercial virtual appliances and different types of malwares. Our supervised learning study shows that our EnhancedBoost algorithm has low computational complexity and high detection rates. We obtained 97.4% true detection and 1.23% false alarm rate when testing real malwares, as well as the advantages of boosting the AUC for the detection process for intrusion data.

We then, employed a more realistic scenario where we only have normal training samples using KNN and LOF algorithms. We compared our SBS-AMF features against AdaBoost-CR features, and our experimental results showed that SBS-AMF achieves significantly better detection performance on the data sets tested. We obtained 96% detection rate and 5% false alarm rate. These results indicate that sufficient information exists in features selected by SBS-AMF to build a real IDS that is not susceptible to the characteristics of the attack behavior, or to specific workload. We also presented a study comparing the execution time of the VMM IDS using LOF implementation on both a CPU platform and a GPU platform. Besides gaining of over 100X speed-up, we observed that the size of the data set, the number of neighbors and the feature space have only a small impact on the computation time when run on a GPU, versus the linear scaling experienced on the CPU.

Furthermore, we addressed several issues in designing the VMM IDS, such as: 1) Impact of window size on classification results, 2) threshold selection, and 3) the alarm mechanism.
Chapter 7

Summary and Conclusion

This thesis consists of two main parts: 1) a theory and algorithm part, and 2) an application part. Part one focuses on the basic research issues arising in feature selection, and claims that feature selection for supervised machine learning tasks can be accomplished on the basis of the margin that boosting algorithm produces. In particular, this part of the thesis investigates the hypothesis that a good feature subset is one that contains features that contribute more to the margin distribution associated with the weighted linear combination that boosting produces. We argue that this concept is more favorable for several reasons. First, boosting hypothesis margins have been used both for theoretical generalization bounds and as guidelines for algorithm design, and thus, a natural goal is to find learners (features) that achieve a maximum margin. Second, current boosting-based feature selection methods measure the relative importance of features based on the Confidence Ratio (CR) of the learned base hypothesis. However, while a feature may have a large CR, it will not contribute to a good overall margin unless its “conditional” margin is also large. Based on this theory, we designed two different embedded-based feature selection algorithms, the SBS-MF and the SBS-AMF.

In part two, we applied the SBS-AMF proposed method to build a lightweight anomaly intrusion detection system in virtualization, VMM IDS. VMM-based IDSs break the boundaries of current state-of-the-art IDSs. They represent a new point in the IDS design space that trades a lack of program semantics for greater malware resistance and ease of deployment. To test the effectiveness and robustness of our proposed VMM IDS, we used different classes of servers, virtual appliances, and workloads, as well as different classes of malwares. Our experimental results showed that SBS-AMF achieves significantly better detection performance on the data sets tested. We obtained on average 96% detection rate and 5% false alarm rate. These results indicate that sufficient information exists in features selected by SBS-AMF to build a real IDS that is not susceptible to the characteristics of the attack behavior, or to specific workload.

We then, described how to develop a GPU-enabled CUDA implementation for the LOF method to accelerate the IDS framework. Our results show that detecting a malware on CPU takes 24
Next, we present the contributions of this thesis and future research directions that this work can take.

### 7.1 Contributions

The contributions of this thesis are:

i) Proposed feature selection algorithms:

   (a) Devised a classifier independent (embedded) feature selection technique – which should be able to reduce the dimensionality of the given problem by eliminating redundant and irrelevant features. In particular, we presented a novel framework of margin based weighting for feature selection which extensively explores the hypothesis margins of boosting. We introduced the idea of utilizing the training examples’ average margin to measure the quality and the relative importance of features, using a proposed weight criterion, termed Margin Fraction (MF). The problem of searching the “best” subset of features is solved by means of a greedy algorithm based on backward selection.

   (b) Extending the weighting and selection methodology to handle the presence of imbalanced data – by defining a new weight metric that characterize the quality of a set of features based on the maximized the Area Under ROC curve (AUC) margin it induces during the process of learning with boosting. We refer to it as AUC Margin Fraction (AMF).

ii) Evaluation of the proposed algorithms – We created grounds for a fair comparison of the proposed feature selection algorithms through extensive comparisons with other state-of-the-art methods, including boosting-based and margin-based feature selection algorithms using real-world high-dimensional data from UCI repository [77] and datasets that contain a large number of features with a small number of samples and significant imbalance between the two classes. Our experimental results showed that our methods, SBS-MF and SBS-AMF achieve equal or significantly better performance to other approaches on the data sets tested.

We also try to address potential issues in feature selection, such as the impact of the existence of correlated features, and exploring differences in performance on highly imbalanced classes versus balanced classes, and we lay out the limitations of our proposed methods and how these potential weaknesses can be addressed.
iii) Developing lightweight anomaly intrusion detection system in virtualization – We applied the proposed feature selection algorithm to design effective IDS for virtual server environments, by addressing the following issues:

(a) Constructing different types of features from raw data (events) utilizing information embedded within the virtual machine monitor (VMM) level.

(b) Handling the presence of excessive features in an intrusion data whose class distributions are imbalanced.

(c) Designing a novel supervised detection algorithm based on Enhanced boosting algorithm with Decision Stumps to detect various categories of attacks.

(d) Applying two unsupervised anomaly detection algorithms by training only with normal data with four different commercial virtual appliances and on a set of approximately 300 real-world malwares.

iv) Accelerating LOF algorithm with GPUs for IDS – We developed a GPU-enabled CUDA implementation for LOF method to accelerate the IDS framework. We achieved more than a 100X.

In summary, this thesis enhances the current state-of-the-art and makes key contributions to the following areas:

- **Machine Learning/Datamining** – Devising classifier independent (embedded) feature selection techniques which should be able to reduce the dimensionality of the given problem by eliminating redundant and irrelevant features and handle the presence of imbalanced data using the concept of the hypothesis margin of boosting.

- **Cyber Security for Virtualization** – Applying the proposed feature selection to design a novel lightweight anomaly intrusion detection in virtualization environments.

- **Graphics Processing Unit (GPU) for Machine Learning** – Developing a GPU-enabled CUDA implementation to accelerate anomaly detection algorithm.

### 7.2 Future Work

Our SBS-MF is an effective method, however it introduces a few limitations which can be explored in the future work. In particular the following issues: (1) our weighting method MF might not be robust to noise since it is based on the hard margins of boosting. Part of the future work is
to investigate how to overcome this issue by utilizing the soft margins of boosting instead, (2) the sequential greedy search approach is used, and this means adding or removing features sequentially, but have a tendency to become trapped in local minima, therefore, we can incorporate other search techniques, and 3) our method is not generally recognized that functional values of the criterion guiding the search for the best feature set are random variables. The results of our SBS-MF may therefore be heavily dependent on the sampled training data. The inherent instability often leads to incorrect conclusions regarding the features effectiveness. Hence, it is important to be able to measure and optimize the stability of resulting feature sets.

For our VMM IDS, we currently construct features based on the events that occur during the execution of many processes within the system. In future work, we can incorporate features constructed on a per-process basis. As a first processing pass over the raw data, one can use the information contained in CR3 (which allows one to distinguish processes) and various timers to create a ordered sequence of (time, process, event) data triples, where each triplet may be associated with some auxiliary information. For example, a given data triple may encode that at a particular time a particular process performed an IO read, and the auxiliary information may be an associated sector number. Note that CR3 alone does not provide enough information to identify a process in the usual sense (e.g., process name, process ID, parent-child relationships, etc.); however, it does allow one to distinguish distinct processes. These data triples can then be mined to extract any number of features, for example-process basis, one can extract the sequence of events over time, a kind of process signature. (2) For any given interval of time, one can extract the set of occurring events. (3) For any given process and interval of time, one can extract the set of associated events. Mathematically, one can take a sequence of data triples and condition on time, process, event, or any combination thereof to extract any number of potentially useful features. Consequently, the detection process of the VMM IDS will be a combination of these modules, per-system and per-process, to produce a stronger and more accurate detector/classifier.
Bibliography


