EFFECTIVE DETECTION OF SECURITY COMPROMISES IN ENTERPRISES USING FEATURE ENGINEERING

BY

JIAYI DUAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Associate Professor Shobha Vasudevan

ABSTRACT

We present a method to effectively detect malicious activity in the data of enterprise logs. Our method involves feature engineering, or generating new features by applying operators on the features of the raw data. We apply the Fourier expansion of Boolean functions to generate parity functions on feature subsets, or *parity features*. We also investigate a heuristic method of applying Boolean operators to raw data features, generating *propositional features*. We demonstrate with real data sets that the engineered features enhance the performance of classifiers and clustering algorithms. As compared to classification of raw data features, the engineered features achieve up to 50.6% improvement in malicious recall while sacrificing no more than 0.47% in accuracy. Clustering with respect to the engineered features finds up to 6 "pure" malicious clusters, as compared to 0 "pure" clusters with raw data features. In one case, exactly one (1) engineered feature could achieve higher performance than 91 raw data features. In general, a small number (<10) of engineered features achieve higher performance than raw data features.

Thanks to my parents for all the support during my school and university years.

ACKNOWLEDGMENTS

Thanks to Prof. Shobha Vasudevan, my academic advisor, for all the support and advising during my master's program.

Thanks to Alina Oprea for providing all the enterprise log data sets for us to do the experiments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DNF	Disjunction normal form
DT	Decision tree
FN	False negative
FP	False positive
PEBL	Positive Example Based Learning
RF	Random forest
RFI	Random forest importance
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
TN	True negative
TP	True positive

a_S	The Fourier coefficient of subset $S \in 1, ..., n$ for a given χ_S.
$\hat{a}_S$	The estimated (from the training data set) Fourier coefficient of subset $S \in 1, ..., n$ for a given χ_S.
f	The ideal target function which satisfies $f(x) = y$ for all examples x in the given data set.
m	Number of examples in the training data set.
$x = (x_1, ..., x_n)$	An example of the n-dimensional data set.
$x^{(i)}$	The i-th example in the training data set.
$x_j^{(i)}$	The j-th element of the i-th example in the training data set.
y	The label of an example in the data set.
$\chi_S(x)$	The parity value of example x on subset $S \in 1, ..., n$.

CHAPTER 1

INTRODUCTION

1.1 Motivation

The security landscape is continuously evolving with a variety of new attacks. Common threats today range from criminal activities which are opportunistic and financially motivated (e.g., [1]) to cyber-espionage and state-sponsored campaigns targeting particular organizations with specific motivation and objectives (e.g., [2]).

To counteract increasingly sophisticated attacks, most organizations deploy a variety of security controls within their perimeter, including firewalls, anti-virus (AV) software, intrusion detection systems, web proxies and network appliances. The security logs generated by these controls are collected and stored by many enterprises, resulting in terabytes of log data generated daily in large organizations.

While these controls help defend against large classes of attacks, they are insufficient for protecting enterprise perimeters in today's continuously evolving threat landscape. It is relatively easy for attackers to evade these controls, as evidenced by the number of successful breaches happening today. As an additional challenge, employees can bring their own devices inside corporate networks, and expose corporate machines to various threats when traveling or working remotely. According to a report published by Verizon [3], in 2014 alone, there were 2000 security breaches among 10,000 surveyed organizations worldwide. The effects of security breaches (e.g., the Sony breach [4], or the breach against the Anthem insurance company [5]) are catastrophic, with estimated losses in 2014 being around 400 million dollars.

To complement the protections offered by security products, large enterprises leverage teams of security analysts. These are experts tasked with identifying suspicious activities in the network. Security analysts spend much time manually investigating alerts, detecting the root cause of the attack, and remedying their effects. It is highly desirable to automate this manual investigation process of detecting malicious communication within an enterprise.

The enterprise logs, as most other data, are created with respect to manually specified features. These features represent aspects of the data that human beings consider relevant. However, it is not clear that manually generated features would be the best parameters to classify, cluster, or make predictions with the data. Given that features are critically important in the efficacy of any learning and/or data analysis task, they merit careful analysis. Manual analysis of enterprise log data has three issues. One is the scale of enterprise log data. For big enterprises, the log data can be terabytes per day, making it extremely hard for humans to manually analyze. Even if the data has been pre-processed to summarize the behavior of each domain, there will still be millions of domains under analysis. Manual analysis will take too much time in such cases. The second issue is the importance of features. Even for domain experts, different people may have different understandings of which feature could be important and which is not. In addition, their opinions may not be suitable for the classifiers used to do the identification. The third issue is that manual analysis is vulnerable to attacks. If domain experts can identify important features and use them for classification, attackers can also identify important features and disguise malicious examples to align them with benign ones (called "mimicry attack"). Attackers just need to know a few benign examples to do so. Now given that manual analysis is not a proper way to process the data and features, we propose an idea of generating all relevant features and using an algorithm to identify or even generate important features. Since all the work is assigned to machine learning, the processing time could be shortened a lot compared to manual analysis, and the features may be more friendly for the machine. Besides, attackers may have difficulty getting to know the features used in the classifier because the analysis is based on the whole training set, and some new features are generated that may be hard for attackers to guess.

Generating new features is necessary in some applications. One of the reasons to do so, as stated above, is to enhance robustness against mimicry attack. Another important reason is that new features can potentially achieve higher accuracy and higher recall of malicious examples, while making the training and testing procedure faster. A disadvantage of raw features is that they are in different format: some features are numeric (real values, such as 35.02, 0.425), others are nominal (strings, such as “True,” “small”). Furthermore, the value scales of numeric features may significantly differ from each other, and numbers of possible values of nominal features may also significantly differ from each other. This makes it difficult to process the data. For example, when calculating the distance between two examples, features with large values could contribute much more than features with small values, and we might even need to use some new metrics to deal with both numeric and nominal features. Some classifiers that use distances may suffer from this. Even for classifiers that do not use distances, there may still be many redundant and unrelated features that misguide the training and may cause over-fitting. Moreover, when we apply clustering algorithms, there are two disadvantages of using raw features: one is that the output heavily depends on the distance measurement, which could be disturbed by different scales and different types of features; the other is that the large number of features makes the data space sparse, and distance measurement may be even less meaningful in such situations. Therefore, transforming the features into a unified format is necessary for both classification and clustering. The transformation is the key to generating new features. However, we want to keep the physical meaning when performing the transformation, and this is a big difference from other feature selection and feature transformation methods.

In this work, we investigate methods for *feature engineering*, or generating new features from operations on existing features of the raw data. We create a new feature space which can then be used for analyzing the data. Adding our engineered features to the raw data features will increase the dimensionality of the problem. Hence, we go through a selection process to identify the most important features among the total set of features and select the smallest subset that gives us an empirically acceptable metric (accuracy, precision, recall, etc).

1.2 Introduction to our approach

We investigate two methods for feature engineering. For both our methods, we generate disjunctive normal form (DNF) formulas with respect to the raw features and extract Boolean features from the formulas.

The first method is based on Fourier analysis of Boolean functions, $f : \{0,1\}^n \longrightarrow \{-1,+1\}$ that has widespread applications in various fields [6]. If the domain of the Boolean function $\{0,1\}^n$ can be thought of as 2^n points lying in R^n, and f as assigning labels $\{-1,+1\}$ to these data points, these data points can be interpolated with a polynomial. The Fourier expansion multiplies each x-interpolant by the desired $f(x)$, and then expands it to a unique multi-linear polynomial that can then be simplified, and the original function can be reconstructed. An unknown function can be learned by approximating its Fourier coefficients, where the n variables of the function correspond to the features of the data. Since the Fourier coefficient is an expectation under the uniform distribution on $\{0,1\}^n$, it can be approximated from (training) examples in the data set. An orthonormal basis is computed for every subset of features. In this case, we use the parity function between feature subsets as the basis. If f is dominated by a few large Fourier coefficients and we know a small subset of features corresponding to those coefficients, f can be approximated by a real polynomial of low degree. In these cases, the coefficients can be approximated quickly and faithfully with a reasonable training set size. We do not use the Fourier expansion to learn the function in this thesis. We use the Fourier expansion to (1) generate new features using the parity function on feature subsets or *parity features*, and (2) compute the Fourier coefficients for estimating the importance of each such parity feature with respect to f. Since Fourier expansion does generate new features as well as rank features and perform feature selection, it can dig into the correlations between features and thus generates more information that may be useful; then the coefficients do the work of feature selection which is more suitable for parity functions than other feature selection methods. We also use lasso and random forest as two ways of feature selection in the comparison between Fourier expansion and other feature selection methods. We can see that the Fourier method performs the best in the experiments.

The Fourier expansion provides a formal method using principles of harmonic analysis to generate new features. The second method we investigate in this thesis is a more heuristic method to achieve the same objective. We apply common Boolean operators like AND, OR, NOT and combinations of these between the Boolean features and generate *propositional features*. We then analyze this list of propositional features with the Fourier coefficients to rank the most important ones. We select the top-ranked propositional features according to their Fourier coefficients. A reason to investigate these type of operations is that a human expert can easily understand and interpret the meaning of these operations between features. This is especially important in the security application, since manual investigation is a part of the process. For the same reason, we limit the number of features per operation to 2 in the propositional features. It is possible to perform more complex integer or real operations for engineering new features. The current work is a first step in this direction.

For evaluating the effectiveness of the feature engineering, we provide a few top-ranked parity features and propositional features to different classifiers and clustering algorithms. We compare the performance (accuracy, precision, and recall) with the performance when using all the raw data features. In the case of our application, recall of malicious domains in the logs is a very important parameter, since a high recall ensures that all the malicious domains are classified correctly, while a low recall means that many malicious domains escape detection. Due to the imbalanced nature of this data set with very few malicious (4.74% and 2.63% in the two data sets) and a large number of benign domains, all the classifiers show a high prediction accuracy, but have a low recall (as low as 20% and no greater than 70%) with raw data features. With feature engineering methods, the recall shoots up to as high as 92.85% while maintaining the accuracy at an acceptable level. We are able to show such high performance with much fewer features than raw data features (<15 in general). We show that even with just one (1) engineered feature, we can get up to 98.02% accuracy and 78.61% recall, as compared to 98.61% accuracy and 70.9% recall with the entire set of 91 features. With clustering algorithms, we show that the clusters with raw data features are not "pure"; *i.e.*, not even one of the clusters can be identified as malicious or benign. In contrast, the engineered features can generate up to 6 "pure" clusters. We demonstrate that the engineered features have high explainability and interpretability for use by a human expert.

In general, parity features outperform propositional features in our experiments with respect to recall achieved, number of features, and quality of clusters. This argues for the more formal harmonic analysis approach to feature engineering over the heuristic approach.

We also compared our feature engineering method with other feature selection methods. The results show that our engineering method beats both lasso and random forest importance (RFI) when using parity features by averagely around 10% higher recall. Meanwhile, our method converts the messy raw features into uniform Boolean features while retaining physical meanings, which is hard for other feature selection and feature transformation methods.

1.3 Adversarial learning

Recently, methods for evading malware detection have been studied. One of the common methods, called mimicry attack [7], focuses on how to let the malicious examples mimic benign examples in some or all features, thus aligning such malicious examples with known benign examples. In this way, a classifier will not detect such malicious examples if it is not robust enough. A mimicry attack can perform differently when the attacker has different knowledge of the procedure of malicious example detection. "F", "T", and "C" denote that the attacker has knowledge of the features, training data set, and the classifier model, respectively. There is also a tool introduced in [7] called "Mimicus" which performs mimicry attack on malicious PDF files.

We used FC and FTC Mimicus [7] on a public PDF data set "Contagio" [8] and extracted features from benign files, malicious files, and disguised (after being modified by Mimicus) malicious files. After training an SVM on 135 raw features with unmodified examples, we tested them on disguised malicious examples. The result shows that the classifier can only identify 3.4% of the disguised malicious examples, while this number is over 99% on unmodified malicious examples. Fortunately, our method of feature engineering makes the features more robust to such kinds of attack. Our modified features retain over 96% accuracy under mimicry attack, compared to 3.4% when using raw features. Our features are much better in such tasks.

1.4 Contribution of our method

In the experiment chapter, we discuss the improvements in detail. In summary, our method makes better features that result in much higher recall of malicious domains, with very little or even no sacrifice of overall accuracy. This means the new features we made can identify more malicious examples than using raw features, without sacrificing overall accuracy. In clustering, there will be more clusters in which a majority of the elements are malicious when using new features. These clusters indicate some common properties of malicious examples and give us information to deal with them. Meanwhile, the new features are much more robust to mimicry attacks than raw features.

Our method also has some properties that may benefit the application. First, the features are Boolean, which makes it much faster to train the classifier models. The raw features are a mixture of numeric and nominal features, and the values of numeric features differ a lot with each other on distributions. Thus, classifiers need to deal with the messy structure of raw features and get slowed down. Besides, calculation of distances is much faster for Boolean features (Manhattan distance) than numeric features (Euclidean distance). Second, our new features are propositions and combinations of propositions with physical meanings. People can easily understand the meaning of each proposition; for example, $Num_connection \geq 10$ means this domain has 10 or more connections with enterprise local machines. This helps when sometimes the classifier or features need to be further analyzed by a human. People can even change some values (thresholds) in the propositions if necessary.

1.5 Structure of this thesis

Chapter 2 summarizes related research in feature selection, feature transformation, security and malware detection, and adversarial learning. Chapter 3 presents the details of our method, including every step we used to generate and rank features and to achieve our experimental results. Chapter 4 discusses what happens when the classifier undergoes a mimicry attack, and how we can use our method to deal with this problem. Chapter 5 introduces the data sets we use to get the experimental results, and chapter 6 presents all the experimental results using these data sets and provides some discussion. Chapter 7 concludes the thesis.

CHAPTER 2

RELATED WORK

2.1 Feature selection and feature transformation

Research in the feature space has focused on (1) feature selection and (2) feature transformation. Feature selection involves selecting the best set of features from the original features of the data. Feature transformation involves numerical transformation of the raw features that do not preserve the physical meaning of the original raw features (such as weighted sum of the features). In contrast, we present an approach to apply operations on and between raw features with an explicit intention to preserve their physical meaning.

Convex function optimization [9, 10, 11, 12, 13] is a very popular feature selection method, where the idea is to build a convex loss function and minimize it. The function has two parts: one is prediction error, the other is the norm (L1-norm, L2-norm, etc.) of weights on original features. Both parts are given a positive coefficient. When minimizing the loss function, the algorithm minimizes the prediction error, while giving proper weight on features. The existence of the second part of the loss function helps in giving high weight to important features and low weight to useless features. Lasso [14] is one such method using L1-norm in the loss function.

Recently, online feature selection [15, 16, 17], or scoring features by weights and training with the highest weights, was introduced. The idea is to use weights to score features, and only keep a limited number of highest weights during training. Online streaming feature selection methods [18, 17] add features one at a time, and evaluate the improvement due to this newly added feature, then keep some of the most important features and discard others.

There are also some other kinds of feature selection methods introduced in recent years:

Xu et al. [19] introduce Gradient Boosted Feature Selection (GBFS), a feature selection method based on Gradient Boosted Trees. Xiang et al. [20] propose a sparse group hard thresholding algorithm for bi-level (features and feature groups) feature selection, and give an error bound for it. Woznica et al. [21] present a method of extracting patterns from feature models, and then use these patterns to further derive feature models for feature interactions. Cai et al. [22] introduce Multi-Cluster Feature Selection (MCFS), which selects features that can best present the clusters of the given unlabeled data set. Xu et al. [23] use spectral feature analysis to deal with both supervised and unsupervised feature selection. Cao et al. [24] present a dual method of tensor-based multi-view feature selection (DUAL-TMFS), which uses SVM for recursively eliminating features, and show its application in identifying brain diseases. Barkia et al. [25] use co-training and random forests, together with a permutation-based out-of-bag feature importance measure, to deal with a data set which has only a small set of labeled data. Farahat et al. [26] first define a reconstruction error of the data matrix based on subsets of features, then give a greedy approach to select features that minimize this reconstruction error. Le [27] uses a multi-layered locally connected sparse auto-encoder to learn high-level concepts completely from unlabeled data, and such concepts can be seen as transformed features. Zhai et al. [28] introduce a method that uses correlation measures as constraints and then use cutting plane strategy for selection. Jiang and Ren [29] propose that impacts of features on similarity matrix should be used as a feature importance measurement. Masaeli et al. [30] introduce the conversion of a transformation-based to a feature-selection method via $1/\infty$ regularization, in order to relax the problem from discrete optimization (selection) to continuous (transformation) optimization. Paul and Drineas [31] introduce a deterministic sampling based feature selection technique, and give worst-case guarantees of the generalization power for a function of selected features compared to all features. Paul et al. [32] perform feature sampling for SVM, and prove that the margin in the feature space is preserved to within ϵ-relative error of the original margin in the worst case.

2.2 Security and malware detection

There are many types of security and malware detection systems in enterprises. Beehive [33] is such a novel system that detects local network hosts that are infected, or host users that violate enterprise policies. The system uses 15 human-decided features of 4 types: destination-based, host-based, policy-based, and traffic-based. Then the system uses clustering on examples with the 15 features. Dahl et al. [34] use random projections and neural networks to classify malware. Their system generates 3 types of features from malware including null-terminated patterns, tri-grams of system API calls, and distinct combinations of a single API call and a parameter input. The number of original features is over 50 million, and they need to use feature selection first to reduce the dimensionality to 179 thousand. Pascanu et al. [35] describe a method of using time-series behavior of objects to classify malware. The classifier model is recurrent networks, an extension of neural networks which is good at dealing with time-series data. Their features are 114 distinct, high-level behavioral events generated by the anti-malware engine, and each time-series stream analysis considers the first 100 events.

As we can see, the above methods use human-defined features, which are perhaps the only option for people to understand what they mean. Some other methods, such as [36] and [37], simply use hexadecimal binary machine codes or n-gram of codes as features. This may yield more useful information than human-defined features, but the system may be hard for a human to understand and maintain since the codes are fixed and un-refinable for each example.

2.3 Adversarial learning

While classification methods are being developed, attackers are looking for ways to evade detection by classifiers. For example, a simple way of attacking is to let malicious objects mimic the behavior of benign objects, such as making spam emails look like normal emails and thus avoid detection. Therefore, it is important to learn classifier models in some ways that are robust against attacking. Brückner et al. [38], [39], and [40] describe the behavior of attackers and defenders as a static prediction game in which there exists a

Nash equilibrium. Biggio et al. [41], [42], and [43] propose a way to combine multiple classifiers and gain higher robustness against attacks. Biggio et al. [42] and Barth et al. [44] introduce a method that uses randomization or disinformation along with the classifiers so that the attackers will be misled when getting to know the system.

CHAPTER 3

OUR METHOD FOR FEATURE ENGINEERING

In this chapter, we introduce the whole procedure of our feature engineering method. Figure 3.1 shows the outline of our method. We first pass the raw data to a rule-based classifier and convert them into Boolean features, then apply Boolean operators to two Boolean features at a time, to get propositional features. We apply the Fourier transform on the propositional features and obtain an importance ranking using Fourier coefficients. In a parallel branch, we group subsets of features and calculate their parity according to the Fourier basis, to get parity features. We rank the parity features by importance using the Fourier coefficients as well. We select the top ranking features from both methods and provide them to classifiers and clustering algorithms. Every step will be discussed in detail in the next sections[1].

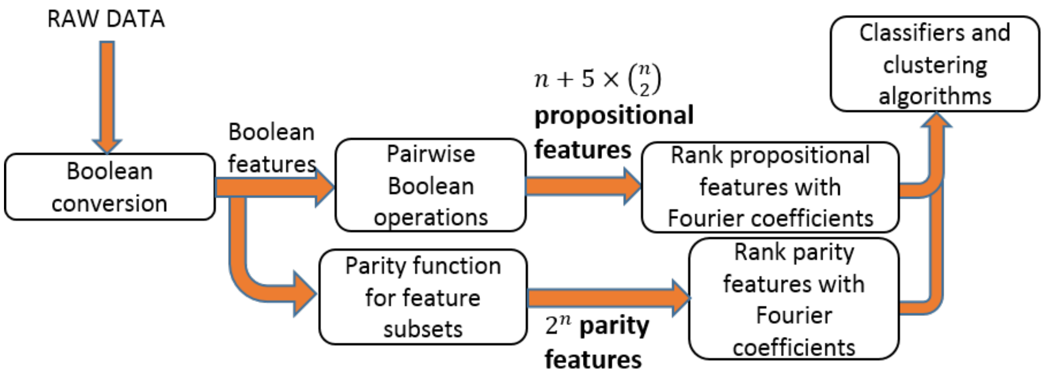

RAW DATA
Boolean conversion
Boolean features
Pairwise Boolean operations
$n + 5 \times \binom{n}{2}$ propositional features
Rank propositional features with Fourier coefficients
Classifiers and clustering algorithms
Parity function for feature subsets
2^n parity features
Rank parity features with Fourier coefficients

Figure 3.1: Flow graph of our method

[1]Some of the codes for implementation of our method can be found in the following Git repository with limited access: `https://duanjiayi1@bitbucket.org/jiayis_code/big_data_code.git`. Please contact me (duanjiayi1@gmail.com) for access to the codes.

3.1 Boolean conversion of raw features

The first step is to convert the numerical or nominal raw features to new Boolean features, in order to apply the Fourier transform. This is similar to the feature transformation that transfers raw features into a new Boolean space. The difference from other feature transformation methods is that it keeps some physical meanings of raw features, and it only transfers raw features that are useful for classification and discards others. We use RIPPER [45] on raw data to generate classification rules first. The classification model generated by RIPPER is a DNF. For example,

$$\begin{aligned}
&(A > 5) \wedge (B < 7) \wedge (D > 10) \rightarrow Class = TRUE\\
&(A > 8) \wedge (C < 15) \wedge (E = \text{``\textit{html}''}) \rightarrow Class = TRUE\\
&All\ others \rightarrow Class = FALSE
\end{aligned}$$

where A, B, C, D, E are original numerical or nominal features, and the values are thresholds generated by RIPPER algorithm.

Each conjunction in the DNF consists of several *Boolean features* like $(A > 5), (B < 7)$, and any example can get a TRUE or FALSE on such propositions.

To make sure the Boolean features are potentially useful, the accuracy of RIPPER's output should not be too low. Since RIPPER is a rule-based classifier, it can reach 100% accuracy if not pruned (which means severe over-fitting). Fortunately, we can measure the importance of Boolean features in the rules and pick out important ones, so we do not mind if we get more rules and more Boolean features in this step. The accuracy of RIPPER output is more important than the number of rules or over-fitting, so we restrict pruning to make the accuracy high enough (e.g. over 98%).

Actually, other rule-based classifiers, such as decision trees, work in this step. The propositions in each node of a decision tree can also be used as Boolean features. The reason of using RIPPER instead of decision trees is that RIPPER outputs DNF format rules but not tree structure rules. In tree structure rules, every proposition in a node is generated on the basis of its parent, its parent's parent, etc., except the root node. In this way, the feature and the corresponding threshold used in the node are influenced by all the parental nodes. What we need are individual propositions, so we want as little such influence as possible. DNF has less such influence because each rule is established on its own, without many constraints from its previous rules. Therefore, RIPPER is our choice.

3.2 Fourier coefficients

3.2.1 Fourier transform and coefficients

Fourier transform on Boolean variables [46] is a method to learn Boolean functions. Its target is to learn a Boolean function $f(x)$ with n Boolean variables $x = (x_1, ..., x_n) \in \{0,1\}^n$ as input and a label $y \in \{-1,1\}$ as output:

$$f : \{0,1\}^n \to \{-1,1\}$$

If f is learned to satisfy $f(x) = y \; \forall x$ in the training set, then $f(x)$ becomes a classifier. In our task, x is a training example, $x_1, ..., x_n \in \{0,1\}$ are n Boolean features of example x, and $y \in \{-1,1\}$ is the label of the example. We are not going to use the whole Fourier transform method to learn the target function, but we use some of its ideas (parity functions and Fourier coefficients) to generate new features and measure the importance of all features, and then select some of the important ones for classification. The generation of new features will blow up the Boolean feature space (generated from Boolean conversion in 3.1) into a much higher dimensional Boolean feature space, but the feature importance ranking and selection will reduce the dimension to no more than 30 for classification.

Just like Fourier expansion in signal processing that uses sine and cosine functions, the Fourier transform here uses parity function of subsets as the orthonormal basis. For any subset of Boolean variables $S \subseteq \{1, 2, ..., n\}$, the parity function $\chi_S(x)$ identifies odd or even parity of set S on example x. If there are odd 1's in subset S of example x, then $\chi_S(x) = -1$; otherwise, $\chi_S(x) = +1$, shown as follows:

$$\chi_S(x) = \prod_{i \in S} (-1)^{x_i} = \begin{cases} +1 \text{ if } \sum_{i \in S} x_i \mod 2 = 0 \\ -1 \text{ if } \sum_{i \in S} x_i \mod 2 = 1 \end{cases} \tag{3.1}$$

The output of parity function is also a Boolean value as $\chi_S(x) \in \{-1, 1\}$. Since there are 2^n subsets of $\{1, ..., n\}$, there will be 2^n selections of S and thus 2^n parity functions on each example x. Each such $\chi_S(x)$ represents a *parity feature* that we can generate from example x.

We now want to project $f(x)$ onto 2^n-dimensional feature space constructed by the parity features. It is desirable to ensure that each axis represented by χ_S is orthonormal to another. Orthonormalcy ensures that $f(x)$ can be expanded as a sum of projections on different χ_S axes. Orthonormalcy is defined as:

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} \chi_{S_1}(x)\chi_{S_2}(x) = \begin{cases} 0 \text{ if } S_1 \neq S_2 \\ 1 \text{ if } S_1 = S_2 \end{cases}$$

Since $\chi_{S_1}, \chi_{S_2}, ..., \chi_{S_{2^n}}$ satisfy such relationship, they can be used as 2^n orthonormal bases of axes in the parity function space.

The Boolean function $f(x)$ expands to:

$$f(x) = \sum_{S \subseteq \{1,...,n\}} a_S \chi_S(x) \tag{3.2}$$

The Fourier coefficient a_S for a given χ_S is computed across all training examples $x \in \{0, 1\}^n$, as equation (3.3).

$$a_S = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \chi_S(x) f(x) \tag{3.3}$$

As $\chi_S(x) \in \{-1, 1\}$, $f(x) \in \{-1, 1\}$, we can easily get $-1 \leq a_S \leq 1$.

This can also be used for reconstructing and thereby learning function $f(x)$ (as in (3.2)). In this thesis, we do not use a Fourier transform based classifier. We use this method only for computing coefficients as in (3.3).

In practice, computing 2^n parity functions of different feature subsets is computationally hard. Hence, an approximation can be used by computing fewer than 2^n subsets. The error due to this approximation can be bounded as follows.

For (3.2), Parseval's identity states

$$\sum_{S \subseteq \{1,...,n\}} a_S^2 = E[f^2] \tag{3.4}$$

where f is the function value (or label). Here we have $f \in \{-1, 1\}$, so $E[f^2] = 1$, and

$$\sum_{S \subseteq \{1,\ldots,n\}} a_S^2 = 1$$

Suppose we know only subset β with $a_\beta = 0.9$, and a hypothesis $h(x) = a_\beta \chi_\beta(x) = 0.9\chi_\beta(x)$; then the expected error is

$$E[(f-h)^2] = \sum_{S \subseteq \{1,\ldots,n\}, S \neq \beta} a_S^2 = 1 - a_\beta^2 = 0.19$$

Mansour [46] provides a bound on the error due to considering only a subset of 2^n possible parity features:

$$Pr[f(x) \neq Sign(h(x))] \leq E[(f-h)^2]$$

where

$$Sign(z) = \begin{cases} +1 \text{ if } z \geq 0 \\ -1 \text{ if } z < 0 \end{cases}$$

Equation (3.3) assumes that the training data set for computing coefficients is a complete set that contains all $x \in \{0,1\}^n$. However, this is seldom the case in practice. We need to approximate (3.3) in the absence of a complete training set. When there's only an incomplete training data set T including $m < 2^n$ examples, the approximation of Fourier coefficient of subset β is

$$\hat{a}_\beta = \frac{1}{m} \sum_{x \in T} \chi_\beta(x) f(x) \tag{3.5}$$

Using the Chernoff bound, there is

$$Pr[|\hat{a}_\beta - a_\beta| \geq \lambda] \leq 2e^{-\lambda^2 m/2}$$

Given that $|\hat{a}_\beta - a_\beta| \leq \lambda$, and

$$E[(f - \hat{a}_\beta \chi_\beta)^2] = 1 - a_\beta^2 + \lambda^2$$

The original error is $1 - a_\beta^2$, so the additional error is λ^2. The bigger m is, the more we can reduce the error.

3.2.2 Scaling Fourier coefficient computation

Algorithm 1 Calculating Fourier coefficients for small subsets

```
1: function FCoeff($x_1^{(1)}, ..., x_j^{(i)}, ..., x_n^{(m)}, y^{(1)}, ..., y^{(m)}, k$)
2:     for $S \subseteq \{1, ..., n\}, |S| \leq k$ do
3:         for $i = 1$ to $m$ do
4:             $\chi_S(x^{(i)}) = \prod_{j \in S}(-1)^{x_j^{(i)}}$
5:         end for
6:         $a_S = \frac{1}{m}\sum_{i=1}^{m} \chi_S(x^{(i)})y^{(i)}$
7:     end for
8:     Return $a_S \; \forall S \subseteq \{1, ..., n\}$
9: end function
```

There will be 2^n parity features in total since a set with n elements has 2^n subsets. 2^n is too large a number to finish the computation of coefficients. Therefore, it is important to identify and select the Boolean feature subsets from 2^n possible subsets. A claim in [46] shows that, if $f(x)$ can be expressed as a decision list [47], then it is sufficient to concentrate on a small set $k \ll n$ of the feature variables. For this subset of k feature variables, Fourier coefficients can be approximated for every subset of this set. In this case, the total number of coefficients we need to calculate is $\binom{n}{1} + \binom{n}{2} + ... + \binom{n}{k}$. In our experiments, for example, we have 98 Boolean features after Boolean conversion, so there will be totally $2^{98} = 3.17 \times 10^{29}$ subsets, too many for any computer. However, if we choose $k = 4$, the number of coefficients we need to calculate is $\binom{98}{1} + \binom{98}{2} + \binom{98}{3} + \binom{98}{4} = 3.77 \times 10^6$, far smaller than 2^{98}. This is a proper number for us to finish the computation while not losing too much information of parities of larger subsets. RIPPER gives us a disjunction normal form (DNF) formula on raw features with a high overall accuracy (over 98.5%) in Boolean conversion. Since DNF is a proper subset of decision lists [47], it is reasonable to consider only the small subsets in our task. We only need to focus on some small subsets with high $|a_S|$, because negating them will cause $2|a_S|$ difference on (3.2). In other words, these subsets are the most important features for classification because they are most likely to influence the function output (label). We will use these absolute values of coefficients as importance measurements of features. Algorithm 1 shows how to calculate Fourier coefficients for small subsets $|S| \leq k$ from the training data examples x and labels y.

Since we do not know which k of n Boolean features are needed to be in the small set, there are $\binom{n}{1} + \binom{n}{2} + ... + \binom{n}{k}$ ways (instead of 2^k ways) of selecting subsets of this small set. Hence the number of subsets can still be large. We use the following method to retain the high coefficient subsets introduced in [46]. We start from subsets that include only one feature, calculate their coefficients, then keep only those with $|a_S| \geq \theta$, where θ is a threshold, and discard others. Next, we add one more feature (any one that is not already in the current subset) to such subsets to get S', calculate their coefficients, then again keep only those with $|a_{S'}| \geq \theta$. We keep doing this until we reach a certain number of features in a subset. This method does not guarantee that we can get all small subsets that have high coefficient, and actually it is possible that a subset of 2 features can get high coefficient while the 2 Boolean features get low coefficients independently. However, this method can empirically be shown to preserve high-coefficient subsets and save a lot of time and memory.

3.3 Generating propositional features

This is a heuristic method for feature engineering. For any 2 features A and B, we apply 5 operators A AND B, A OR B, $\neg A$ AND B, A AND $\neg B$, and A XOR B. These operations do not have orthonormal relationship or properties displayed by parity functions. However, these operators have a concrete and interpretable physical meaning. For n Boolean features, now we have $5 \times \binom{n}{2}$ new features as the output of pairwise Boolean operators. We call the $5 \times \binom{n}{2}$ new features together with n Boolean features *propositional features*. We treat the propositional features as new added feature dimensions and calculate Fourier coefficients for them. Algorithm 2 shows this process.

We restrict Boolean operations to 2 Boolean features at a time to keep computation scalable. Consider applying operators on 3 features when we have 98 Boolean features at the beginning; the number of new features generated in this way will be around 20 million. This is even much more than the number of parity features at $k = 4$. If we consider operators on 4 or more features, the number of new features will be exploding, so we chose to stay at 2 features. We found that Boolean features with high coefficients are frequently repeated as one of the features in the top-ranked propositional

Algorithm 2 Fourier coefficients for propositional features

```
1: function PwCoeff($x_1^{(1)}, ..., x_j^{(k)}, ..., x_n^{(m)}, y^{(1)}, ..., y^{(m)}$)
2:     for $i = 1$ to $n - 1$ do
3:         for $j = i + 1$ to $n$ do
4:             for $k = 1$ to $m$ do
5:                 $F_{ij1}(x^{(k)}) = x_i^{(k)}\ AND\ x_j^{(k)}$
6:                 $F_{ij2}(x^{(k)}) = x_i^{(k)}\ OR\ x_j^{(k)}$
7:                 $F_{ij3}(x^{(k)}) = \neg x_i^{(k)}\ AND\ x_j^{(k)}$
8:                 $F_{ij4}(x^{(k)}) = x_i^{(k)}\ AND\ \neg x_j^{(k)}$
9:                 $F_{ij5}(x^{(k)}) = x_i^{(k)}\ XOR\ x_j^{(k)}$
10:            end for
11:            $a_{ijl} = \frac{1}{m} \sum_{k=1}^{m} F_{ijl}(x^{(k)}) y^{(k)}\ \forall l = 1, 2, 3, 4, 5$
12:        end for
13:    end for
14:    Return $a_{ijl}$ $\forall l = 1, 2, 3, 4, 5$ and
       FCoeff($x_1^{(1)}, ..., x_j^{(i)}, ..., x_n^{(m)}, y^{(1)}, ..., y^{(m)}, 1$)
15: end function
```

features. The repetition of a feature in the top-ranked propositional features leads to redundant information from a data splitting viewpoint. We prefer non-repeated propositions in the propositional features.

We found a problem in which, if an original Boolean feature has a high coefficient, it might appear many times in the new created features as one of the two features in the pairwise operation. For example, if a feature, say F_1, has a high coefficient, then it might be possible that we can find (F_1 AND F_2), (F_1 AND F_5), (F_1 OR F_6), and (F_1 AND $\neg F_9$) all have high coefficients. This is not what we want to see, because the extra information given by these features is not as much as we expected. (F_1 AND F_2) only gives extra information on F_2 compared to F_1 itself, not as much as other features like (F_3 AND F_4). It is not a good idea to simply eliminate the operations with repeated features as we lose all the extra information combined with F_1.

In the next step we compute coefficients for all the $n + 5 \times \binom{n}{2}$ features and rank them in descending order of importance.

3.4 Selecting the top-ranked features using Fourier coefficients

The engineered parity and propositional features are more numerous than the raw data features. Hence, we need to select the most important among them. We continue using Fourier coefficients for this purpose. The calculation of the coefficients of parity features is the same as (3.5). Here we do not know the function $f(x)$, but for each example x, we have its label, which is a value in $\{-1, 1\}$. Therefore we use the label as the value of $f(x)$ for each example. As we do not have all examples in $\{0, 1\}^n$, we use the whole training set as T.

To calculate the coefficients of propositional features, we need not consider subsets. We only need to calculate coefficients for one feature at a time (in other words, set $k = 1$) for both original Boolean features and newly generated features. The coefficient is calculated as

$$a_i = \frac{1}{m} \sum_{x \in T} \chi_i(x) f(x)$$

where T is the training set, m is the number of training examples, $f(x)$ is the label of example x, and $\chi_i(x)$ is the parity of i-th feature (ONE feature).

Then we pick only a few most important features for evaluation. Although features f_i and $\neg f_i$ have opposite sign coefficients, using $\neg f_i$ instead of f_i is treated the same by classification and clustering algorithms. Rule-based classifiers automatically deal with negations, and other classifiers and clustering algorithms use Manhattan distance instead of Euclidean distance between examples. We use the absolute value of coefficients $|a_i|$ to rank the features in the descending order of $|a_i|$.

We select features with the largest $|a_S|$ or $|a_i|$ to provide classification and clustering algorithms. We do not use the Fourier transform based learning, since the number of features we select is no more than 30, which is less than the total number of engineered features. This is insufficient for the Fourier transform based learner.

After ranking, we select only some features corresponding with the largest $|a_i|$; we do not use these features for Fourier transform, but rather feed them into classification and clustering algorithms. We use at most only the top 30 features in each experiment, which is very few compared to the total number of features. This is insufficient for Fourier transform, especially when we can estimate the coefficients only from the training set T, not the entire set $\{0,1\}^n$. But for clustering and classification, 30 could be a proper number, as it is much smaller than the number of raw features, which is what we want to see. Meanwhile, very high dimensions are ill suited to clustering algorithms because data points will be too sparse, making it difficult to find clusters.

3.5 Comparison with other feature selection or transformation methods

As mentioned in the related works, there are mainly two ways of engineering features: feature selection and feature transformation. Feature selection methods, such as lasso [14] and random forest importance [48] measurement, usually give a score for each feature showing how important they are in classification. If a feature is very valuable in classification, it will be given a high score. If a feature does not contribute to classification (*i.e.* erasing this feature has little impact on accuracy), it will be given a low score. If we use such methods, we can keep some of the most important features and erase others. This can eliminate the influence of useless features and thus help prevent over-fitting, and also simplify the classifier model and shorten processing time. Meanwhile, the selected features are still raw features and do not change at all, so the physical meanings are preserved. However, because selected features are still raw features, the training and testing time of the classifier is still much longer than using Boolean features, especially for classifiers like SVM since the kernel (e.g., distance, inner production, etc) calculation is much easier for Boolean vectors than numeric vectors. Besides, the robustness against mimicry attack is not improved because the features are not changed.

The other way involves using feature transformation methods, such as PCA, that work well in many fields but may not be suitable for this field. Since such methods transform features, they usually do not retain the physical meanings of the features, making it harder for manual analysis when necessary. For example, PCA uses a projection matrix to project the raw features into another lower-dimensional features space. Every new feature in the new feature space is a linear combination of the raw features. The advantage of this method is that it keeps nearly all the information provided by the raw features and only needs to discard very little information. With more information, the classifiers will have a greater chance to get higher accuracy. However, a linear combination of raw features, such as $0.3 \times connections + 0.8 \times sent\_bytes - 0.6 \times URL\_length$, is usually impossible for humans to understand. This can be a disadvantage when it is necessary to manually analyze and control some parts of the classification procedure.

For comparison with other feature methods, we chose lasso and random forest importance (RFI) as two baselines. Both the methods can give weights or scores to each feature, so we can rank the features according to their score and select the top ones for classification. We can use each of them to select top features at 3 stages: top raw features, top Boolean features right after Boolean conversion, and top parity features. We use the same classifiers on these selected features to get the accuracy and recall, then compare with our method.

In lasso, a weight vector $\vec{w}$ is trained to minimize the loss function

$$f_{LOSS} = \sum_{1}^{m} (y_i - \vec{w} \cdot \vec{x_i})^2 + \lambda ||\vec{w}||_1$$

where $\vec{x_i}$ is the i-th example, y_i is the label of i-th example, m is the number of training examples, $||\vec{w}||_1$ is the L1-norm of vector $\vec{w}$, and λ is a parameter. When the loss function is minimized, $\vec{w}$ should be a sparse vector connecting x and y. λ controls how sparse $\vec{w}$ would be. The larger λ is, the more concentration will be put on the sparsity of $\vec{w}$ instead of prediction error $\sum_{1}^{m} (y_i - \vec{w} \cdot \vec{x_i})^2$, and the more zeros in $\vec{w}$. We can select different values of λ to control the number of non-zero elements in $\vec{w}$ and thus select the most important features.

In random forest training, tens or hundreds of decision trees will be trained, but each tree only uses a random subset of features and a random subset of examples in training, and uses the same subset of features to predict. The final prediction output comes from voting of all trees. There is an importance measurement in random forest called “MeanDecreaseAccuracy.” It gives a value to each feature, showing how much the accuracy will decrease if the value of this feature is permuted in all trees. The greater this value, the more important this feature for classification. We can therefore calculate this value for all features and rank them from greatest to smallest, then pick the most important ones for classification.

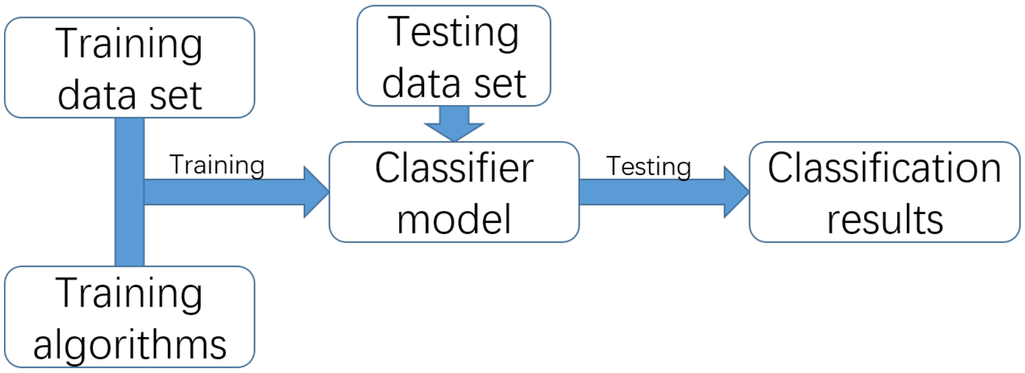
CHAPTER 4

ADVERSARIAL LEARNING

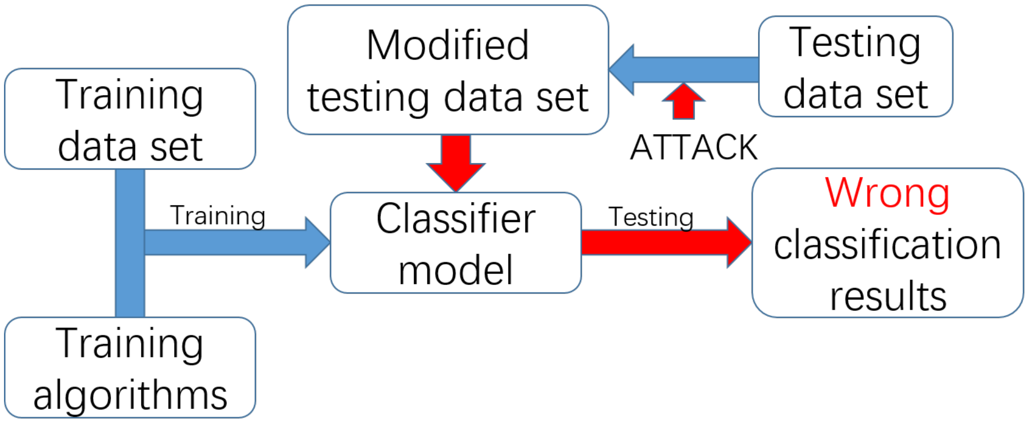
We select adversarial learning as an application of our feature engineering method as well, since this is also an important part of security. According to [49], adversarial learning focuses on dealing with misclassification due to modifications on malicious examples by attackers. If an attacker gets access to some parts of the classifier model, he will be able to find an algorithm that helps malicious examples avoid detection by firewalls, anti-virus systems, etc. Figure 4.1a shows how a regular classifier works, and Figure 4.1b shows how a classifier can be affected under attack. A mimicry attack is one such algorithm, which learns properties of the benign class from some benign examples, then uses such properties to disguise malicious examples and lets them look like benign examples, thus letting them avoid detection. In this chapter, we talk more about how mimicry attacks work, and how we can use our feature engineering method and do some experiments to see if our feature engineering method can deal with such attacks.

4.1 Mimicry attack on PDF files

Mimicus [7] is a tool for mimicry attack on PDF files. Figure 4.2 shows an example of PDF file format. Usually a PDF file has four parts: header, body, cross-reference table, and trailer. By using a widely used feature extractor described in [7], totally 135 features can be extracted from a PDF file. The features are typically author information (author_len, author_num, etc), count of some objects (count_eof, count_image_large, count_javascript, etc), creation date, size, etc. The principle of Mimicus is to add some injected content to the PDF file between cross-reference table and trailer, as shown in Figure 4.3, so that some features can be modified. What kind of content it chooses to add, however, depends on how Mimicus learns about benign



(a) Regular classifier model



(b) Classifier model under attack

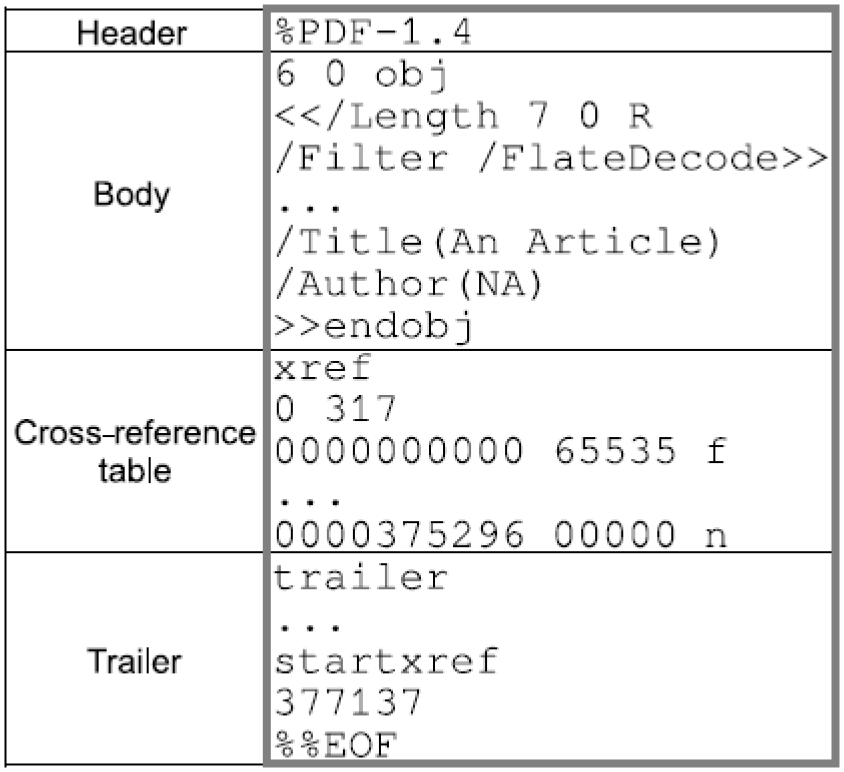
Figure 4.1: One way of attack for malicious examples to avoid detection

example properties. There are several scenarios in which Mimicus can learn from benign examples. Each scenario can be expressed by “F,” “T,” and “C.” “F” means the attacker (Mimicus) knows about the feature set that are used for classification, “T” means the attacker knows a subset of training examples, “C” means the attacker knows which classifier and what algorithm are used in malware detection. For example, “FC” means the attacker knows about the feature set and the classifier, thus he can attack these features and classifier precisely and therefore increase the probability that an attacked malicious example can successfully avoid detection. In theory, FTC should be the most powerful scenario of mimicry attack because the attacker knows the most.

Mimicus tries to add proper contents so that the extracted features look like benign examples. However, by just adding injected contents to PDF files, not all features can be changed perfectly to let them look exactly like benign examples. Among 135 features, 67 features remain the same and cannot be changed by using this method. The remaining 68 features can be modified, but since some features are interdependent of each other, the values cannot be changed exactly as Mimicus wants without any constraints. Figure 4.4 shows how feature values can be changed. The first column shows feature names of a subset of features, the second column (BEFORE) shows the original feature value of a PDF file, the third column (AFTER) shows the target feature value calculated by Mimicus, and the last column (FILE) shows the feature values of the actual modified PDF file. As we can see, there are some differences between target values and actual values due to interdependency between features. Because of the unchanged 67 features and interdependency between the remaining 68 features, classifiers still stand a chance to correctly identify the malicious examples, but only if they are well trained and sufficiently robust.

4.2 Feature engineering for adversarial learning

Since our feature engineering method can produce and select better features than raw features for classification, we believe that our method can invalidate the F in mimicry attack scenarios. This means attackers will not know the exact features we use for classification. However, attackers may still know



Header	`%PDF-1.4`
Body	`6 0 obj`
	`<</Length 7 0 R`
	`/Filter /FlateDecode>>`
	`...`
	`/Title(An Article)`
	`/Author(NA)`
	`>>endobj`
Cross-reference table	`xref`
	`0 317`
	`0000000000 65535 f`
	`...`
	`0000375296 00000 n`
Trailer	`trailer`
	`...`
	`startxref`
	`377137`
	`%%EOF`

Figure 4.2: An example of PDF file format

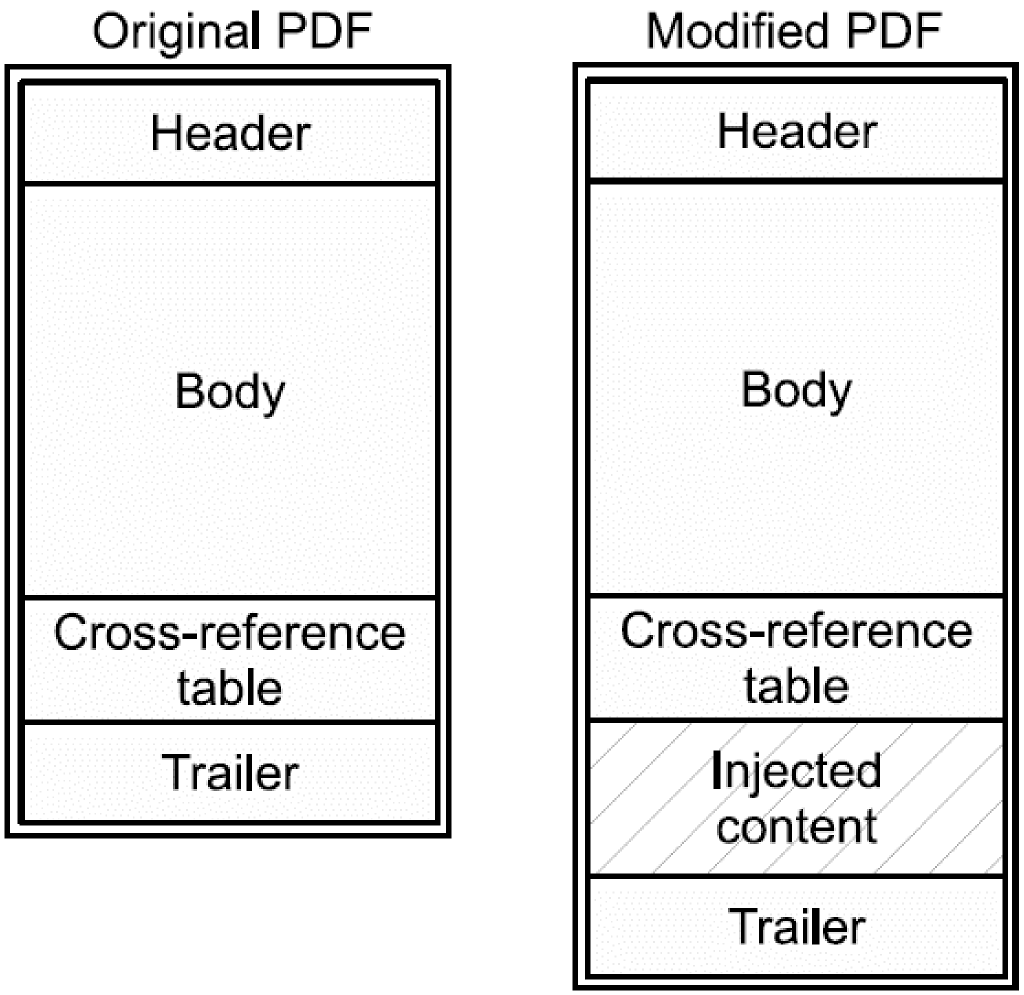
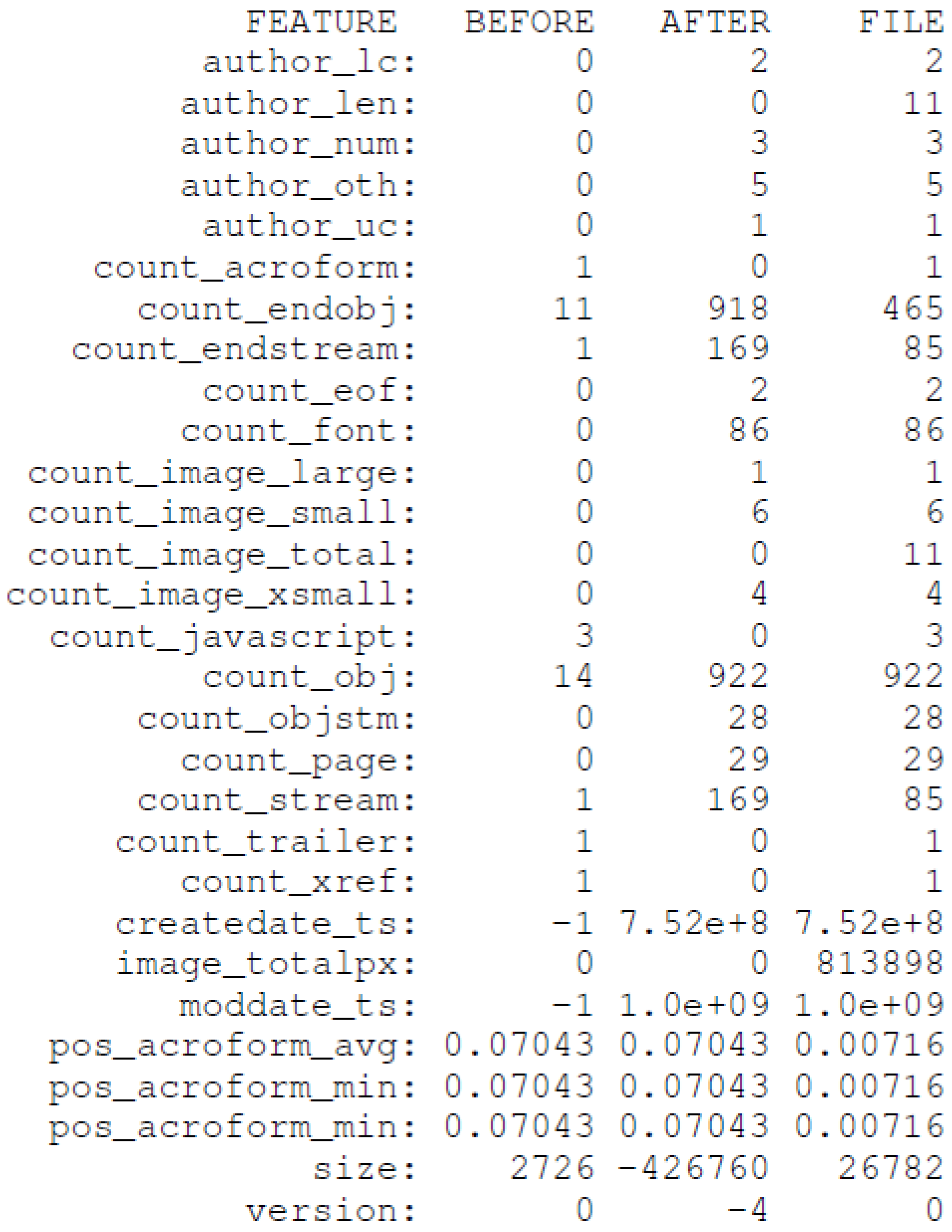


Figure 4.3: How Mimicus changes PDF files to modify feature values



```
           FEATURE   BEFORE   AFTER    FILE
         author_lc:       0       2       2
        author_len:       0       0      11
        author_num:       0       3       3
        author_oth:       0       5       5
         author_uc:       0       1       1
    count_acroform:       1       0       1
      count_endobj:      11     918     465
   count_endstream:       1     169      85
         count_eof:       0       2       2
        count_font:       0      86      86
 count_image_large:       0       1       1
 count_image_small:       0       6       6
 count_image_total:       0       0      11
count_image_xsmall:       0       4       4
  count_javascript:       3       0       3
         count_obj:      14     922     922
      count_objstm:       0      28      28
        count_page:       0      29      29
      count_stream:       1     169      85
     count_trailer:       1       0       1
        count_xref:       1       0       1
     createdate_ts:      -1 7.52e+8 7.52e+8
     image_totalpx:       0       0  813898
        moddate_ts:      -1 1.0e+09 1.0e+09
  pos_acroform_avg: 0.07043 0.07043 0.00716
  pos_acroform_min: 0.07043 0.07043 0.00716
  pos_acroform_min: 0.07043 0.07043 0.00716
              size:    2726 -426760   26782
           version:       0      -4       0
```

Figure 4.4: The original feature value, target feature value calculated by Mimicus, and actual output feature value of a PDF file

about the training file set and the classifier and training algorithm. We will use our feature engineering method on FC and FTC scenarios to see how it can handle the attacks.

Figure 4.5 shows the procedure of using raw features for classification. The blue arrows show the typical classification without attack, and the red arrows show the procedure under a mimicry attack. Theoretically the recall of malicious examples will drop drastically under attack, compared to unchanged malicious examples. Figure 4.6 shows how our feature engineering method could be used in classification. We will apply our feature engineering method on training, testing, and attacked testing examples, producing a new set of features on the examples, and then send them to the training or testing step. Note that the three feature engineering steps in the figure use the same set of RIPPER rules and the same Fourier ranking method generated from the training data set; therefore, the training and testing sets share the same set of features. This is an important point to make sure the classifier works correctly.

From Figure 4.6 and our feature engineering method in chapter 3, we can see that the feature set has completely changed. In this way, the F in scenario FC and FTC is nearly invalidated. Therefore, the effect of the mimicry attack should have been reduced. We will see the detailed results in experiments. However, it is hard to say if such scenarios can still be called “FC” and “FTC”. If F is removed and only C and TC are left, then it might not be fair to compare the results with the baseline, which uses raw features for all the training and testing. If we consider the feature engineering method as a part of the classifier model, or consider the dashed line in Figure 4.6 as a black box, then maybe it can still keep the F and make a fair comparison.

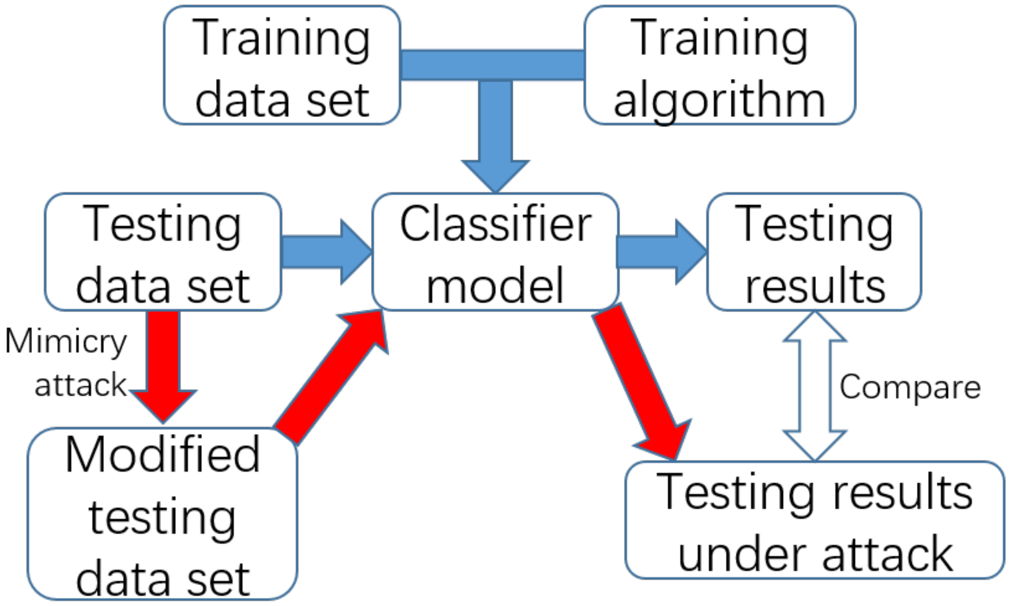


Figure 4.5: The procedure of using raw features for classification, with or without mimicry attack. This is to validate how much a mimicry attack would affect classification, and compare to feature engineering methods as a baseline.

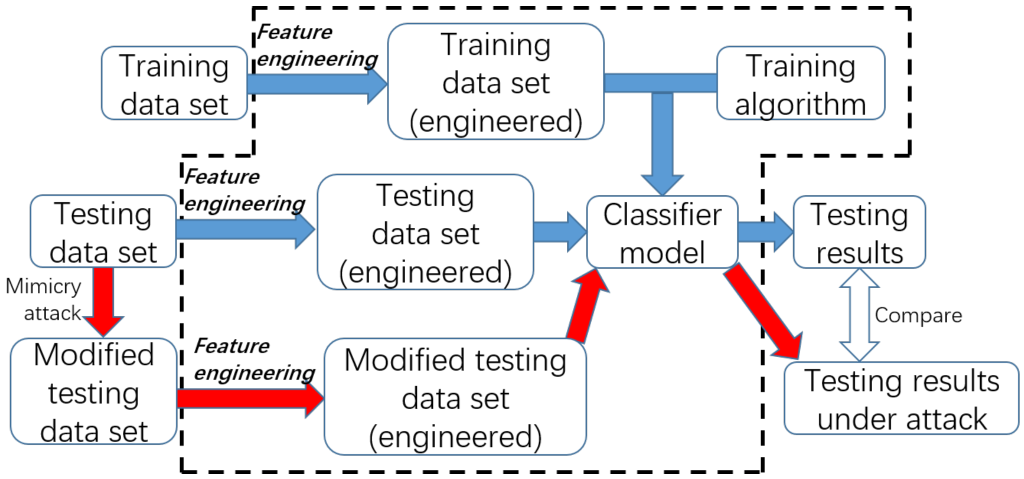


Figure 4.6: The procedure of using feature engineering method to deal with mimicry attack

CHAPTER 5

DATA SETS

This chapter introduces the data sets we use for the experiments. We have two data sets of enterprise log data for testing the performance of engineered features when using 6 classifiers, and another PDF file data set specially for adversarial learning.

5.1 Enterprise log data sets

We have 2 data sets of enterprise log data, containing information about connections between local machines and external web domains. Everything about connections is logged in the log file, including domain name, sent/received bytes, URL, content file name, etc. The data is pre-processed by extracting features for each external domain from all the connections. The raw features are numerical or nominal features such as number of connections, content type, domain age, sent/received bytes, etc. Figure 5.1 shows a small part of the data set. The features shown here are number of hosts, number/average/max/min of connections, total received/sent bytes, and average/max of ratio of received bytes. There are many other features in the set. Each feature value of a certain domain example is extracted from all connection logs between local machines and this domain. For example, when extracting Total_sent_bytes (total sent bytes from local machines to the domain), we look for logs of all connections from local machines to this specific domain, and add up all sent bytes. Each example (domain) has a label of malicious (has aggressive or dangerous behaviors and may attack local machines, steal data, etc), benign (does not behave dangerously), or unknown. We label these data examples using 2 external data bases: Alexa ranking and Virus Total score. Alexa gives ranking of domains in the descending order of global traffic, and Virus Total records how many times a domain is reported malicious. If a domain has a Virus Total score greater than or equal to 3, then it will be labeled malicious. If a domain has a Virus Total score less than 3, and its Alexa ranking is in the top 100k, then it will be labeled benign. The rest of the domains are unlabeled. Though this is already a good way to identify malicious domains, the actually labeled examples are just a small part of the whole data set. In our data set 2, only around 17.6% of the examples can be labeled, and only around 0.5% examples in the whole data set can be labeled malicious. This is far from enough, so we want to train a classifier to do more. We only use the labeled data examples to do the experiments. The scales of two data sets are:

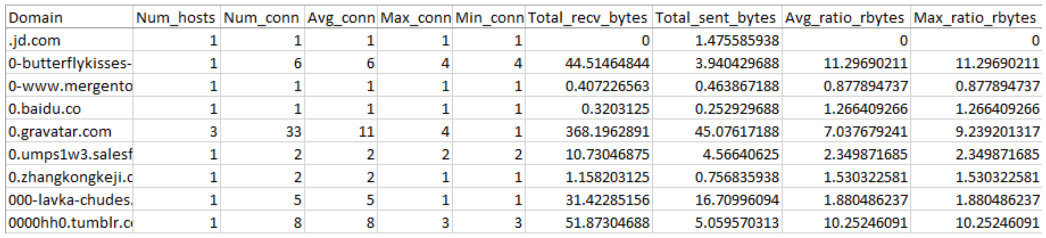
Data set 1 has totally 242,074 examples (thus, 242,074 domains). 45,928 examples are successfully labeled: 43,753 are benign (95.26%) and 2,175 are malicious (4.74%). Each example has 63 features.

Data set 2 has totally 1,116,516 examples. 196,522 examples are successfully labeled: 191,355 are benign (97.37%) and 5,167 are malicious (2.63%). Each example has 91 features.

The original target is to use classification algorithms to learn a model with these examples, so that we can identify benign or malicious domains in the future, and stop local machines from connecting malicious domains in advance. In our experiments, malicious domains are labeled positive, and benign ones are labeled negative.

The numbers of raw features of the two data sets are 63 and 91, not too many, but there are still many redundant features. We used our method on the two data sets to produce top-ranked Boolean features, and compared the classification and clustering results between raw features and our new features.

For adversarial learning, we use another data set in Contagio. It contains 9,000 benign PDF files and 10,980 malicious PDF files. After feature extraction, each PDF file will become an example with 135 features such as author (author_len, author_num, etc), count of some objects (count_eof, count_image_large, count_javascript, etc), create date, size, etc. We will split the data into training and testing sets, then use Mimicus to attack the malicious files in the testing set. We will first use raw features for classification as a baseline, then use engineered features for classification and compare



Domain	Num_hosts	Num_conn	Avg_conn	Max_conn	Min_conn	Total_recv_bytes	Total_sent_bytes	Avg_ratio_rbytes	Max_ratio_rbytes
.jd.com	1	1	1	1	1	0	1.475585938	0	0
0-butterflykisses-	1	6	6	4	4	44.51464844	3.940429688	11.29690211	11.29690211
0-www.mergento	1	1	1	1	1	0.407226563	0.463867188	0.877894737	0.877894737
0.baidu.co	1	1	1	1	1	0.3203125	0.252929688	1.266409266	1.266409266
0.gravatar.com	3	33	11	4	1	368.1962891	45.07617188	7.037679241	9.239201317
0.umps1w3.salesf	1	2	2	2	2	10.73046875	4.56640625	2.349871685	2.349871685
0.zhangkongkeji.c	1	2	2	1	1	1.158203125	0.756835938	1.530322581	1.530322581
000-lavka-chudes.	1	5	5	1	1	31.42285156	16.70996094	1.880486237	1.880486237
0000hh0.tumblr.c	1	8	8	3	3	51.87304688	5.059570313	10.25246091	10.25246091

Figure 5.1: A small part of the data set, including a few examples and a few raw features.

the overall accuracy and recall (malicious) to see if our method gives better results.

5.2 Applying feature engineering to data sets

After Boolean conversion via RIPPER, we got 77 Boolean features for data set 1 (such as $(MaxConnection \geq 9)$, $(DomainAge \leq 37)$, etc), and 98 Boolean features for Data set 2.

We use algorithm 1 for parity features. The number of parity features depends on k. When $k = 1$, we only look at individual original Boolean features. When $k = 4$, we look at subsets of any 1, 2, 3, or 4 features. We have 98 Boolean features at the beginning for data set 2, then setting $k = 1$ gives us totally 98 features, $k = 2$ gives 4,851, $k = 4$ gives 3,769,277, $k = 6$ gives 1,124,298,483. $k = 6$ is nearly 300 times $k = 4$, which takes much more time and memory, but when looking at the coefficients in our observation, the largest coefficients almost appear at subsets of 2, 3 or 4 features. Therefore, we use $k = 2 \sim 4$ in our experiments. When using $k = 4$ we need to start from $k = 1$ and add features one by one, and keep only subsets with $|a_S| \geq 0.7$ at each step, in order to prevent getting out of memory. Here are some examples of the parity subsets: $\{(Num\_ASNs \leq 0), (UA\_Popularity \geq 0.013514)\}$ with coefficient of -0.7650, $\{(Reg\_Age \leq 135), (ASN = UnknownIP), (Levels \leq 2), (Reg\_Validity \leq 1826)\}$ with coefficient of -0.8634, etc. Note that since we do not have the complete training set, Parseval's identity (3.4) is not satisfied here.

We use algorithm 2 to get propositional features. We can get 14,707 propositional features for data set 1, and 24,108 propositional features for data set 2. Here are some examples of the propositional features: $((Sub\_domains \leq 3)$

AND ($Update\_Age \geq 193$)) with coefficient of -0.9158, (($Update\_Validity \leq 1126$) OR ($Length \geq 24$)) with coefficient of 0.5954, etc.

Since the data sets are imbalanced, accuracy mainly reflects the prediction result of benign domains. For example, if we classify all examples as benign in data set 2, we can still get accuracy as high as 97.37%, but this is meaningless for malicious domains. In our task, *malicious examples are more important than benign ones*, since a false positive classification of a malicious domain only causes delay in access, but a false negative classification of a malicious domain would cause damage to the enterprise. We can tolerate false positives more than false negatives in this application.

Due to the importance of malicious domains, we use two metrics to evaluate the result: the overall accuracy $\frac{TP+TN}{All}$, and recall of positive (malicious) $\frac{TP}{TP+FN}$. **Recall of positive is more important than overall accuracy, but it is no more than 70% in our data set when using raw features for classification, sometimes as low as 28%, although accuracy can easily reach 98%.** We can therefore tolerate a decrease in accuracy if it comes with an increase in recall.

5.3 Contagio PDF file data set

For adversarial learning, we use another data set called Contagio. This is a publicly accessible file set for signature testing and research. It contains 16,800 clean (benign) files and 11,960 malicious files of different formats. We will use the PDF files in this data set for our experiments. The PDF files we use are 10,980 malicious files and 9,000 benign files. We use 2/3 of the files (7,320 malicious + 6,000 benign) for training classifier models with raw features and engineered features, then use the remaining 1/3 files (3,660 malicious + 3,000 benign) for testing. The 3,660 malicious files in the test set will be tested before and after mimicry attack. That is to say, our experiment process will be:

1. Raw features as baseline

   Training set: 7,320 malicious + 6,000 benign examples with 135 raw features

Testing set: (a) 3,660 malicious + 3,000 benign examples with 135 raw features without mimicry attack

(b) 3,660 malicious examples with 135 raw features with FC mimicry attack

(c) 3,660 malicious examples with 135 raw features with FTC mimicry attack

2. Boolean features right after Boolean conversion

   Training set: 7,320 malicious + 6,000 benign examples with Boolean features

   Testing set: (a) 3,660 malicious + 3,000 benign examples with Boolean features without mimicry attack

   (b) 3,660 malicious examples with Boolean features with FC mimicry attack

   (c) 3,660 malicious examples with Boolean features with FTC mimicry attack

3. Parity features after Fourier method with $k = 4$

   Training set: 7,320 malicious + 6,000 benign examples with parity features

   Testing set: (a) 3,660 malicious + 3,000 benign examples with parity features without mimicry attack

   (b) 3,660 malicious examples with parity features with FC mimicry attack

   (c) 3,660 malicious examples with parity features with FTC mimicry attack

Since benign files will not be attacked, we only need to test them once in each experiment with different features. We focus on the recall of malicious examples in testing, but we also test benign files in each experiment to make sure the precision and recall of benign examples do not drop too much. The only classifier we use here is random forest since it is what Mimicus attacks for the F scenario, and we need to match the classifier with Mimicus.

CHAPTER 6

EXPERIMENTS

This chapter summarizes the experimental procedures and results. The results of the enterprise log data set include accuracy and recall (malicious) using propositional Boolean features, parity features, and propositional features, when using 6 classifiers. Also included are the timing and efficiency analysis, classification results discussion, comparison with other Boolean conversion methods, comparison with other feature selection methods, and clustering results. For adversarial learning, we present results and discussion of the recall on both benign and malicious examples.

6.1 Experimental setup

For classification, we use 6 algorithms (SVM, AdaBoost, RIPPER, decision tree (DT), random forest (RF), and PEBL) to build models, and evaluate using 3-fold cross validation. The classifiers are:

SVM: Support Vector Machine, a classification algorithm that finds a separator which has the largest margin for either of the two classes (or to any of the classes for multi-class SVM). The separator can be linear or non-linear according to its kernel function.

AdaBoost: A classification algorithm. It consists of a number of very simple classifiers, and learns weights of such classifiers. The final classification hypothesis is the weighted sum of simple classifiers.

RIPPER: Repeated Incremental Pruning to Produce Error Reduction, a rule-based classifier. The output classification model is a DNF.

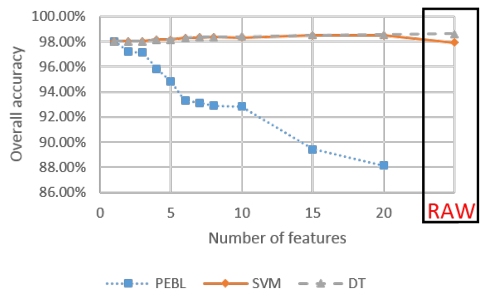
Decision Trees: a tree-structured classification algorithm. Each non-leaf node considers a feature and has several branches for different cases of the value of this feature. Leaves are classes. An example can follow the nodes and branches to a leaf.

Random Forests: a tree-structured classification algorithm that consists of a number of decision trees. Each decision tree uses a random sample of features to build the classifier, and the forest uses the most frequently appearing class among the trees as final classification result.

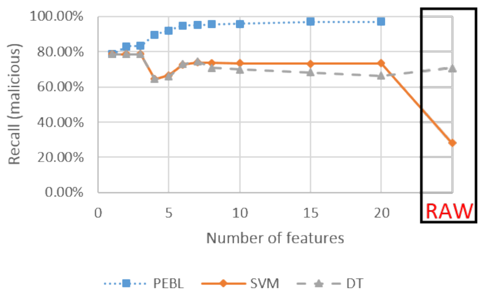
PEBL: Positive Example Based Learning [50], an SVM-based classification algorithm. It deals with the case when the training data only has a few positive examples and a large number of negative or unlabeled examples. It finds a tight boundary containing almost all the positive examples.

These algorithms are some widely used classifiers with different core ideas and measurements. We use them to test if our new features can be used for different types of classifiers. We compare the accuracy and recall (malicious) with raw features using the same classifier to get the improvement. For SVM, AdaBoost, RIPPER, DT, and RF, we use the default parameters of Weka 3.7.12 [51] to build the classifiers. For PEBL, we implement it according to [50]. For clustering, we use EM and k-means algorithm.

PEBL needs to be initialized with "strong negative" examples. We use the top-ranked features to select these examples. For example, when using top 5 features for classification, we look at the top 5 features. If the coefficient of feature 1 (F_1) is positive, then it is more likely that $\chi_{F_1}(x_i)$ and label y_i are both positive or both negative. On the other hand, if the coefficient of F_1 is negative, then it is more likely that one of $\chi_{F_1}(x_i)$ and label y_i is positive and the other is negative. For strong negative examples, we pick those for which all 5 features are more likely to make the label negative, *i.e.*, $\chi_{F_j} = -1$ if $a_{F_j} > 0$ or $\chi_{F_j} = 1$ if $a_{F_j} < 0$. For example, if the coefficient of top 5 features are $0.6, -0.5, -0.4, 0.4, 0.2$, then an example will be marked as strong negative if and only if $\chi_{F_1} = -1, \chi_{F_2} = 1, \chi_{F_3} = 1, \chi_{F_4} = -1, \chi_{F_5} = -1$, or equivalently, $F_1 = TRUE, F_2 = FALSE, F_3 = FALSE, F_4 = TRUE, F_5 = TRUE$. As for raw features, this method does not work because it needs to use the Fourier coefficients to identify strong negatives but we do not have coefficients for raw features. In this case, the only way we can find strong negatives is to calculate the distance between each positive example and each



(a) Overall accuracy



(b) Recall of malicious domains

Figure 6.1: The accuracy and recall (malicious) of the 3 classifiers using the top 1-20 features, compared with raw features, on data set 2. PEBL gets up to 96.96% recall with 88.15% accuracy, while others get 97.9%-98.6% accuracy and up to 78.61% recall.

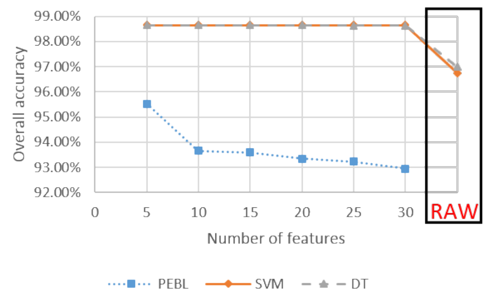
negative example, and then some negative examples will be labeled strong if they are far away from all positive examples. However, this is too time-consuming and we cannot afford doing this. Therefore, we do not have PEBL classification results when using raw features.

6.2 Performance of propositional (Boolean) features

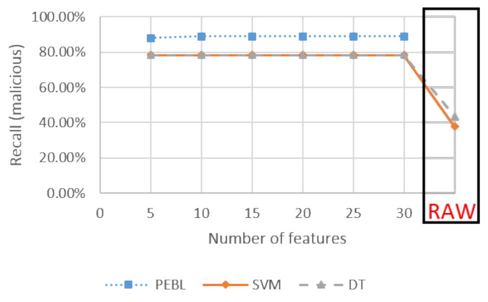
In this experiment, we only looked at the individual Boolean features (equivalent to $k = 1$). Although these are a part of propositional features, we analyze them as a separate class of features. The top 1-20 features were selected for classification.

Figure 6.1a and 6.1b show the accuracy and recall (malicious) of the 3 classifiers using top 1-20 features, compared with raw features, on data set 2. Note that the last point of each line is the result of using raw features on the classifiers. AdaBoost, RIPPER, and RF are not shown here. They have very similar accuracy as DT. SVM has similar recall (malicious) as DT for engineered data, but 40% lower for raw features. RIPPER and RF have similar recall (malicious) as DT.

As shown in the result, regardless of the number of features we used, SVM, RIPPER, DT, and RF have similar accuracy compared with raw features respectively. When using 6-7 features, SVM, RIPPER, DT, and RF get a bit higher recall than the raw features. PEBL is an outlier that gets much higher recall and much lower accuracy compared to other 5 classifiers. This shows



(a) Overall accuracy



(b) Recall of malicious domains

Figure 6.2: The accuracy and recall (malicious) of the 3 classifiers using the top 5-30 parity features on data set 1. PEBL gets up to 88.97% recall with 92.95% accuracy, while others get 98.5%-98.7% accuracy and 76.3%-78.5% recall and are all higher than raw.

that we can usually get higher recall and nearly same accuracy when we use much fewer engineered features than raw features. PEBL is an algorithm that focuses on positive examples much more than negative ones, so it trades off accuracy for recall of malicious domains. It discovers nearly all the positive examples, but also leads to false positives.

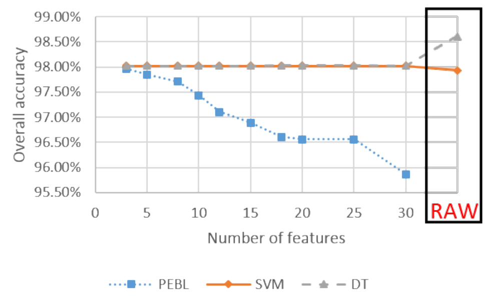
In conclusion, the engineered propositional (Boolean) features get higher recall of malicious domains, and a bit higher or lower accuracy depending on the classifier, on data set 2.

As we can see, the accuracy of PEBL drops very fast and can easily go below the accuracy of random guess, while the accuracies and recalls of the other 5 classifiers nearly reach convergence at 20 features. Since the number of raw features is not so large (63-91), it is not meaningful to use too many engineered features for classification. Therefore, we decide that the max number of engineered features used in our experiments should be 20-30.

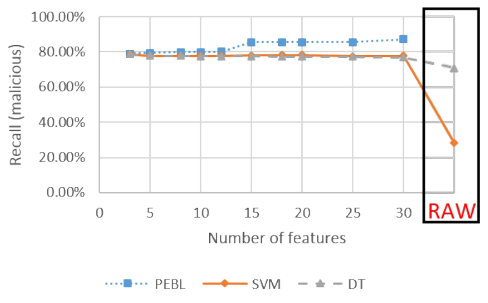
6.3 Performance of parity features

In this experiment we consider the parity features. For data set 1 we set $k = 2$ and for data set 2 we set $k = 4$. The top 5-30 features were used for classification.

Figure 6.2a and 6.2b show the accuracy and recall (malicious) of the 3 classifiers using top 5-30 parity features on data set 1. AdaBoost, RIPPER, and RF are very close in accuracy to DT for engineered features, and their



(a) Overall accuracy



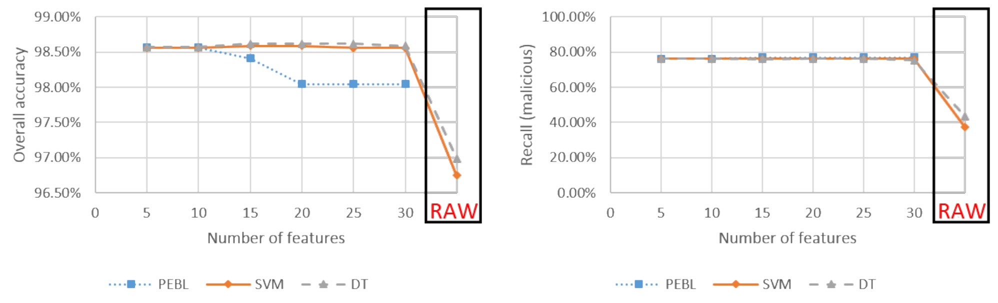
(b) Recall of malicious domains

Figure 6.3: The accuracy and recall (malicious) of the 3 classifiers using the top 5-30 parity features on data set 2. PEBL gets up to 87.25% recall with 95.87% accuracy, while others get 98.0% accuracy (lower than raw) and 60.8%-78.6% recall (higher than raw).

accuracy for raw features ranges from 96.65% to 98.56%. Recall (malicious) of RIPPER and RF are nearly the same as DT for engineered features, and 39.2% and 59.5% for raw features. Recall (malicious) of AdaBoost is 2% lower than DT, and 69.70% for raw features. Figure 6.3a and 6.3b show the same result on data set 2. AdaBoost, RIPPER, and RF have accuracy and recall very close to DT, but they have lower accuracy and higher recall (malicious) than raw features, which have 97.95% to 98.82% accuracy and 59.6% to 70.5% recall of malicious domains.

For data set 1, the engineered parity features seem to be very good, as we can always get both higher accuracy and higher recall for all 5-30 features compared to raw features. For data set 2, we can only get higher recall with lower accuracy, but the improvement in recall is much higher than drop of accuracy, which is acceptable for our application. PEBL trades off too much accuracy for recall but fortunately we can control the number of features to decide how far it goes in this trade-off.

In conclusion, the engineered parity features get both higher accuracy and higher recall of malicious domains on data set 1, higher recall of malicious domains on data set 2, and a bit higher or lower accuracy depending on the classifier on data set 2.



(a) Overall accuracy (b) Recall of malicious domains

Figure 6.4: The accuracy and recall (malicious) of the 3 classifiers using the top 5-30 propositional features on data set 1. PEBL gets up to 77.01% recall with 98.04% accuracy, while others get 98.56%-98.62% accuracy and 75.1%-76.3% recall and are all higher than raw.

6.4 Performance of propositional features

In this experiment we considered only the top 5-30 propositional features. Figure 6.4a and 6.4b show the accuracy and recall (malicious) of the 3 classifiers using the top 5-30 propositional features on data set 1. AdaBoost, RIPPER, and RF have similar accuracy and recall (malicious) to DT for engineered features, and 96.65% to 98.56% accuracy and 39.2% to 69.7% recall (malicious) on raw features.

In conclusion, the engineered propositional features get both higher accuracy and higher recall of malicious domains on data set 1.

6.5 Timing comparison between raw and engineered features

Table 6.1 shows timing of 5 classifiers using raw features and 5 or 30 engineered features on data set 2. PEBL is not included here because it is hard for raw features. For raw features, the timing is just the time of classification. For engineered features, the timing includes Boolean conversion, Fourier ranking, and classification. Typically, Boolean conversion takes the same time for all tests on the same data set, but Fourier ranking takes much more time for $k = 4$ than $k = 1$, and classification takes more time for 30 features than 5 features.

Table 6.1 shows the exact amount of time. Here we found that for SVM,

Table 6.1: Timing comparison between raw and engineered features (seconds)

Classifier	Engineered k=1		Engineered k=4		Raw
	5 features	30 features	5 features	30 features	
SVM	1109.91	1405.29	4843.54	5138.92	134801.2
AdaBoost	667.14	675.87	4400.77	4409.5	224.13
RIPPER	673.26	993.63	4406.89	4727.26	2618.4
DT	665.19	694.26	4398.82	4427.89	890.76
RF	708.24	990.69	4441.87	4724.32	833.97

all tests on engineered features take much less time than raw features; for AdaBoost, all tests on engineered features take more time than raw features; for RIPPER, decision trees, and random forests, tests on engineered features of $k = 1$ take the same or less time than raw features, but tests of $k = 4$ take more time. From the previous section we know that even $k = 1$ can get much improvement in recall over raw features, but $k = 4$ does not have much improvement over $k = 1$, so we can say that $k = 1$ is more efficient. Meanwhile, we found that our feature engineering method is good for SVM, RIPPER, decision trees, and random forests, but not so good for AdaBoost.

In conclusion, propositional Boolean features ($k = 1$) is more efficient for the whole process, but parity features may get higher recall and accuracy.

6.6 Analysis of feature engineering on classification

In data set 2, we observe a trade-off between accuracy and recall. However, the engineered features achieve a significant improvement in recall while not sacrificing too much accuracy. Figure 6.5 shows the extent of improvement in recall, as well as the corresponding change in accuracy for the engineered features. The engineered features work best with SVM. This might be because SVM needs to consider kernel calculation, but raw features are in different formats, and the value scale differs a lot between different features. Boolean and parity features, however, are in the same format, making it a lot easier for SVM to separate examples between classes. Parity features of data set 1 show the best results with improvement in both accuracy and recall (malicious). In general, parity features give better results than propositional features, and Boolean features give nearly the same results as parity features.

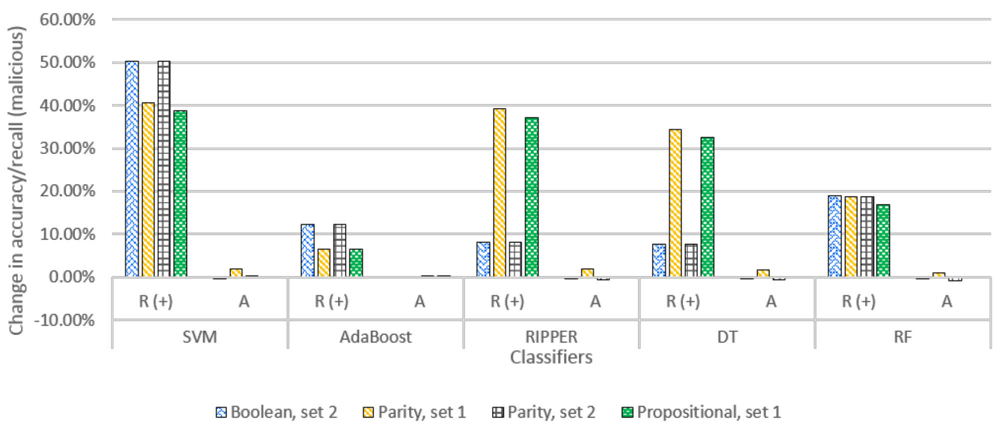


Figure 6.5: Change in accuracy (A) and recall of malicious domains (R(+)) compared with raw features, when getting highest recall with engineered features. Change in accuracy is very little compared to recall and drop in accuracy is acceptable.

The drop in accuracy is acceptable considering of the improvement in recall.

Table 6.2 shows our top-ranked features and a domain expert's interpretation of how they are related with identifying malicious domains. Some features may appear several times with different thresholds (such as $Sub\_domains \leq 2$, etc.). This means the feature engineering method has provided top-ranked features with reasonable explanations that are meaningful to domain experts. In comparison to feature transformation approaches, this is the advantage of our approach.

Figure 6.6 shows the maximum improvement in recall of malicious domains (averaged across 5 classifiers) and the corresponding average number of features used to achieve this maximum improvement. The change in accuracy is very small (shown in Figure 6.5) compared to recall, so we do not need to consider it. For data set 1, 5 propositional features average around 26% improvement in recall, while 10 parity features are required to reach around 28%. For data set 2, Boolean features used fewer features to get nearly the same improvement as parity features (1-3 features for averagely 20% of improvement in recall), while propositional features used more features without as much improvement in recall.

PEBL is an algorithm different from other ones that focuses on recall much more than accuracy. Figure 6.7 shows the trade-off between accuracy and recall (malicious) for Boolean and parity features, and Table 6.3 shows the

Table 6.2: Explainability of top-ranked features

Features	Explanations
$Sub\_domains \leq 1$	Most legitimate domains have many sub-domains on second-level, but malicious domains have few of them.
$ASN = UnknownIP$	We were not able to resolve the IP address of the domain, meaning that the domain is no longer registered. This is suspicious as most likely the domain has been taken down.
$Levels \leq 2$	Many malicious domains have 2 or 3 levels.
$Reg\_Age \leq 69$	Malicious domains are in general more recently registered than legitimate ones.
$Reg\_Validity \leq 365$	Malicious domains are registered for shorter periods of time to reduce attacker cost.
$Length \geq 13$	Long URLs are more suspicious since they might be used for malware communication to a command-and-control center.
$Avg\_ratio\_rbytes \geq 103.634489$	The ratio of bytes sent over bytes received. For legitimate web sites, usually more content is received than sent by end hosts.
$Update\_Validity \leq 365$	Number of days from update till expiration. Malicious domains have lower expiration dates than legitimate ones.

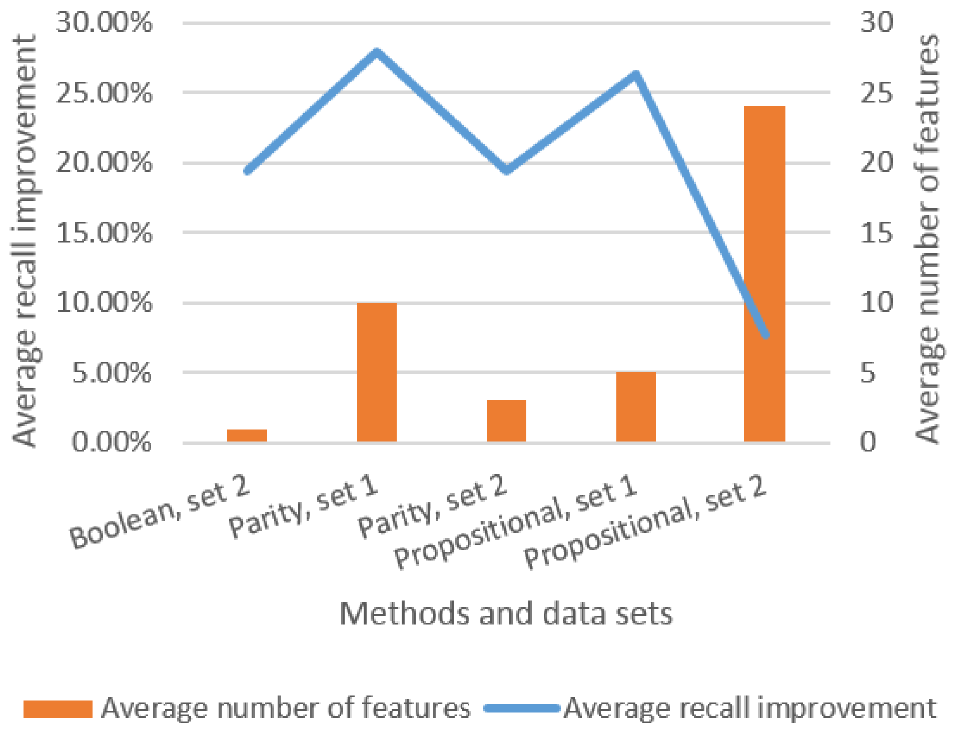


Figure 6.6: Maximum improvement in recall of malicious domains (line, higher is better) and the corresponding number of features used to achieve this maximum improvement (column bars, lower is better). Parity and Boolean features are better than propositional features.

Table 6.3: Result for highest recall (malicious) and corresponding accuracy by PEBL

Method	Data set	Accuracy	Recall
Original Boolean	2	88.15%	96.96%
Parity k=2	1	88.97%	92.95%
Parity k=4	2	95.87%	87.25%
Propositional	1	98.04%	77.01%
Propositional	2	98.03%	78.61%

exact value of highest recall and the corresponding accuracy. Note that in each experiment, the highest accuracy always corresponds to lowest recall, and the highest recall always corresponds to lowest accuracy. It is possible to control this trade-off of PEBL by choosing the number of features. When initializing the strong negative examples, the more features we use, the stricter the standard we use to identify strong negatives (see section 6.1). The fewer strong negatives we use initially, the more false positives and fewer false negatives we will get.

6.7 Comparison with other Boolean conversion methods

Since there are many Boolean conversion methods to convert raw features into Boolean features, RIPPER is a more effective way in this application compared to others. Actually, other rule-based classifiers can also do this, such as decision trees, but in decision trees every node is developed on the basis of its parent and parents' parent etc., so the stand-alone propositions may not be very effective without its parent. RIPPER is better here because every rule in a DNF stands alone, and every proposition is only restricted by other propositions before it in the same rule. This is different from decision trees, where all other propositions are restricted by the root. As for other Boolean conversion methods without using a classifier, we ran an experiment to compare RIPPER to one such method in Weka, namely supervised discretization, which uses mutual information to perform Boolean conversion.

We use RIPPER and Weka's discretization on data set 2 raw features to generate two sets of Boolean features, and use both of them for Fourier

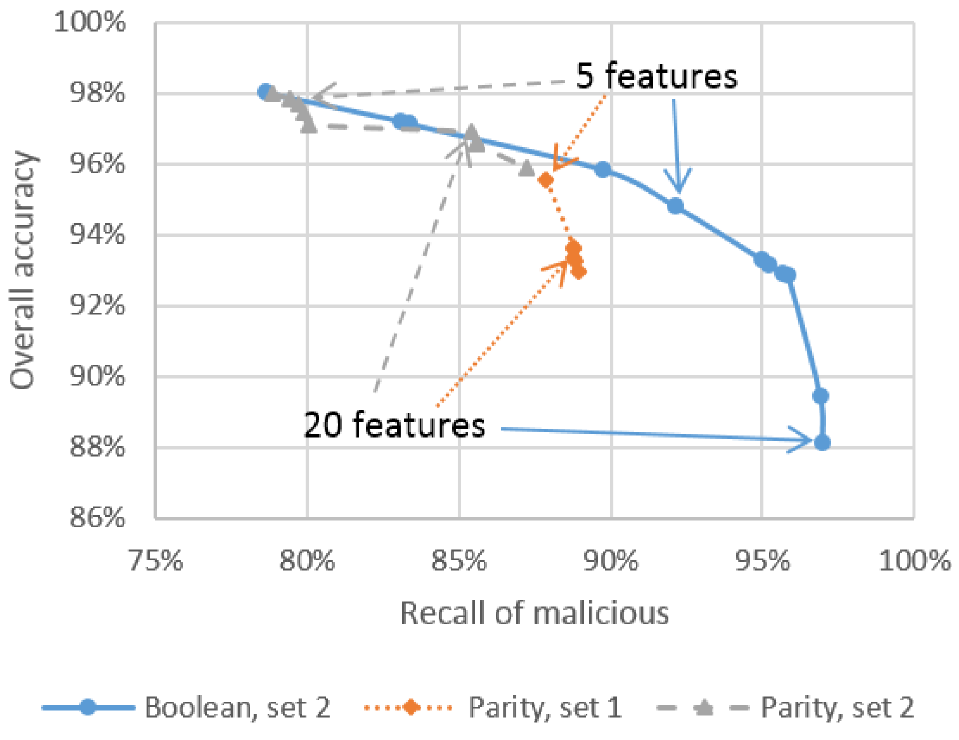
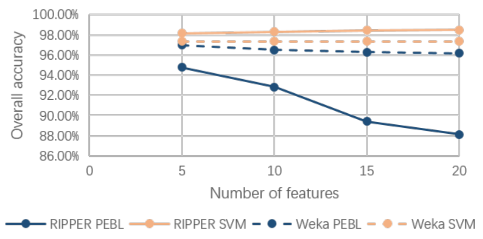
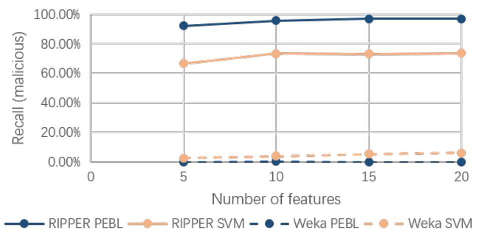


Figure 6.7: The trade-off between accuracy and recall (malicious) for Boolean and parity features using PEBL. The arrows show the position when using 5 or 20 features.



(a) Overall accuracy



(b) Recall of malicious domains

Figure 6.8: The accuracy and recall (malicious) of PEBL and SVM using top 5-20 Boolean features generated by two Boolean conversion methods on data set 2. Although Weka's discretization gets higher accuracy with PEBL, its recall is as low as <10% and therefore useless.

method with $k = 1$ (single feature ranking). Figure 6.8a and 6.8b show the result. We can see that Weka's discretization gets higher accuracy with PEBL, but the recall of Weka's converted features is too low to reach even 10%, which means these features are useless when we need to focus on malicious examples.

In conclusion, RIPPER is much better than this mutual information based Boolean conversion.

6.8 Comparison with other feature selection methods

In this experiment, we use lasso and random forest in three positions to take the place of Fourier method, and see the performance. They will be used for: raw features, Boolean features after Boolean conversion, and parity features. For raw features, we give all raw features to lasso and RFI and let them rank the features. Since Fourier method is the whole procedure, it cannot be compared using raw features. For Boolean features, we give all Boolean features generated by RIPPER propositions to lasso, RFI, and Fourier method, and let them rank the features. For parity features, we wanted to generate all parity features for $k = 4$ and then give them to feature selection methods, but as there are nearly 4 million parity features that will cost too much space and time, our actual procedure is to first select the top 1000 parity features according to Fourier coefficients, and then give them to lasso and RFI to rank. The top 30 features with highest Fourier coefficients will also be used in comparison. We use our data set 2 with

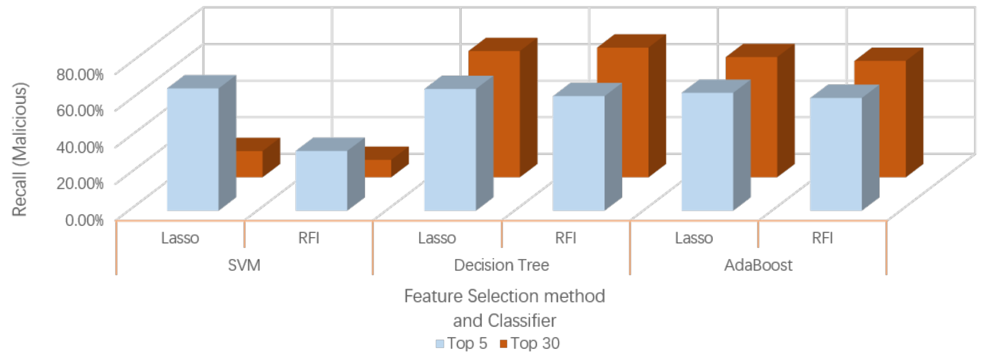
Table 6.4: Top-ranked Boolean features selected by lasso, RFI, and Fourier method

Feature rank	Lasso	RFI	Fourier
1	Sub_domains ≤ 1	Num_hosts ≤ 1	Sub_domains ≤ 1
2	Reg_Age ≥ 10	Num_ref_doms ≤ 1	ASN = UnknownIP
3	Num_ASNs ≤ 0	Num_ASNs ≤ 0	Num_ASNs ≤ 0
4	Reg_Age ≤ 69	Avg_depth ≥ 1.93	Levels ≤ 2
5	Reg_Validity ≤ 365	Max_depth ≤ 3	Sub_domains ≤ 2

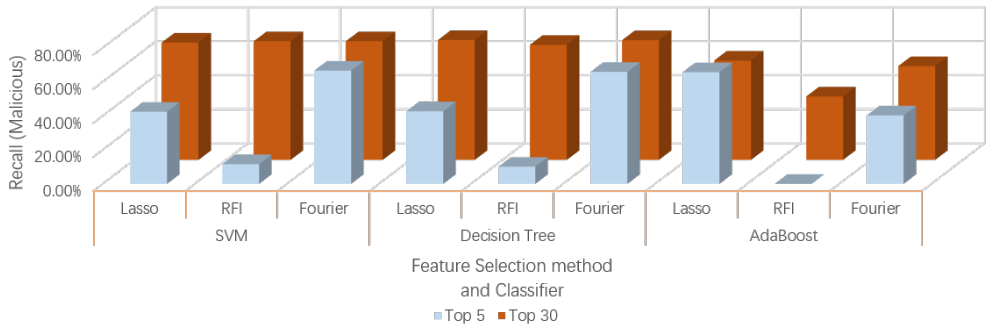
SVM, decision trees, and AdaBoost for evaluation. The results show that the accuracy values are very close to each other among all experiments (maximum difference less than 1%), but recall values are not, so we only need to look at the recall of malicious examples.

The results are shown in Figure 6.9a, 6.9b, and 6.9c, for top raw features, top Boolean features after Boolean conversion, and top parity features for $k = 4$. In Figure 6.9b, we can see that RFI performs the worst, Fourier method performs better for SVM and decision tree especially when using top 5 features, but it is beaten by lasso in AdaBoost. However, in Figure 6.9c, Fourier method wins every one of the experiments regardless of classifiers or number of features. By comparing Figure 6.9a and 6.9c, we can also see that Fourier method ranked parity features beat lasso and RFI ranked raw features in nearly every experiment. If we compare the performance of lasso and RFI across 3 figures, we can see that raw features sometimes get even better recall than Boolean and parity features, and parity features are not always better than Boolean features. This might mean that our Boolean conversion method and parity features do not suit lasso and RFI so well as Fourier method. Besides, we can see that the recall difference between the top 5 and 30 features is very large in some cases of lasso and RFI, sometimes reaching 58%. The reason might be that lasso and RFI do not get consistent feature importance. In this way, top features may not be truly important for classification, but a lower-ranked feature may drastically improve the recall. Since Fourier method focuses only on Boolean features and parity features are designed for Fourier methods, such result is not surprising.

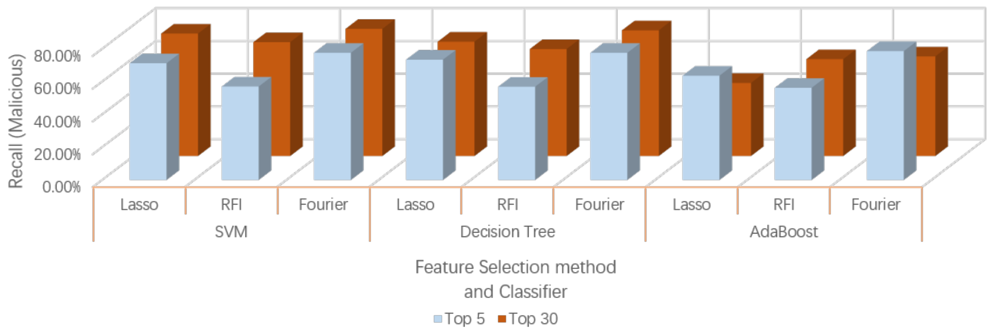
The top 5 Boolean features in data set 2 selected by each method are shown in Table 6.4. As we can see, the 3rd feature is the only one in common for all 3 methods, and the 1st feature is the same for lasso and Fourier method,



(a) Comparison of recall (malicious) using 3 classifiers with top 5 and 30 raw features ranked by lasso and random forest importance.



(b) Comparison of recall (malicious) using 3 classifiers with top 5 and 30 Boolean features (after Boolean conversion) ranked by lasso, random forest importance, and Fourier coefficients. RFI works the worst, Fourier method is the best in SVM and decision tree especially when using 5 features, lasso is the best in AdaBoost.



(c) Comparison of recall (malicious) using 3 classifiers with top 5 and 30 parity features ($k = 4$) ranked by lasso, random forest importance, and Fourier coefficients. Fourier method wins in every experiment, and is better than raw features ranked by lasso or RFI in 6.9a.

Figure 6.9: Comparison of recall using 3 classifiers with 3 types of features and 3 types of feature ranking methods

but all the rest are different. This shows that the different methods use very different principles and get very different ranking output. From Figure 6.9 we can see that the Fourier features work the best among 3 methods, at least for this data set.

In conclusion, the Fourier method is better than lasso or RFI in generating and selecting features, especially with larger k values.

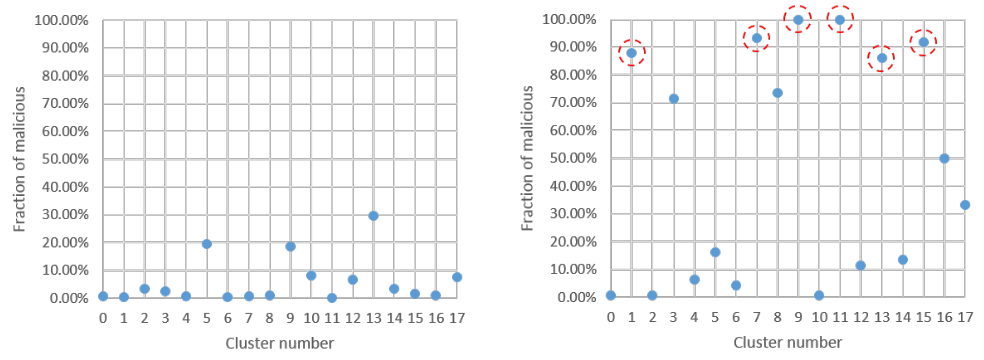
6.9 Clustering with engineered features

We compare the clusters obtained from raw features and engineered features. The evaluation method is to look at the fraction of positive (malicious) examples in each cluster according to the labels. If the majority of a cluster is positive, then we can say this cluster is a “pure” positive cluster, and its characteristic (cluster centroid) is a representative of positive examples. In the same way, if the majority of a cluster is negative, then it is a “pure” negative cluster, and its characteristic (cluster centroid) is a representative of negative examples.

We first used EM algorithm to roughly find the proper number of clusters, then used k-means algorithm with manually decided number of features. Figure 6.10a shows the fraction of positive examples in each cluster when using 63 raw features in data set 1. There are many “pure” negative clusters, but no positive clusters. Meanwhile, each cluster contains from 1.26% to 10.11% of all examples. This is a poor result, since the examples are distributed into 18 clusters that are not “pure” enough to distinguish them as malicious or benign.

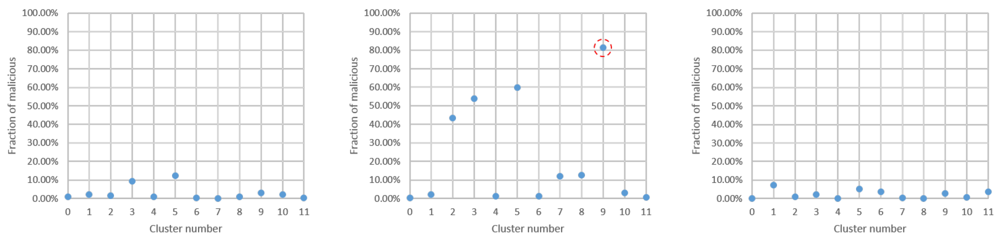
Figure 6.10b shows the fraction of positive examples in each cluster when using the top 10 parity features (limited $k = 2$) of data set 1. Now, there are 6 “pure” positive clusters: cluster #1, #7, #9, #11, #13, #15. The biggest cluster #0 contains 89.99% of the examples, and is a “pure” negative cluster.

Figure 6.11a and 6.11b show the result of a similar experiment on data set 2: 91 raw features compared to top 20 parity features (limited $k = 4$). Our method has a similar effect with data set 1: number of “pure” positive clusters increased from 0 to 1 (cluster #9), and found a big “pure” negative cluster #0 that contains 95.07% of all examples, while the fractions of positive examples are 2.99% to 19.17% in clusters with raw features. Figure 6.11c is



(a) Raw features (b) Engineered features

Figure 6.10: Comparison of fraction of malicious domains in each cluster between raw features and engineered features on data set 1. “Pure” positive clusters are shown by dotted circles.



(a) Raw features (b) Engineered features (c) RFI selected features

Figure 6.11: Comparison of fraction of malicious domains in each cluster between raw features, engineered features, and random forest importance selected features, on data set 2. “Pure” positive clusters are shown by dotted circles.

the clustering result using the top 20 features selected by the random forest importance method. This result is worse than raw data, since no cluster has more than 10% malicious examples.

In conclusion, the engineered features help in getting “pure” clusters and summarizing properties of such clusters.

6.10 Results of adversarial learning

In this experiment we use 7,320 malicious + 6,000 benign PDF files in Contagio data set for all the training work, including generating RIPPER rules and propositions for Boolean conversion and building classifier models. Then we use the rest 3,660 malicious + 3,000 benign files for testing. The 3,660 malicious files will be tested 3 times, in status of unattacked, after FC mimicry attack, and after FTC mimicry attack. Since benign files are not attacked, we look at them separately with malicious files.

We have originally 135 raw features. After Boolean conversion, we got 18 Boolean features, and used all 18 features for classification. After Fourier method, we select top 30 parity features ($k = 4$) for classification. Since this mimicry attack focuses on random forest, we only use random forest as the classifier. The result is shown in Table 6.5. We can see that the mimicry attack is effective for raw features, making the recall of malicious files dropped down to 3.4%. However, the Boolean features can already handle mimicry attack to some extent, and parity features can further reduce the influence of mimicry attack, boosting the recall back to above 96%, although the recall of unattacked files dropped a little bit. The last row shows the recall of benign files to show if the identification of benign files changes when using different features. The output shows that Boolean features can get a bit higher recall on benign files but parity features get a bit lower than raw. Considering the drastic improvement of recall of malicious files, we think this little drop is acceptable.

The improvement of robustness mainly comes from Boolean conversion. After conversion, only 18 Boolean features are generated. As there are 135 raw features, most of the raw features are meaningless for classification and are discarded. Moreover, there is a threshold in each of the propositions used to generate Boolean features, such as the number "10" in proposition "count_eof<10". As long as the modified (attacked by mimicry) feature value does not go to the other side of the threshold, the Boolean feature value remains unchanged. Since only half of the 135 features can be changed and not freely, the Boolean conversion could improve much of robustness. Furthermore, Fourier method and parity features discover the correlation between features and thus get more useful information for classification.

In conclusion, the engineered features invalidate the F scenario in mimicry attack and maintain a very high level of recall in detecting malicious examples, while not losing too much recall on benign examples.

Table 6.5: Recall of unattacked malicious files, attacked malicious files, and benign files, when using raw features, Boolean features, and parity features

Recall	Raw features	Boolean features	Parity features
Unattacked	99.9%	99.9%	99.5%
FC_mimicry	36.0%	89.0%	96.7%
FTC_mimicry	3.4%	88.1%	98.4%
Benign files	97.9%	98.2%	96.1%

CHAPTER 7

CONCLUSION

In conclusion, we have shown that the feature engineering approach benefits classification and clustering algorithms. It produces higher performance with respect to raw data and feature selection approaches, while reducing the dimensionality of the problem significantly. Even very few of the engineered features are able to produce a high recall of malicious domains in the enterprise log data, while retaining interpretability by a human. Meanwhile, features generated by our method are much more robust to mimicry attack. In general, the parity features outperform the propositional features, arguing for the more formal, harmonic analysis method of feature engineering.

REFERENCES

[1] J. Zorabedian, “Anatomy of a ransomware attack: CryptoLocker, CryptoWall, and how to stay safe,” Available from blogs.sophos.com, March 03, 2015.

[2] Mandiant, “Apt1: Exposing one of China’s cyber espionage units,” Report available from www.mandiant.com, February 18, 2013.

[3] “Verizon 2015 data breach investigations report,” http://www.verizonenterprise.com/DBIR/2015/, April 13, 2015.

[4] J. Steinberg, “Massive security breach at Sony – here’s what you need to know,” Available from www.forbes.com, December 11, 2014.

[5] J. Steinberg, “Massive breach at health care company Anthem Inc,” Available from www.usatoday.com, February 05, 2015.

[6] R. O’Donnell, J. Wright, and Y. Zhou, “The Fourier entropy–influence conjecture for certain classes of boolean functions,” in *Automata, Languages and Programming.* Springer, 2011, pp. 330–341.

[7] P. Laskov et al., “Practical evasion of a learning-based classifier: A case study,” in *2014 IEEE Symposium on Security and Privacy.* IEEE, 2014, pp. 197–211.

[8] C. Smutz and A. Stavrou, “Malicious PDF detection using metadata and structural features,” in *Proceedings of the 28th Annual Computer Security Applications Conference.* ACM, 2012, pp. 239–248.

[9] S. Yang, L. Yuan, Y.-C. Lai, X. Shen, P. Wonka, and J. Ye, “Feature grouping and selection over an undirected graph,” in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2012, pp. 922–930.

[10] X. Cai, F. Nie, and H. Huang, “Exact top-k feature selection via l 2, 0-norm constraint,” in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence.* AAAI Press, 2013, pp. 1240–1246.

[11] Q. Gu, Z. Li, and J. Han, "Joint feature selection and subspace learning," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1. Citeseer, 2011, p. 1294.

[12] J. Wang and J. Ye, "Multi-layer feature reduction for tree structured group lasso via hierarchical projection," in *Advances in Neural Information Processing Systems*, 2015, pp. 1279–1287.

[13] B. Jiang, C. Ding, and B. Luo, "Covariate-correlated lasso for feature selection," in *Machine Learning and Knowledge Discovery in Databases.* Springer, 2014, pp. 595–606.

[14] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[15] K. Yu, X. Wu, W. Ding, and J. Pei, "Towards scalable and accurate online feature selection for big data," in *Data Mining (ICDM), 2014 IEEE International Conference on.* IEEE, 2014, pp. 660–669.

[16] J. Wang, P. Zhao, S. C. Hoi, and R. Jin, "Online feature selection and its applications," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 3, pp. 698–710, 2014.

[17] X. Wu, K. Yu, H. Wang, and W. Ding, "Online streaming feature selection," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 1159–1166.

[18] K. Yu, W. Ding, D. A. Simovici, and X. Wu, "Mining emerging patterns by streaming feature selection," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2012, pp. 60–68.

[19] Z. Xu, G. Huang, K. Q. Weinberger, and A. X. Zheng, "Gradient boosted feature selection," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2014, pp. 522–531.

[20] S. Xiang, T. Yang, and J. Ye, "Simultaneous feature and feature group selection through hard thresholding," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2014, pp. 532–541.

[21] A. Woznica, P. Nguyen, and A. Kalousis, "Model mining for robust feature selection," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2012, pp. 913–921.

[22] D. Cai, C. Zhang, and X. He, "Unsupervised feature selection for multi-cluster data," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2010, pp. 333–342.

[23] L. Xu, Q. Zhou, A. Huang, W. Ouyang, and E. Chen, "Feature selection with integrated relevance and redundancy optimization," in *Data Mining (ICDM), 2015 IEEE International Conference on.* IEEE, 2015, pp. 1063–1068.

[24] B. Cao, L. He, X. Kong, P. S. Yu, Z. Hao, and A. B. Ragin, "Tensor-based multi-view feature selection with applications to brain diseases," in *Data Mining (ICDM), 2014 IEEE International Conference on.* IEEE, 2014, pp. 40–49.

[25] H. Barkia, H. Elghazel, and A. Aussem, "Semi-supervised feature importance evaluation with ensemble learning," in *Data Mining (ICDM), 2011 IEEE 11th International Conference on.* IEEE, 2011, pp. 31–40.

[26] A. K. Farahat, A. Ghodsi, and M. S. Kamel, "An efficient greedy method for unsupervised feature selection," in *Data Mining (ICDM), 2011 IEEE 11th International Conference on.* IEEE, 2011, pp. 161–170.

[27] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 8595–8598.

[28] Y. Zhai, M. Tan, I. Tsang, and Y. S. Ong, "Discovering support and affiliated features from very high dimensions," *arXiv preprint arXiv:1206.6477*, 2012.

[29] Y. Jiang and J. Ren, "Eigenvalue sensitive feature selection," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 89–96.

[30] M. Masaeli, J. G. Dy, and G. M. Fung, "From transformation-based dimensionality reduction to feature selection," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 751–758.

[31] S. Paul and P. Drineas, "Deterministic feature selection for regularized least squares classification," in *Machine Learning and Knowledge Discovery in Databases.* Springer, 2014, pp. 533–548.

[32] S. Paul, M. Magdon-Ismail, and P. Drineas, "Feature selection for linear svm with provable guarantees," *arXiv preprint arXiv:1406.0167*, 2014.

[33] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th Annual Computer Security Applications Conference.* ACM, 2013, pp. 199–208.

[34] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing.* IEEE, 2013, pp. 3422–3426.

[35] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, 2015, pp. 1916–1920.

[36] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on.* IEEE, 2001, pp. 38–49.

[37] I. Santos, C. Laorden, and P. G. Bringas, "Collective classification for unknown malware detection," in *Security and Cryptography (SECRYPT), 2011 Proceedings of the International Conference on.* IEEE, 2011, pp. 251–256.

[38] M. Brückner, C. Kanzow, and T. Scheffer, "Static prediction games for adversarial learning problems," *Journal of Machine Learning Research*, vol. 13, no. Sep, pp. 2617–2654, 2012.

[39] M. Brückner and T. Scheffer, "Nash equilibria of static prediction games," in *Advances in Neural Information Processing Systems*, 2009, pp. 171–179.

[40] M. Brückner and T. Scheffer, "Stackelberg games for adversarial prediction problems," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2011, pp. 547–555.

[41] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, "Bagging classifiers for fighting poisoning attacks in adversarial classification tasks," in *International Workshop on Multiple Classifier Systems.* Springer, 2011, pp. 350–359.

[42] B. Biggio, G. Fumera, and F. Roli, “Adversarial pattern classification using multiple classifiers and randomisation,” in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR).* Springer, 2008, pp. 500–509.

[43] B. Biggio, G. Fumera, and F. Roli, “Multiple classifier systems for robust classifier design in adversarial environments,” *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 27–41, 2010.

[44] A. Barth, B. I. Rubinstein, M. Sundararajan, J. C. Mitchell, D. Song, and P. L. Bartlett, “A learning-based approach to reactive security,” in *International Conference on Financial Cryptography and Data Security.* Springer, 2010, pp. 192–206.

[45] W. W. Cohen, “Fast effective rule induction,” in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 115–123.

[46] Y. Mansour, “Learning boolean functions via the fourier transform,” in *Theoretical Advances in Neural Computation and Learning.* Springer, 1994, pp. 391–424.

[47] R. L. Rivest, “Learning decision lists,” *Machine Learning*, vol. 2, no. 3, pp. 229–246, 1987.

[48] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[49] D. Lowd and C. Meek, “Adversarial learning,” in *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining.* ACM, 2005, pp. 641–647.

[50] H. Yu, J. Han, and K. C.-C. Chang, “PEBL: positive example based learning for web page classification using svm,” in *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2002, pp. 239–248.

[51] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The Weka data mining software: an update,” *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.