Automated Generation and Selection of Interpretable Features for Enterprise Security

[1]Jiayi Duan*, [2]Ziheng Zeng†, [3]Alina Oprea‡,[4]Shobha Vasudevan§
[1,2,4]University of Illinois at Urbana-Champaign, Urbana, IL USA, [3]Northeastern University
Email: {*jduan4,†zzeng13, §shobhav}@illinois.edu,‡a.oprea@northeastern.com

***Abstract*—We present an effective machine learning method for malicious activity detection in enterprise security logs. Our method involves feature engineering, or generating new features by applying operators on features of the raw data. We generate DNF formulas from raw features, extract Boolean functions from them, and leverage Fourier analysis to generate new parity features and rank them based on their highest Fourier coefficients. We demonstrate on real enterprise data sets that the engineered features enhance the performance of a wide range 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. We also observe better isolation of malicious clusters, when performing clustering on engineered features. In general, a small number of engineered features achieve higher performance than raw data features according to our metrics of interest. Our feature engineering method also retains interpretability, an important consideration in cyber security applications.**

I. Introduction

In this paper, we introduce a feature engineering approach that generates a small number of features with high interpretability. Our features are generated with the intention of improving the performance of supervised classification algorithms, as well as unsupervised clustering algorithms. Our method is based on Fourier analysis of Boolean functions [1], [2]. We generate disjunctive normal form (DNF) formulas with respect to the raw features and extract Boolean functions from these formulas. We then apply Fourier analysis to these Boolean functions to generate new Boolean features and to select the most important features. While our method is generically applicable, we focus our approach toward detecting malicious activity in enterprise log data.

A. Enterprise security

Enterprises are targeted by increasingly sophisticated threats. To counteract these, organizations deploy a variety of security controls on their networks, including firewalls, anti-virus (AV) software, intrusion detection systems, and web proxies. These devices generate security logs, resulting in terabytes of log data collected daily by large organizations [3]–[6]. At the same time, large enterprises leverage teams of security analysts in Security Operations Centers (SOCs), whose role is to identify suspicious activities and complement existing security controls. Irrespective of whether alerts are generated by machine learning algorithms or other approaches, the role of human experts is irreplaceable and crucial to the security process. Security analysts manually investigate alerts, identify false positives, find the attack root cause, and remedy their effects in a time and resource intensive process. Through personal communication and systemic experimentation, we found out that security analysts highly value *easily interpretable features* when analyzing outputs of machine learning algorithms.

The value of classification or clustering methods in detecting malicious activity, then, is critically reliant on high quality features. Manually generated features are the de facto standard used by existing machine learning algorithms in security (e.g., [3]–[5], [7]). However, manual feature generation is focused on known attributes of the network. With increasing variability of malware attacks, security research has found it more efficient to extract features [7], and recently to generate new features [8] from the network traffic data. However, the engineered features in approaches like [8] are not human interpretable, limiting their deployment in industrial practice.

Security related data, particularly large scale log data, have two distinguishing characteristics. Firstly, the data tends to be highly imbalanced in practice, with a small number of positive (malicious) examples and a vast majority of benign examples [8]. This is due to the malicious traffic being a small fraction of overall network traffic, and even lesser being labeled as such. Secondly, precision/accuracy measurements are relatively easy to optimize for in this data, but malicious recall tends to be low. Most classifiers achieve high precision by a low number of false alerts. However, their detection is limited to the known samples in the training data, causing them to have low recall. Current methods prioritize high precision in a small number of detected malicious activities, while sacrificing recall [4], [5].

B. Our feature engineering solution

Our feature engineering method leverages Fourier analysis of Boolean functions [1], [2]. We generate disjunctive normal form (DNF) formulas for the raw features, extract Boolean functions from these formulas, and apply Fourier analysis to generate new parity features, as well as rank the most important ones.

We demonstrate experimentally that our methods improve malicious recall significantly for a broad range of classification and clustering algorithms. Our experiments are performed on two real datasets with labeled malicious activities collected from a large organization. These datasets are highly imbalanced and representative of real enterprise data. Due to the lack of publicly available enterprise log data, we focus on these real enterprise data sets, wherein we present in depth analyses.

We compare the performance (accuracy, recall and Area Under the Curve (AUC)) of various algorithms using our top few ranked engineered features with the performance of the same algorithms using the raw data features. Our experiments show that most classifiers and clustering algorithms have a low recall (as low as 20% and no greater than 70%) with raw data features. With feature engineering methods, malicious domain recall rises to as high as 92.85% while maintaining the accuracy at an acceptable level and improving AUC for some classifiers. We are able to achieve such high performance with much fewer engineered features compared to the number of raw data features (less than 15 most of the time). With clustering algorithms, we show that the clusters with raw data features are not "pure"; *i.e.*, not even a single cluster has a majority of malicious examples. In contrast, our engineered features generate up to 6 "pure" clusters. To the best of our knowledge, ours is the first feature engineering approach that improves the performance of both clustering and classification algorithms. We demonstrate through feedback provided by security experts that our top ranked engineered features have high explainability.

II. Related work

A. Feature engineering and feature transformation

Research in feature space has focused on (1) *feature selection*, or selecting the best set of features from the raw features; (2) *feature transformations*, or numerical transformations of raw features through weighted sums or matrix operations; and recently (3) *feature engineering*, or generating new features with a focus to improve classification performance.

Popular feature selection methods include convex function optimization [9]–[11], Lasso [12], ReliefF [13], FCBFK [14]–[16], and online feature selection [17], [18]. However, feature selection relies on the raw feature space, and is therefore limited by the quality of raw features. Feature transformation methods such as Principal Component Analysis (PCA) [19] and Independent Component Analysis [20] are very popular, but they typically do not preserve meaning of the combined features.

Recently, several automated feature engineering methods have shown promise in improving classification accuracy for different data sets. ExploitKit [21] proposes a framework for generating many candidate features using common operators (e.g., max, min, average, standard deviation, count). However, to find important features, it needs a ranking classifier that requires additional training data and is computationally expensive to train. AutoLearn [22] designs a regression method between feature pairs for feature generation and applies feature selection for identifying the top relevant features. Neither AutoLearn nor ExploitKit are designed to generate features that retain interpretability and are understandable by human experts.

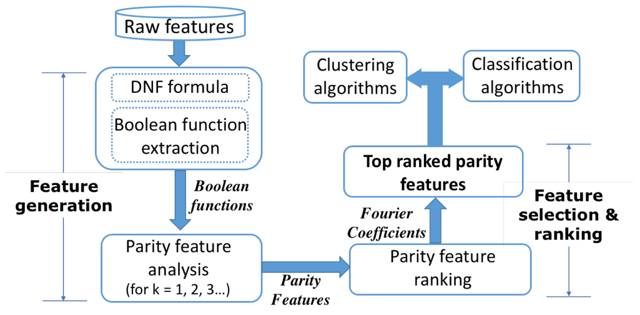

Raw features
DNF formula
Boolean function extraction
Feature generation
Boolean functions
Parity feature analysis (for k = 1, 2, 3...)
Parity Features
Parity feature ranking
Fourier Coefficients
Top ranked parity features
Feature selection & ranking
Clustering algorithms
Classification algorithms

Fig. 1: Our Fourier-analysis feature engineering approach

The definition of feature interpretability in some methods like Lasso [23] means quantifying the impact of a feature on the prediction outcome of the overall model. This is distinct from our definition of interpretability, which refers to semantically meaningful features, which are easily understandable by human experts.

B. Related security research

Improving enterprise security through machine learning techniques has been recognized as an interesting research direction recently. A number of papers propose using web proxy logs for detecting malware activities in enterprises, for example Beehive [3], ExecScent [24], WebWitness [7], BAYWATCH [5], and MADE [6]. All the above methods leverage manually defined features by domain experts, and thus retain the interpretability of the machine learning algorithms. However, most of them are optimized for high precision, and suffer from low recall of malicious activities.

In the context of security applications, several systems propose automated feature representations. Dahl et al. [37] use random projections of system calls and feed-forward neural networks for malware classification of a large malware corpus. Pascanu et al. [38] use recurrent neural networks for feature representation of malware system calls, and apply them to malware classification. Bartos et al. [8] design a system for malware detection that classifies legitimate and malicious enterprise flows using feature representations invariant to common malware behavior changes. These systems do not aim to preserve feature interpretability, and thus have limited value in practical deployment. To the best of our knowledge, we propose the first method for automated feature engineering for enterprise security that achieves the important goals of low dimensional feature space and interpretability. Our method is applicable to both classification and clustering algorithms and maximizes malicious recall.

III. Our method for feature engineering

Figure 1 shows the outline of our feature engineering approach. Each step is discussed in the following subsections.

A. Boolean feature generation

The first step is to convert numeric or nominal raw features into new Boolean features, in order to apply the Fourier transform. Unlike other feature transformation methods, our Boolean conversion technique does not convert all raw features into Boolean features, but identifies useful features for classification, and then converts only those features.

We use the RIPPER classifier [25] on raw data to generate classification rules first. The classification model generated by RIPPER is a formula in Disjunctive Normal Form (DNF). For example,

$$(A > 5) \wedge (B < 7) \wedge (D > 10) \rightarrow Class = TRUE$$
$$(A > 8) \wedge (C < 15) \wedge (E = \text{“}html\text{”}) \rightarrow Class = TRUE$$
$$All\ others \rightarrow Class = FALSE$$

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

Each conjunction in the DNF consists of several *Boolean features* such as $(A > 5)$ and $(B < 7)$, and any example is classified as TRUE or FALSE on such propositions. To ensure that the Boolean features are potentially useful, the accuracy of RIPPER's output should be relatively high. Since RIPPER is a rule-based classifier, it can reach 100% accuracy if not pruned (as it could lead to over-fitting). We select a small number of relevant features, and therefore our method is not affected by the large number of rules and features generated by RIPPER. We restrict pruning, such as the classification accuracy generated by RIPPER is higher than labeling all examples as "Benign". For this setting RIPPER successfully distinguishes the two classes and achieves 98% accuracy.

Other rule-based classifiers could be substituted for RIPPER in this step. Our motivation for using RIPPER is that the generated Boolean features from DNF rules are interpretable and have semantic meaning. In tree-based classifiers such as decision trees, every proposition is influenced by all the ancestors of a node in the tree. RIPPER is not constrained by previously generated rules and can thus generate a broader set of Boolean features.

B. Fourier analysis

1) Fourier transform and coefficients: Fourier Transform on Boolean variables [1] 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: $f : \{0,1\}^n \rightarrow \{-1, 1\}$. 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.

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:

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

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.

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

$$f(x) = \sum_{S \subseteq \{1, ..., n\}} a_S \chi_S(x) \tag{2}$$

The Fourier coefficient a_S for a given χ_S is computed across all training examples $x \in \{0,1\}^n$, as equation (3).

$$a_S = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \chi_S(x) f(x) \tag{3}$$

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 (2)). In this paper, we do not use a Fourier transform based classifier. We use this method only for computing coefficients as in (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 described in [1].

Equation (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) 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

$$\hat{a}_\beta = \frac{1}{m} \sum_{x \in T} \chi_\beta(x) f(x) \tag{4}$$

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 $a_S$ $\forall S \subseteq \{1, ..., n\}, |S| \leq k$
9: end function
```

2) Scaling the Fourier coefficient computation: There will be 2^n parity features in total since a set with n elements has 2^n subsets. To scale the computation of Fourier coefficients, it is important to identify and select a subset of Boolean features from 2^n possible subsets. A claim in [1] shows that, if $f(x)$ can be expressed as a decision list, 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.

We obtain, through RIPPER, disjunctive normal form (DNF) formulas on raw features with a high overall accuracy (over 98.5%) in Boolean conversion. Since DNF is a proper subset of decision lists, it is reasonable to consider only the small subsets in our task. We only need to focus on some small subsets with high $|a_S|$, since negating them will cause $2|a_S|$ difference on (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.

C. Selecting the top-ranked features using Fourier coefficients

We need to select the most important features among the numerous engineered parity features. We continue using Fourier coefficients for this purpose. Calculation of the coefficients of parity features is the same as in (4). Here, we do not know the function $f(x)$, but for each example x, there is a label- a value in $\{-1, 1\}$. 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.

We then pick only a few important features for evaluation. We use the absolute value of coefficients $|a_S|$ to rank the features in the descending order of $|a_S|$. After ranking, we select a few features with the largest $|a_S|$ values. We use up to 30 engineered features in each experiment, which is much smaller than the total number of features. Fourier-transform based classification needs more features to be effective. We therefore use other classification and clustering algorithms for our purpose. Clustering algorithms, in particular, perform better with fewer dimensions than higher dimensions, as shown in our experiments.

IV. Data sets

A. Enterprise log data sets

We obtained access to two data sets of enterprise logs collected over four months, containing information about connections between local machines and external web domains in a large enterprise. Information such as URL, domain name, sent and received bytes, web referrer, and content type is recorded for each connection. The data is pre-processed by extracting features for each external domain contacted by enterprise machines as in MADE [6]. Each example (domain) is labeled as malicious, benign, or unknown. The examples are labeled using Alexa ranking and VirusTotal score. For our purposes, all domains in Alexa top 100K are considered benign, while all domains with a VirusTotal score at least 3 are malicious (this is consistent with security methodology, e.g., [4], [6]). We only consider here benign and malicious domains, and ignore unknown domains from the dataset.

Data set 1 includes 242,074 domains with 63 features each. Among these, 45,928 examples are successfully labeled: 43,753 are benign (95.26%) and 2,175 are malicious (4.74%). **Data set 2** has 1,116,516 examples with 91 features each. Among these, 196,522 examples are successfully labeled: 191,355 are benign (97.37%) and 5,167 are malicious (2.63%). Our goal is to use classification algorithms to train a model that can predict benign or malicious domains in the future and prevent users from connecting to malicious ones. In our experiments, malicious domains are labeled positive, and benign ones are labeled negative.

B. Applying feature engineering to data sets

After Boolean conversion via RIPPER, we obtain 77 Boolean features for Data Set 1, and 98 Boolean features for Data set 2, for example (Max_Conn ≥ 9) or (Reg_Age ≤ 37).

We use Algorithm 1 for generating parity features. The number of parity features depends on k. When $k = 1$, we only look at original Boolean features in isolation. When $k = 4$, we look at subsets of 1, 2, 3, or 4 features. Using 98 raw features in Data Set 2, we obtain 98 parity features for $k = 1$, 4,851 parity features for $k = 2$, 3,769,277 parity features for $k = 4$, while $k = 6$ generates 1,124,298,483 parity features! The number of features for $k = 6$ is nearly 300 times higher than for $k = 4$, resulting in huge increase in computational and memory cost. Interestingly, the largest coefficients appear in subsets of 2, 3 or 4 features. Therefore, we use $k \in \{2, 3, 4\}$ in our experiments. When using $k = 4$ we start from $k = 1$ and add features one by one, and keep only subsets with $|a_S|$ higher than a threshold (set at 0.7) at each step, in order to keep the number of features manageable. Here are some examples of the parity subsets: {(Num_ASNs ≤ 0), (UA_Popularity ≥ 0.013514)} with coefficient of -0.7650, and {(Reg_Age $\leq$ 135), (ASN = UnknownIP), (Dom_Level ≤ 2), (Reg_Validity $\leq$ 1826)} with coefficient of -0.8634.

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 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.

We use three metrics to evaluate the results: the overall accuracy $\frac{TP+TN}{All}$, recall of positive (malicious) class $\frac{TP}{TP+FN}$, and Area Under the Receiver Operating Characteristic Curve (AUC). Malicious recall is more important than overall accuracy, but it is very low in our data set when using raw features for classification (between 28% and 70%). Interestingly, both accuracy and AUC can easily reach 98% using raw features. We can therefore tolerate a decrease in accuracy and AUC if it does not lead to many false positives, provided that we obtain a substantial increase in recall.

V. Experiments

A. Experimental setup

For classification, we use six algorithms (SVM, AdaBoost, RIPPER, decision tree (DT), random forest (RF), and PEBL [26]) to train models, and evaluate them according to our metrics using 3-fold cross validation. These algorithms are widely popular classifiers based on different core techniques and optimization. We use all of them to test if our new features can be used for different types of classifiers. We compare the accuracy and malicious recall of our engineered features versus raw features for the same classifier. For SVM, AdaBoost, RIPPER, DT, and RF, we use the default parameters of Weka 3.7.12 [27] to build the classifiers. Note that the same set of default parameters is used for both our method and the baseline methods, so the comparison is fair, even without performing in-depth parameter tuning and optimization.

We implement PEBL according to [26]. PEBL is a semi-supervised method that needs to be initialized with "strong negative" examples. We use the top-ranked parity features to select these negative examples. For example, assume that we use top 5 features for classification. 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, *i.e.*, $\chi_{F_j} = -1$ if $a_{F_j} > 0$ or $\chi_{F_j} = 1$ if $a_{F_j} < 0$. For strong negative examples, we pick those for which all 5 features are more likely to make the label negative. However, to identify "strong negative" examples in raw data, a classifier needs to be trained to identify examples from the negative class. Hence, PEBL does not work with raw features, and PEBL results are presented only on parity features.

For clustering, we use Expectation Maximization (EM) and the k-means algorithm. We are the first to consider the benefit of engineered features in unsupervised learning methods. Clustering is extremely useful in security applications to identify machines infected by the same malware, or web domains with similar network activity. Here our metric of interest is different. We look for "pure clusters", defined as clusters consisting of a majority of malicious domains. A clustering algorithm that generates more pure clusters is better at distinguishing malicious domains from the legitimate ones.

B. Performance of propositional (Boolean) features

In this experiment, we considered the top 1-20 ranked individual Boolean features (equivalent to $k = 1$). Figures 2a and 2b show the accuracy and malicious recall of the 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. SVM has similar malicious recall as DT for engineered data, but 40% lower recall for raw features.

We find that the accuracy of SVM, RIPPER, DT, and RF are similar for engineered features compared to raw features. We find that a small number of engineered features can get higher recall than using 20 raw features for all 5 classifiers other than PEBL. PEBL is an outlier that gets much higher recall and much lower accuracy compared to the other 5 classifiers. It discovers nearly all the positive examples, but also leads to false positives.

We conclude that we can get higher recall (around 20% higher recall of malicious domains on average) and nearly same accuracy when we use far fewer engineered features (between 1 and 3) than raw features (91).

TABLE I: Comparison of AUC (malicious) using 5 classifiers with top 5 parity features and all raw features from Data Set 2. Values with asterisks are the best ones among that column. Proposed method wins in 3 out of 5 classifiers.

AUC (Malicious)	SVM	AdaBoost	RIPPER	DT	RF
Parity Features	0.885*	0.884	0.884*	0.884*	0.884
Raw Features	0.729	0.976*	0.863	0.867	0.984*

C. Performance of parity features

In this experiment, we consider parity features ($k > 1$) generated by the Fourier expansion method. The k values are set through empirical validation to provide the best performances on our data sets. The optimal value of k for Data Set 1 was $k = 2$ and for Data Set 2 was $k = 4$. The top 5-30 engineered features were used for classification.

Figures 3a and 3b show the accuracy and malicious recall of the 6 classifiers using top 5-30 parity features ($k = 4$) on Data Set 2. The recall achieved with 5 parity features is already higher than the recall with all raw features. The same observations hold for the similar experiment done on Data Set 1. Although the accuracy is slightly lower, it is still acceptable for our application.

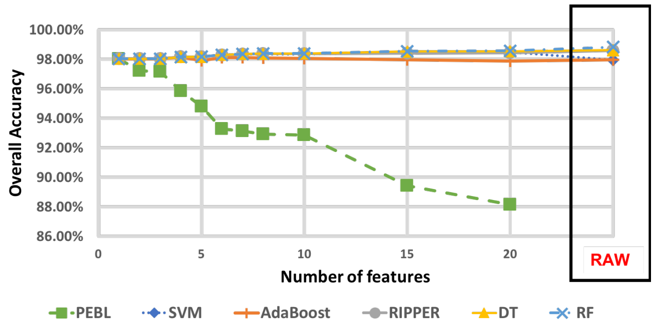
Figure 4 shows the extent of improvement in recall, as well as the corresponding change in accuracy for parity features compared to raw features on both Data Set 1 and Data Set 2. It is clear that engineered features achieve a significant improvement in recall while not sacrificing too much accuracy. Among all the considered classification models, the engineered features work best with SVM. We conjecture this to be because of SVM's kernel calculation that benefits from the representation of Boolean and parity features.

Table I shows the AUC of 5 classifiers using top 5 parity features and raw features on Data Set 2. AUC is improved in three out of five classifiers. Although AUC is not consistently improved with parity features for all classifiers, the recall improvement of using the parity features is always much more significant than the drop in AUC.

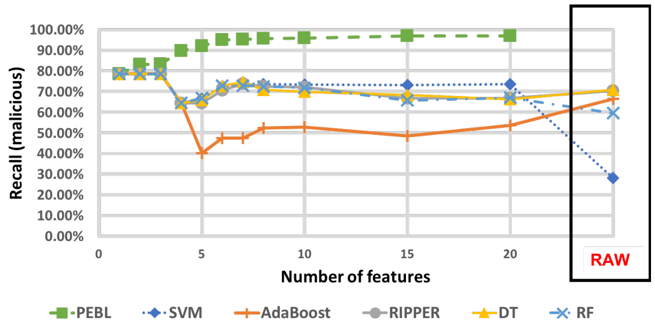
We conclude that we achieve higher recall, similar accuracy, acceptable if not higher AUC with far fewer engineered features than raw features for both data sets.

D. 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,

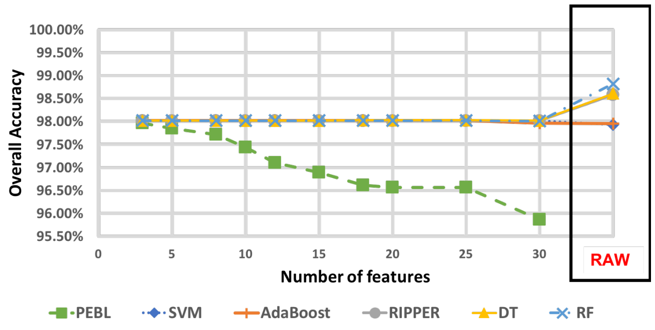


(a) Overall accuracy

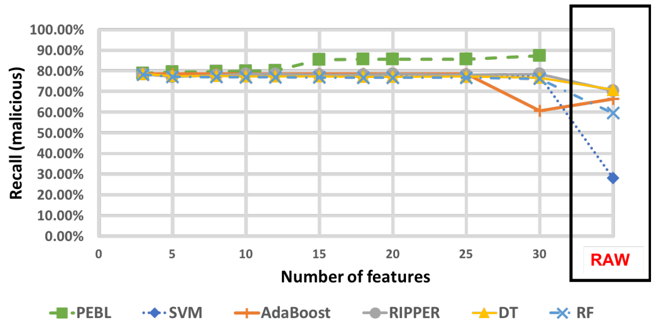


(b) Recall of malicious domains

Fig. 2: Accuracy and malicious recall of 6 classifiers of our top 1-20 propositional features compared with raw features on Data Set 2. PEBL has 96.96% recall and 88.15% accuracy, while other methods have up to 78.61% recall and 97.9%-98.6% accuracy.



(a) Overall accuracy



(b) Recall of malicious domains

Fig. 3: Accuracy and malicious recall of 6 classifiers using the top 5-30 parity features on Data Set 2. AdaBoost, RIPPER, and RF are very close in accuracy to DT for engineered features, and their accuracy for raw features ranges from 96.65% to 98.56%. Recall of AdaBoost, RIPPER and RF are nearly the same as DT for engineered features, and 69.70%, 39.2% and 59.5% for raw features. PEBL gets up to 87.25% recall with 95.87% accuracy, while others get 60.8%-78.6% recall.

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 the EM algorithm to find the proper number of clusters, then used k-means algorithm with fixed number of clusters. Figure 5 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 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.

A similar experiment was done on Data Set 2: 91 raw features compared to top 20 parity features (limited $k = 4$). The number of "pure" positive clusters increased from 0 to 1, and 95.07% of negative examples are grouped in a large "pure" cluster, while the fractions of positive examples are 2.99% to 19.17% in clusters with raw features.

We conclude the engineered features contribute to "pure" positive clusters (1-6 more than using raw features) and can thus isolate malicious domains better than raw features.

E. Processing times for raw and engineered features

The time taken by the considered 5 classifiers using raw features is compared to the time taken by 30 parity features on Data Set 2. For raw features, the reported timing is just the time taken for classification. For parity 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 using parity features of $k = 4$ can take up to 6.5 times longer than $k = 1$ and up to 18 times longer than raw feature processing. We saw in sections V-B and V-C that even with $k = 1$, it is possible to get a significant improvement in recall over raw features. As we discussed in section III-B2, the number of features is $\binom{n}{1} + \binom{n}{2} + ... + \binom{n}{k} \sim n^k$ for $k \ll n$. If the application is

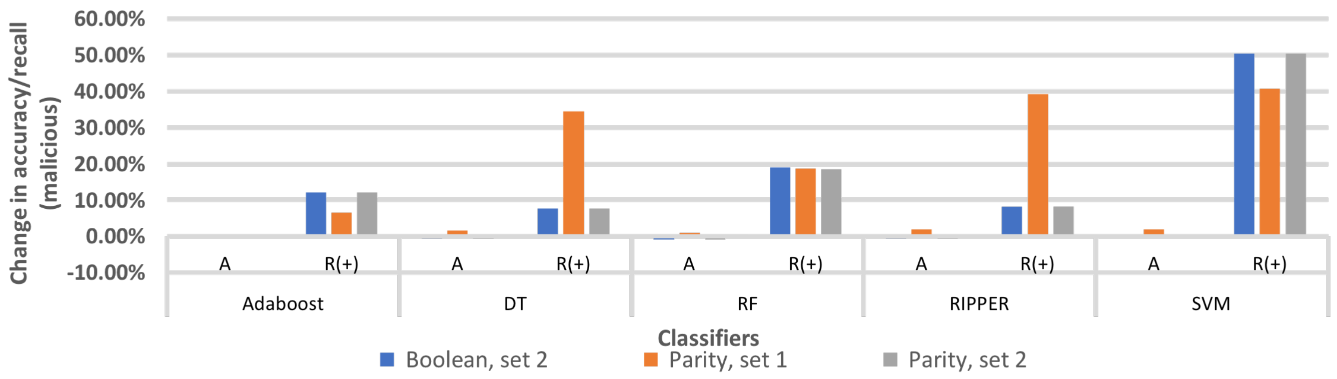


Fig. 4: Change in accuracy (A) and recall of malicious domains (R(+)) compared with raw features, when getting highest recall with engineered features. Improvement in recall is much more significant than small accuracy change.

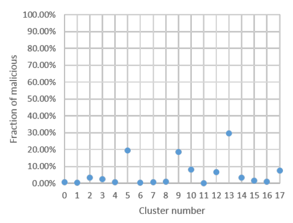


Fig. 5: Raw features

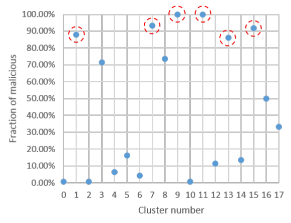


Fig. 6: Engineered features

Fig. 7: 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.

performance-sensitive, $k = 1$ is recommended over $k = 4$ for higher recall and faster computation.

F. Interpretability of parity features

Table II shows our top-ranked features and a security domain expert interpretation of how they are related with identifying malicious domains. Some parity features have more than one proposition and they should be interpreted by using AND among propositions. AND here means that all propositions within the same parity feature should be satisfied together. The domain expert confirmed that the generated features are aligned with typical attack indicators. This experiment indicates that our feature engineering method has the potential to generate top-ranked features that are meaningful to domain experts. This is the advantage of our technique in comparison to existing feature selection and transformation approaches.

G. Comparison with other feature selection methods

Finally, we compare our Fourier-analysis feature engineering method with five popular feature selection methods: Lasso, RFI, FCBFK, ReliefF, and CIFE. The Fourier method selects the top ranked parity features, while other feature selection methods select from a larger set of parity features.

Specifically, we generate all parity features for $k = 4$ on Data Set 2. Since the number of parity features becomes large, we select the top 1000 parity features ranked by Fourier coefficients and pass them to the competing feature selection methods. We run existing feature selection methods to select 5 and 30 features given these 1000 features as input. We compare these with the top 5 and 30 selected by highest Fourier coefficients and use SVM, decision trees, and AdaBoost as classifiers.

The results are shown in Table III. We observe that top parity features selected by our method outperform existing feature selection methods in 5 out of 6 experiments. We find that the difference in recall between the top 5 and 30 features is very large in some cases for competing feature selection methods, sometimes reaching as high as 64.6% for ReliefF and AdaBoost. The reason might be that these methods do not compute feature importance consistently. In this way, top features may not be truly important, whereas a lower-ranked feature may drastically improve the recall. This is unsuitable for building a robust classifier. We conclude that our method obtains a maximum of 79% and an average of 29% higher recall of malicious domains compared to other feature selection methods.

VI. Conclusion

In conclusion, we propose a method based on the RIPPER classifier and Fourier coefficients for automated feature engineering in security applications. We show that the feature engineering approach benefits a wide range of classification and clustering algorithms used for enterprise malware detection. It produces higher malicious recall with respect to raw features and existing feature selection approaches, while reducing the dimensionality of the problem significantly. We showed that a small number of engineered features are able to produce high recall of malicious domains in the enterprise log data, while retaining interpretability by human experts.

In future work, we plan to extend and apply our feature engineering method to other security applications, for instance malware detection from system call data. We are also interested in adapting our methods to other application areas with different constraints, metrics of interest, and optimization objectives.

TABLE II: Interpretability of top-ranked features.

Features	Interpretations by Security Analysts
Dom_Sub ≤ 1	Most legitimate domains have many sub-domains on second-level, but malicious domains have few of them.
$\{$(ASN=UnknownIP), (Dom_Level ≤ 2), (Reg_Validity ≤ 1826), (Reg_Age ≤ 135)$\}$	Cannot resolve domain at the time query was done. Possibly the domain is inactive or was taken down; Domain has 2 levels, while most legitimate domains will have more than 2 levels; Domain has a registration validity less then 1826 days, while by default domains are registered for at least 3 years; Domain is relatively recently registered (4 months ago) while most legitimate domains have a longer registration age.
$\{$(UA_Popularity ≥ 0.01), (Num_ASNs ≤ 0)$\}$	User agent string is used by at least 1% of the machine population (it has popularity above 1%); Cannot resolve domain at the time query was done since possibly the domain is inactive or was taken down;
Avg_URL_Length ≥ 13	Long URLs are more suspicious since they might be used for malware communication to a command-and-control center.
Avg_ratio_rbytes ≥ 103.63	The ratio of bytes sent over bytes received. For legitimate web sites, usually more content is received than sent by end hosts.
Update_Validity ≤ 365	Number of days from update till expiration. Malicious domains have lower expiration dates than legitimate ones.

TABLE III: Comparison of malicious recall using 3 classifiers with top 5 and 30 parity features ranked by Fourier method, Lasso, random forest, CIFE, ReliefF, and FCBFK importance. **RME**: Recall of malicious examples, **DT**: Decision tree, **AB**: AdaBoost, **PF**: Parity features. Results should be compared in columns. Bold values indicate results of our proposed algorithm, and values with asterisks are the best in that column. Fourier method wins in 5 out of 6 experiments.

RME (%)		**SVM**		**DT**		**AB**	
		T5	**T30**	**T5**	**T30**	**T5**	**T30**
PF	**Fourier**	**78.6***	**78.6***	**78.6***	**77.6***	**78.6***	**60.8**
	RFI	0.00	23.4	11.3	69.9	0.00	37.3
	Lasso	78.6	66.6	42.9	71.3	65.7	58.3
	CIFE	78.6	74.1	56.4	64.6	69.3	52.1
	ReliefF	32.7	65.7	32.7	67.4	0.00	64.6
	FCBFK	78.6	61.1	78.6	65.0	41.6	53.9

Acknowledgements

We thank the enterprise that provided access to the security logs for analysis. We are grateful to the entire EMC Critical Incident Response Center (CIRC) for supporting this research. This work has been partially funded by the NSF grant CNS-1717634 and Boeing Corporation.

References

[1] Y. Mansour, "Learning boolean functions via the Fourier transform," in *Theoretical Advances in Neural Computation and Learning*. Springer, 1994, pp. 391–424.

[2] 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.

[3] 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 *Proc. 29th Annual Computer Security Applications Conference*, ser. ACSAC '13, 2013.

[4] A. Oprea, Z. Li, T.-F. Yen, S. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2015.

[5] X. Hu, J. Jang, M. P. Stoecklin, T. Wang, D. L. Schales, D. Kirat, and J. R. Rao, "BAYWATCH: robust beaconing detection to identify infected hosts in large-scale enterprise networks," in *DSN*. IEEE Computer Society, 2016, pp. 479–490.

[6] A. Oprea, Z. Li, K. Bowers, and R. Norris, "MADE: Security Analytics for Enterprise Threat Detection," in *Proc. 34th Annual Computer Security Applications Conference*, ser. ACSAC '18, 2018.

[7] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad, "Webwitness: Investigating, categorizing, and mitigating malware download paths." in *USENIX Security Symposium*, 2015, pp. 1025–1040.

[8] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016, pp. 807–822.

[9] 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.

[10] 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.

[11] 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.

[12] R. Tibshirani, "Regression shrinkage and selection via the Lasso: a retrospective," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 3, pp. 273–282, 2011.

[13] I. Kononenko, "On biases in estimating multi-valued attributes," in *Ijcai*, vol. 95. Citeseer, 1995, pp. 1034–1040.

[14] B. Senliol, G. Gulgezen, L. Yu, and Z. Cataltepe, "Fast correlation based filter (fcbf) with a different search strategy," in *Computer and Information Sciences, 2008. ISCIS'08. 23rd International Symposium on*. IEEE, 2008, pp. 1–4.

[15] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 856–863.

[16] D. Lin and X. Tang, "Conditional infomax learning: an integrated framework for feature extraction and fusion," in *European Conference on Computer Vision*. Springer, 2006, pp. 68–82.

[17] 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.

[18] 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.

[19] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[20] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4, pp. 411–430, 2000.

[21] G. Katz, E. C. R. Shin, and D. Song, "ExploreKit: Automatic feature generation and selection." in *ICDM*, F. Bonchi, J. Domingo-Ferrer, R. A. Baeza-Yates, Z.-H. Zhou, and X. Wu, Eds. IEEE, 2016, pp. 979–984.

[22] A. Kaul, S. Maheshwary, and V. Pudi, "AutoLearn: Automated feature generation and selection," in *2017 IEEE International Conference on Data Mining (ICDM)*, Nov 2017, pp. 217–226.

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

[24] T. Nelms, R. Perdisci, and M. Ahamad, "ExecScent: Mining for new C&C domains in live networks with adaptive control protocol templates," in *Proc. 22nd USENIX Security Symposium*, 2013.

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

[26] 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.

[27] 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.