Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Discovering features for detecting malicious websites: An empirical study

John McGahagan IV, Darshan Bhansali, Ciro Pinto-Coelho, Michel Cukier*

Center for Risk and Reliability, University of Maryland, College Park, MD, USA

ARTICLE INFO

Article history:
Received 13 November 2020
Revised 27 May 2021
Accepted 9 June 2021
Available online 16 June 2021

Keywords:
Feature-based malicious website detection
Web security
Supervised learning
Feature selection

ABSTRACT

Website features and characteristics have shown the ability to detect various web threats – phishing, drive-by downloads, and command and control (C2). Prior research has thoroughly explored the practice of choosing features ahead of time (*a priori*) and building detection models. However, there is an opportunity to investigate new techniques and features for detection. We perform a comprehensive evaluation of *discover*ing features for malicious website detection versus selecting features *a priori*. We gather 46,580 features derived from a `response` to a web `request` and, through a series of feature selection techniques, discover features for detection and compare their performance to those used in prior research. We build several detection models using unsupervised and supervised learning algorithms over various sampling and feature transformation scenarios. Our approach is evaluated on a diverse dataset composed of common threats on the internet. Overall, we find that *discovered* features can achieve more efficient and comparable detection performance to *a priori* features with 66% fewer features and can achieve a Matthews Correlation Coefficient (MCC) of up to 0.9008.

1. Introduction

Security and incident response teams are constantly inundated with cyber threats. Many of the most devastating (ransomware, disruption of communication systems, *etc.*) are enabled by malicious websites. Detecting and preventing communication with these websites could prevent many attacks. One method of detection is to extract features from websites and web traffic and build detection models from learning algorithms. This practice has demonstrated success in prior research. However, over the past 10 years, there have only been slight changes to the features used for detection. This consistency, in the face of a constantly evolving threat, presents an opportunity to re-evaluate their utility and determine if other approaches can outperform selecting features *a priori* (choosing features that have demonstrated success in prior research or are motivated by specific examples of threats). A re-evaluation of the features and methods for detection may lead to better detection (higher accuracy, lower false positive rates, *etc.*) or more efficient (less computational burden or instrumentation overhead) detection capabilities for academic studies and security operations center implementations.

In this paper, we present a comprehensive evaluation of the discovery of features for malicious website detection instead of *a priori* selection of features. We do so by collecting features from a `response` to a web `request`. Our dataset consists of benign websites from the Alexa Top Domains (Amazon.com 2018) provided by (Durumeric et al., 2015). The malicious websites consist of phishing webpages, drive-by downloads, and other malicious websites including command and control (C2) URLs provided by the Cisco Talos Intelligence

* Corresponding author.
E-mail addresses: jmcgahag@umd.edu (J. McGahagan IV), darshanb@umd.edu (D. Bhansali), clpc@umd.edu (C. Pinto-Coelho), mcukier@umd.edu (M. Cukier).
https://doi.org/10.1016/j.cose.2021.102374

Group (Cisco Talos Intelligence Group 2021). We apply a series of feature selection techniques to discover features suitable for detection of malicious websites. We investigate their detection performance using unsupervised and supervised learning algorithms in various sampling and feature transformation scenarios. We compare the detection performance of *discovered* features to those selected *a priori*. Overall, we find:

1 The discovery approach produces features that yield similar performance to features selected a priori, but requires fewer features;
2 The discovery approach identifies features that produce higher classification metrics when using numerical feature transformation and principal component analysis with supervised learning further demonstrating its benefits over selecting features *a priori*; and
3 The discovery approach identifies features used in prior research, as well as new features and feature combinations.

We make the following contributions:

We demonstrate the potential of discovering features for malicious website detection by achieving a Matthews Correlation Coefficient (MCC) of 0.9008 without parameter tuning and 0.9003 with parameter tuning.

We show that new features need to be discovered and evaluated for their ability to detect malicious websites by showing that supervised models built from *discovered* features outperformed the supervised models built with features selected *a priori* by an average MCC of 0.004 with 66% fewer features without parameter tuning and an average MCC of 0.0024 with parameter tuning.

To our knowledge this is the first work that 1) leverages three aspects of web `response` features (URL features, webpage content features, and HTTP `response` headers) as the sole means for detection, 2) leverages feature selection as the primary means for identifying features for malicious website detection, 3) compares the ability of *discovered* features to those from prior research, and 4) re-evaluates features from prior research to evaluate their relevance.

This paper is structured in the following manner. Section 2 describes related work on malicious website detection using *a priori* features. Section 3 defines our research questions. Section 4 describes our analytical methods. Section 5 presents our results, and Section 6 contextualizes them within our research questions and other work. Sections 7 and 8 describe the limitations of this study and the conclusion, respectively.

2. Related work

Machine learning has been applied in many cybersecurity studies and has shown potential to detect various threats and malicious websites. Three threats are commonly detected in prior research – phishing webpages, drive-by downloads, and C2 infrastructure.

Ma (Ma et al., 2009) exhibited one of the earliest approaches to detect phishing webpages and spam. They used the URL structure and host-based properties gathered from other sources (IP denylists, WHOIS, domain name properties, and geographic properties) with Naïve Bayes (Murphy, 2006) and Logistic Regression (McCullagh and Nelder, 1989) classifiers. In (Ma et al., 2009)-(Ma et al., 2011), the authors used similar features and added the use of Online Learning (Bottou and LeCun, 2004) to detect malicious URLs. CANTINA (Zhang et al., 2007) relied primarily on features from the webpage content and created heuristics, and was evaluated based upon 100 legitimate URLs from a prior study (3sharp, 2019) and 100 URLs from PhishTank (Open DNS, 2019). CANTINA+ (Guang et al., 2011) improved upon this approach by adding features and training and testing models built from various machine learning methods. The authors relied on subject matter expertise to select features and utilized search engine features – features derived from search results. Whittaker (Whittaker et al., 2010) utilized a Logistic Regression classifier and used millions of URLs for evaluation. Marchal (Marchal et al., 2016) took a similar approach as CANTINA+ and gathered 212 features, differentiated between languages on the webpage, and analyzed the URL structure. Their method used a Gradient Boosting (Friedman, 2002) algorithm and 1,213 malicious and 5,000 benign samples in their training, and 1,553 entries in their testing set. Phishmon (Niakanlahiji and Al-Shaer, 2018) leveraged lengths of HTTP headers as features for their phishing detection mechanism and Li (Li et al., 2020) used feature transformation to perform better phishing detection.

Drive-by downloads (malicious code execution) can be detected with Machine Learning. JSAND (Cova et al., 2010) identified malicious JavaScript with 10 features associated with drive-by downloads. Their approach relied on instrumenting a browser, executing the code, and recording the events. Cujo (Rieck et al., 2010) performed static and dynamic analysis of JavaScript. The static analysis relied on lexical tokens and the dynamic analysis relied on known attack patterns. The sequences from static and dynamic analysis were transformed into Q-grams - a sequence of 'q' words within the code execution - which were then used to train a Support Vector Machine classifier. Zozzle (Curtsinger et al., 2011) performed static analysis by first de-obfuscating the JavaScript and creating features from two parts – the *context* in which it appears (try/catch block, *etc.*), and some text. They used *contexts* relevant to malicious JavaScript detection. Features were selected and used to train a Naïve Bayesian classifier on 919 malicious entries and 7,976 benign samples and achieved a false positive rate of 0.0003% on 1.2 million files. Revolver (Kapravelos et al., 2013) examined the AST (abstract syntax tree) created from the JavaScript, created sequences of nodes, and compared the similarity to known malicious sequences. Prophiler (Canali et al., 2010) detected drive-by downloads with features commonly used in phishing detection (webpage content). They trained their model on 787 samples of drive-by downloads with HTML elements, static JavaScript features, URL features, and features from DNS. Arrow (Zhang et al., 2011) detected drive-by downloads with HTTP traces pulled from logs instead of from the JavaScript. JavaScript analysis is often the source of features in studies to detect drive-by downloads however other features commonly associated with phishing detection have also been used.

The last threat commonly detected in prior research is C2 (bot) URLs. Authors in (Perdisci et al., 2013; Perdisci et al., 2010) performed clustering on high-level features (total number of HTTP `requests`, number of GET/POST `requests`, `response`

lengths, *etc.*) and lower-level features (specific headers) and created HTTP traffic clusters to derive signatures. ExecScent (Nelms et al., 2013) focused on clustering `requests` and built templates for detection from HTTP traffic clusters and derived signatures from the URL path length, query component, user-agent string as well as other headers. These features and similar features have been used in other studies as well as in (Kheir et al., 2015), which built clusters of HTTP `requests` by extracting the URL path, URL parameters, and URL method (GET, POST, *etc.*). Zarras (Zarras et al., 2014) used header chains (sequences of HTTP headers) for their detection method to create signatures from clustering. (Beigi et al., 2014) also had the goal of detecting bots and gathered 'flow-based' features (extracted from network traffic) over a period to do so. Yadav, in (Yadav et al., 2010), aimed to detect domains generated by domain generation algorithms (DGA) (MalwareBytes Labs, 2016) by examining features purely from domains names. (Yan et al., 2020) leveraged four URL features – length, percentage of vowels, percentage of digits, and URL suffix and evaluated five different classifiers.

Although studies aimed to detect differing threats, some have been able to detect multiple threats with various types of features. For example, (Eshete et al., 2012) utilized features to detect phishing webpages and drive-by downloads, and (Xu et al., 2013) selected various feature types – webpage content, flow-based features, URL features, *etc.* and leveraged (The Honeypot Project, 2009) as their dataset. Additionally, feature selection and transformation are becoming more common in threat detection studies. Authors in (McGahagan IV et al., 2019) and (McGahagan et al., 2019) leveraged feature selection and evaluated the effectiveness of new HTTP header and webpage content features and showed that new features can be incorporated to improve detection. (Zhou et al., 2020) leveraged correlation-based feature selection in building an ensemble classifier for detection. (Li et al., 2020) achieved an improvement in accuracy from 68% to 86%, 58% to 81%, and 63% to 82% over three classifiers when incorporating feature transformation into malicious URL detection.

3. Research Questions

Prior studies have relied on preconceived notions of relevant (*a priori*) features – URL length, `<iframe>`s, *etc.*, for detection. This reliance has demonstrated success however one of the main challenges for security teams, who often rely on these features, is the number of false positives (Bloomberg, 2019). Attacks change over time (Willard, 2015), as do the technologies used by attackers and developers of benign websites. Hence, there is a need to re-evaluate the features used to detect malicious websites. For example, HTML5 (W3School 2020), released in 2014, introduces new tags (elements). Thus, features that were new to HTML5 could not have been included in research prior to 2014. Additionally, there are techniques such as feature selection (Featuretools 2019), that have been shown to improve classification (M et al., 2012), that can be employed to discover features which may be more applicable to the detection of malicious websites.

Research Question 1 – Is feature *discovery* feasible for malicious website detection?

Even if feature discovery is feasible, there is no guarantee that it is better than selecting features *a priori*. To date, there is little insight into how *discovered* features' detection ability compares to those from prior research and if features from prior research are or are not still relevant for detection.

Research Question 2 – How do *discovered* features' detection ability compare to those from prior research?

Once we've established the feasibility of *discovered* features and compared their detection ability to the detection ability of *a priori* features from prior research, we then investigate the *discovered* features along with operational constraints. A constraint within an operational scenario is that a network is exposed to various threats simultaneously and leverages denylists (among other tools) to prevent communication with the malicious website. Several studies focus on identifying specific threats - (Marchal et al., 2016) focused on phishing, (Rieck et al., 2010) studied to drive-by downloads, and (Perdisci et al., 2013; Perdisci et al., 2010) focused on detecting C2 infrastructure and have demonstrated success doing so. Studies focusing on a single type of threat are often the most successful approaches from research. Marchal (Marchal et al., 2016), who achieved an AUC of 0.999 on English phishing webpages, focused only phishing webpages. This paradigm is useful for understanding the nature of specific threats. However, an operational scenario may not be able to optimize their detection method to detect a specific threat type. These observations motivate the exploration of other methods (and features) that can detect several types of threats. Finding features (or a method of finding features) for detecting several types of threats is a potential solution to this challenge.

Success in an operational scenario also depends on the availability and choice of features used - certain features are often associated with detecting certain types of threats. For example, phishing is typically best detected from HTML on a webpage, drive-by downloads are often detected by the JavaScript, and C2 infrastructure is often detected by the URL structure or HTTP headers. When a user (or service) retrieves a malicious website, there may not be information about the type of threat, the malicious website, or the relevant features for detecting that website. Furthermore, a website could be a phishing webpage, result in a drive-by download, and serve as C2 infrastructure. Additionally, some features that are useful for detection may not be available in a timely manner. DNS information, search engine results, WHOIS features have all shown promise for detection, but we cannot guarantee that these features are available when determining whether to block communication with a website. Web `response` features, that is, those that can be gathered from a `response` to a web `request` have characteristics that are advantageous for this challenge – 1) they provide features that can detect phishing, drive-by downloads, and C2 websites; 2) they can be gathered in a mechanism similar to a web browser; 3) the features can be extracted without executing or rendering the webpages (which could prevent drive-by downloads); and 4) they can be gathered without relying on external sources like DNS servers or search engines. This study leverages features derived from the `response`, which are derived directly from the website.

Research Question 3 – Can a *discovery* approach be applied to several threats when only features from a web `response` are available?

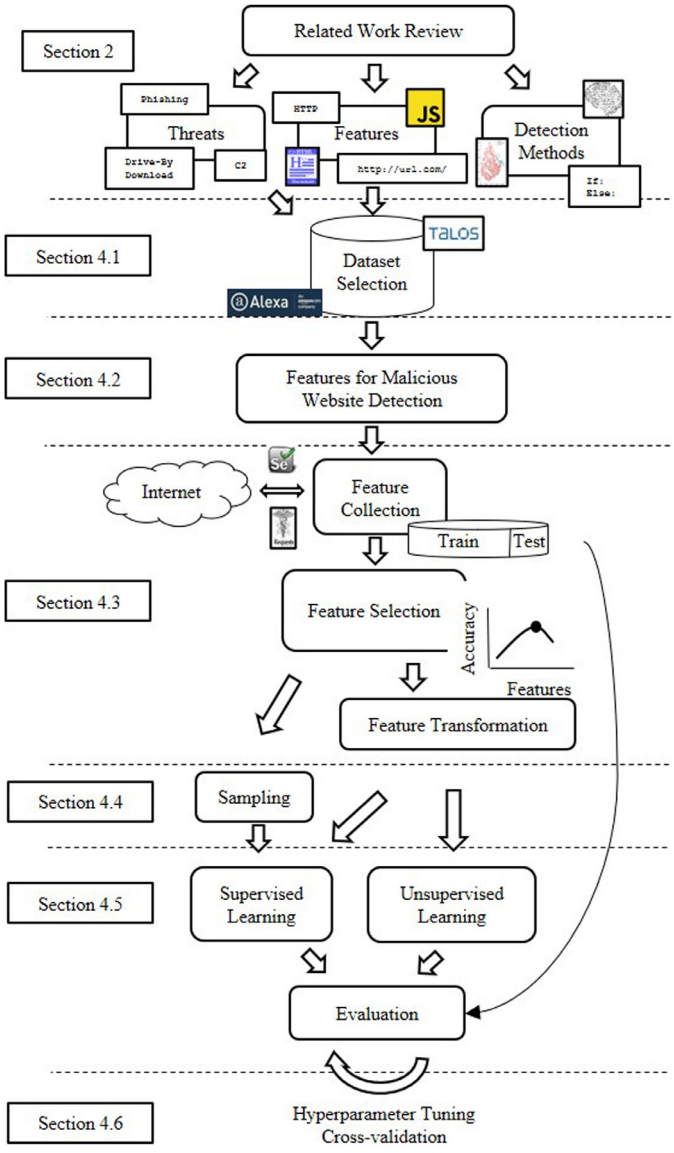

Fig. 1 – High-Level Approach. Images were provided by Pixabay (Pixabay 2019) or were created

4. Methods

Fig. 1 provides an overview of the seven-step analytical process we used in this research.

1 *Related Work Review* – gathering information to understand threats detected in prior research, features capable of detection, and the methods used for detection;
2 *Dataset Selection* – considering potential dataset sources and size, and procuring a dataset;
3 *Features for Malicious Website Detection* – using operational constraints to determine features to include in our base to discover features;
4 *Feature Collection, Selection, and Transformation* – collecting features from websites in our dataset, using feature selection to discover features to detect malicious websites, and using feature transformation to create additional features to detect malicious websites;
5 *Sampling* – creating three sampling scenarios to address the imbalance of our dataset;
6 *Unsupervised and Supervised Learning* – training unsupervised and supervised learning algorithms on *discovered* features, *a priori* features, and features created via feature transformation; and
7 *Hyperparameter Tuning and Cross-validation* – performing hyperparameter tuning and cross-validation to attempt to improve our models.

The following sections describe our analytical process in detail.

4.1. *Dataset selection*

Our dataset consists of two portions – benign entries from the 34,742 Alexa Top Domains, and 4,441 malicious entries provided by Cisco Talos Intelligence Group (Cisco Talos In-

telligence Group 2021). The Alexa Top Domains have been used in several studies including (He et al., 2011). The malicious entries encompass several threat types (phishing, drive-by download, and C2). Both datasets are provided by external organizations and are not hand-selected or created for the purpose of this study.

To select a size for the benign dataset, we first survey prior research. We chose a benign dataset of 40,000 websites (although only 34,742 could be collected as discussed in following paragraphs). However, this number is subjective. Small and large datasets have advantages – smaller datasets allow for focus and deeper analysis of a few samples, while larger datasets may potentially be more representative. The Cisco Talos Intelligence dataset consisted of 7,039 websites of which we were able to collect 4,441 websites. We used sampling techniques to address the imbalance in our datasets.

4.2. *Features For Malicious Website Detection*

We derive 46,580 features from the `response` to an HTTP GET `request` – lexical URL features, webpage content (the HTML and JavaScript on the page), and HTTP headers in the `response`. Leveraging all three of these facets has several advantages. First, we may increase the ability to detect multiple types of threats (URL features are commonly used to detect C2 and phishing websites, JavaScript on the webpage is used to detect drive-by downloads, *etc.*). Second, considering different features sources may capture additional (and potentially unknown) aspects of the threat. Third, these features can be gathered immediately after the website is accessed and do not require rendering the webpage or recording features over a period (which risks allowing malicious activity to proceed).

URL Features – We extract counts of specific Ngrams, patterns, and characteristics within the URL that have been used for malicious website detection in prior research and we expand our Ngrams to include other unstudied Ngrams. For example, prior research has considered the number of Top-Level Domains within the URL and specifically has noted whether the URL is a ".com", ".net", *etc.,* URL. Additionally, we looked for the presence and counts of English words within the URL. The full list of Ngrams, patterns, and characteristics are listed below.

- Presence of an IP address, port, or well-known port
- The length of URL
- Count of each character (a to z and special chars)
- Count of total vowels and consonants
- Count of total letters, digits, and special characters (ex. the URL counts twelve letters, eight digits, one special character)
- Counts of file extensions and Top-Level Domains (ex. this URL contains two occurrences of "com")
- Count of respective English words from (Github 2018)
- Count of the number of words of a given length (ex. there are three words with a length of four, two words with a length of five, *etc.*)

Webpage Content Features – We gather webpage content (HTML and JavaScript) features with `Pyselenium` (Pyselenium 2018). Method counts are relevant because certain methods are associated with maliciousness. For example, the `escape()` and `unescape()` methods are often used in JavaScript obfuscation. Obfuscation has legitimate purposes but is often used by malicious developers to make JavaScript more difficult to analyze. We gather method counts of 401 methods from (MDN web docs 2018) – 17 of which have been studied for malicious website detection. They are listed in Table 1.

Table 1 – JavaScript Methods from Prior Research

JavaScript Methods from Prior Research	
Method	*Motivation*
createElement, write	This method modifies the data object model (DOM).
charCodeAt, concat, unescape escape, substring, replace, fromCharCode	This method has been used in JavaScript obfuscation.
Eval	This method enables the execution of a string as code.
Exec	This method and can be used in obfuscation.
link, search	This global method is considered suspicious and has appeared in many types of attacks.
parseInt	This method has been associated with malicious combinations of methods.
addEventListener	Event attachments can be considered.
setInterval, setTimeout	The method has been used in drive-by downloads.

We capture features from the HTML including the counts of tags or elements, details from the URLs on the webpage, and information in the URLs on the webpage. The following HTML produces the proceeding features.

```
<html>
<body>
<a href=http://www.cnn.com/>
<br>
</br>
<a href=http://www.google.com/>
</body>
</html>
<a> count=2
<body> count=1
<br> count=1
Total out of domain URLs=2
Total tag count=5
Total href count=2
<a href ="http*">=2
<a href ="*.com">=2
```

We also collect additional information including the presence of small elements (i.e. `<frame>`, `<iframe>`, *etc.* elements with a length or width less than 2) and static counts of JavaScript methods. In summary, the webpage content features we collected are listed below.

- Static counts JavaScript methods
- Counts of HTML tags

- Small elements – iframes, img, embed, *etc*..
- HTML element attributes
- Out of domain, absolute, and relative URLs
- Extension and protocol of links on the webpage

HTTP Headers Features – To collect these features, we perform a GET `request` using the Python `requests` (Reitz et al., 2019) library and gathered features from the `response`.

4.3. *Feature Collection, Selection, and Transformation*

Feature Collection – We performed feature collection in August of 2018 starting with 40,000 benign websites and 7,039 malicious websites. We recorded the content and the HTTP headers from the `responses` to GET `requests` and discarded entries whose GET `requests` failed. We were able to collect data from 34,742 of the Alexa domains, and 4,441 of the Talos entries. Causes of failed `requests` include timeouts, firewalls blocking our IP address, or the web URL no longer being valid (more prevalent in the malicious dataset). We collected 46,580 features in total – 28,162 derived from the URL, 17,746 derived from the webpage content, and 672 derived from the HTTP `response` headers.

Feature Selection – We split our dataset into two portions – a training set (80% of total data), and a testing set (20% of total data). The training set is used for feature selection and model building, and the testing set is placed aside for evaluation. To perform feature selection, we first remove features that are the same for 95% or more of our training data. We remove additional features with the XGBoost (XGBoost 2019) algorithm as in (McGahagan et al., 2019). XGBoost is a Gradient Boosting algorithm that also computes feature importance – a normalized metric between 0 and 1 that represents how much that feature influences the decision on whether a website is malicious. Feature importance is calculated with Gini Impurity (Raileanu and Stoffel, 2004) and is derived from the average reduction in impurity across all splits for a specific feature. We build a detection model with XGBoost from a 70:30 split of training-to-test data. We calculate the respective feature importances in this model and have a resulting set of features and their corresponding importance. We iteratively enter each feature importance as a threshold to the SelectFromModel (McGrath and Gupta, 2008) technique, a transformer used to select features based on their weights. This produced a set of features for each threshold. We used the set of features associated with each threshold and rebuilt our XGBoost models to obtain an accuracy for each set of features. At this point, we have a list of sets each containing four elements – a threshold (Thresh), number of features (n), set of features (f), and an accuracy. Our goal is to achieve the best performance with as few features as practical. We review the data to locate a relative accuracy maximum. The presence of a relative maximum in accuracy, as "n" decreases, is how we determine which set of features to use for detection. We find a relative maximum accuracy at n=36 features as shown below.

```
Thresh=0.001, n=105, Accuracy: 97.78%
...
Thresh=0.006, n=43, Accuracy: 97.72%
Thresh=0.007, n=36, Accuracy: 97.75%
Thresh=0.009, n=26, Accuracy: 97.69%
```

We calculated the correlation (James et al., 2013) of the features and observed that two features have high correlation to other features in the list with correlation values of 0.84 and 0.93. We removed the features with high correlation to arrive at 34 features. These 34 features are the *discovered* features. This method of feature selection enables us to take an iterative approach of finding a set of features and provides an alternative to other approaches (Zhou et al., 2020). Our study is based on comparing the performance of the models built from the 34 *discovered* features and 99 features we have observed are used in prior research (to include the works reference in this study) that were selected *a priori*.

Feature Transformation – Although we have 34 *discovered* and 99 *a priori* features to build our detection models, we also compare their effectiveness by comparing the detection abilities of features they can produce. We perform feature transformation on the 34 *discovered* and 99 *a priori* features to form additional features and evaluate their ability to detect malicious websites. We perform two types of feature transformation scenarios – feature transformation with feature selection (FT w/FS) and feature transformation with principal component analysis (FT w/PCA) to build a better understanding of how the *discovered* features detection ability compares to that of the *a priori* features. Both involve first transforming the 34 *discovered* and 99 *a priori* features into new features. We refer to these new features as the *transformed* features. Since feature transformation will produce many features, we then perform feature selection with additional selection techniques and dimensionality reduction with PCA to generate a smaller set of features to build our detection models.

The 34 *discovered* and 99 *a priori* features are all numerical. This characteristic allowed us to use pair-wise feature transformations: addition, multiplication, and division. We used the `Python` library, `featuretools` (Featuretools 2019) to create the set of *transformed* features. We then perform feature selection on the *transformed* features using four different techniques: Correlation (James et al., 2013) as used in (Basnet et al., 2012), Select K Best (scoring function chi-square), Recursive Feature Elimination (RFE), and Select From Model (only used with feature transformation with feature selection). The choice of these techniques is motivated by prior research and current data science techniques. We input the *transformed* features into each of these four techniques to produce four sets of features. We keep (select) the transformed features that at least three of these techniques select as relevant. For the feature transformation with PCA, we perform PCA on the *transformed* features to create "n" components, mixtures, or combinations of variables that capture the maximum variance, which are then used to build detection models. By using a cumulative scree plot, we identify components that capture maximum variance between the features within the data.

4.4. *Sampling*

Our dataset has imbalance, 4,441 malicious websites to 34,742 benign websites, which could influence the detection performance of models built in this study and could affect certain algorithms more than others. For example, K-Nearest Neighbor is affected more by class imbalances in training data versus

decision tree-based classifiers. To account for this potential impact, we create three sampling scenarios for the training data from which we will build supervised learning models.

1 *No-sampling* – training data are used as is;
2 *Over-sampling* – malicious websites are over-sampled with the SMOTE technique (Chawla et al., 2002) provided by (SMOTE, 2019) to equal the number of benign websites; and
3 *Under-sampling* – benign websites are randomly under-sampled to equal the number of malicious websites.

4.5. *Unsupervised and Supervised Learning*

We leverage unsupervised and supervised learning algorithms to build malicious website detection models. We capture the accuracy, AUC, MCC, Precision, and Recall for each model. We focus our discussion on the MCC (1 indicates a perfect classifier, 0 a random classifier, and -1 complete disagreement between the predicted and actual value).

Models Using Unsupervised Learning Algorithms – Unsupervised learning algorithms, such as clustering and anomaly detection, has been used to detect malicious websites and traffic in (Cova et al., 2010) and (Perdisci et al., 2013; Perdisci et al., 2010. Clustering is more applicable to distinguishing and discovering different classes within a dataset. For example, (Nelms et al., 2013) used clustering to distinguish HTTP traffic among different families of malware. Since we have only two data classes (malicious and non-malicious), we chose to leverage an anomaly detection approach where benign websites are the normal case and the malicious websites are anomalous. We build unsupervised models using Autoencoders (Baldi, 2012) with Keras (Keras, 2020) and a One Class SVM from the 34 *discovered* features, 99 *a priori* features, *transformed* features using the FT w/FS technique and components from the FT w/PCA technique. For autoencoders, we train our model on benign websites only so that the model learns the features of a benign website. We identify malicious websites based on the reconstruction error of the test dataset. For the One Class SVM, we treat the benign websites as inliers and the malicious websites as outliers.

Models Using Supervised Learning Algorithms – We also leverage eight supervised learning algorithms to detect malicious websites that fall into categories of learning used in prior research: Nearest Neighbor (Peterson, 2009), and Ensemble Methods (Dietterich, 2000). Supervised learning has often performed better than unsupervised learning in prior research. Additionally, we have thousands of examples of both malicious and benign websites that can be used in training of supervised learning models. The choice of building eight models was motivated by two factors. First, we wanted to explore performance of various models built with the *discovered* and *a priori* features to gain a more thorough understanding of the features' detection ability. Second, we found that (Eshete et al., 2012) leveraged seven different supervised algorithms which were combined with a voting (Ruta and Gabrys, 2005) algorithm.

Of our eight models, five are Ensemble methods and provide a measure of feature importance (The Artificial Imposter 2019) based on the Gini Impurity: (Adaptive (Ada) Boosting (AB) (Schapire, 1999), Extra Trees (ET) (Geurts et al., 2006), Random Forest (RF) (Breiman, 2001), Gradient Boosting (GB), and XGBoost (XGB)). Feature importance enables us to examine which features contribute the most to the classification decision and allows us to create a ranking of the most important features for detection. The other three models do not provide a measure of feature importance. However, they provide additional insight into how the selected features perform across well-known machine learning algorithms and enable a more thorough comparison of the two sets of features. They are the Bagging Classifier (BC) (Breiman, 1996), an Ensemble Method, and K-Nearest Neighbors (KNN), a Nearest Neighbor Method. We also build a Voting classifier (V) created from the Random Forest, K-Nearest Neighbors, and Bagging classifiers with a 'soft' voting configuration. When building the models, we use the default parameters provided by (Scikit-Learn 2020) for the respective models to reduce subjectivity and to be consistent. Our analysis in sections 5.1 through 5.4 is done on the models built with default parameters. We also perform hyperparameter tuning and cross-validation (Kohavi, 1995) where we vary the model hyperparameters and adjust the training and testing data with Kfold (Friedman et al., 2009) validation. Doing so attempts to improve our models. This is discussed in Section 4.6.

The supervised models are built and evaluated with the 34 *discovered* features and 99 *a priori* features in the three sampling scenarios discussed in Section 4.4. In addition to evaluating the supervised models in the sampling scenarios, we also rebuild and evaluate the eight models using the features created by FT w/FS and by FT w/PCA. Full confusion matrices are available in Section 10.

4.6. *Hyperparameter Tuning and Cross-Validation*

In our last step, we perform parameter tuning and cross-validation on our supervised models to improve them. We tune the respective hyperparameters for each model and perform cross-validation. We use StratifiedKFold (Friedman et al., 2009) for 5-fold cross-validation (Kohavi, 1995) and used the balanced accuracy scoring metric, provided by (Scikit-Learn 2020), when performing Grid Search (Syarif and Wills, 2016) in an attempt to optimize our performance. Hyperparameters are specific to each classifier and include the number of neighbors, weights, number of estimators, max number of features, max leaf nodes, boosting algorithm, and 'hard' or 'soft' voting configuration. Full confusion matrices are available in Section 10.

5. Results

5.1. *Unsupervised Results*

In the unsupervised case, we observe an average MCC of 0.5491 with *discovered* features and 0.4699 with *a priori* features. Full results are shown in Fig.s 2 and 3.

The unsupervised models do not perform well though we do see improvement in the feature transformation scenarios. The detection results are not great hence we focus most of our analysis on supervised methods.

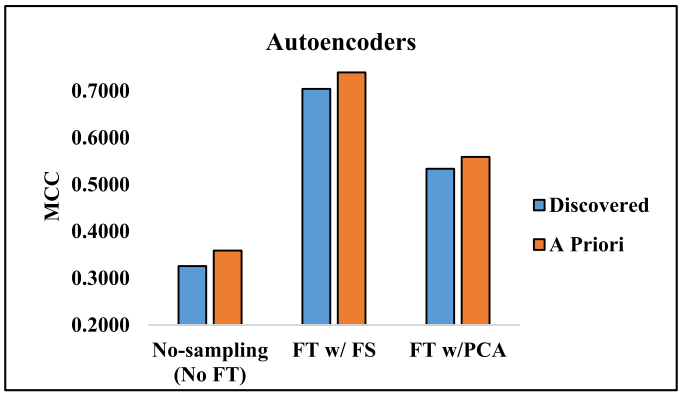

Fig. 2 – Unsupervised Autoencoders

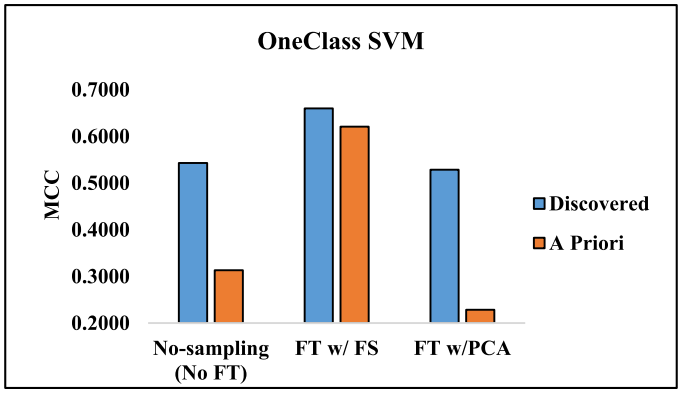

Fig. 3 – Unsupervised One Class Support Vector Machine

5.2. *Feature Importance Analysis*

Rankings of the 34 *discovered* features and their respective types – Webpage Content (WC), URL, and HTTP – are shown in Table 2 in the no, over, and under-sampling scenarios. New features are shaded in gray and the rank and importance are separated by a colon.

The total number of HTML tags and the URL length features are consistent across the sampling scenarios and account for an average 22.7% of total feature importance. Half of the *discovered* features are from the webpage content and account for 43% of total feature importance, and, of the webpage content features, only one feature – counts of the `push()` method – is a JavaScript feature. Eleven of the *discovered* features are URL features which account for 44% of total feature importance, and the remaining six features are HTTP headers which account for 13% of total feature importance. Four out of the top five *discovered* features with the highest average importance are URL features (URL length, total extensions in the URL, the count of 'w' and '.' characters), and account for approximately ˜31% of total feature importance. Nine of the *discovered* features are HTML features that represent resources via the `src` or `href` attributes or other attributes that can specify resources and account for 17% of total feature importance. Twelve of the *discovered* features, to our knowledge, have not been studied for their role in malicious website detection and account for an average 17% of total feature importance.

The total number of HTML tags is the most important *discovered* feature and is part of a webpage's *content complexity* (Butkiewicz et al., 2011). More content requires additional analysis if the webpage is malicious and provides added opportunities to place malicious content inside the webpage. For example, a webpage consisting only of text will not cause a drive-by download, whereas a page with various links, JavaScript, and other resources such as <`iframe`>s, may enable a drive-by download. The next *discovered* feature is frequently used in prior works – the URL length. This feature is not surprising since attackers use 'tiny' as well as longer URLs (McGrath and Gupta, 2008).

URL features, especially the counts of the respective characters in the URL, are observed to be helpful for detecting malicious websites. The count of 'w' characters along with counts of 'z', 'i', 'l', 'y', and 'p' all appear in our list. There are no specific known associations of these characters with malicious URLs. However, all of these, except for the letters 'i' and 'l', are infrequently used in the English language with z, y, p, w occurring 0.27%, 1.77%, 3.16% and 1.28%, respectively, of the time (Keating, 2020). The letters 'i' and 'l' are prevalent in Kwyjino malware (Yadav et al., 2010). Furthermore, features derived from character counts including ratios, distance vectors, and similarity have been studied to detect bots and

Table 2 – Discovered Features Ranked

Feature – Type	*No*	*Over*	*Under*
Total HTML Tags - WC	1: 0.131	1: 0.131	1: 0.113
URL Length - URL	2: 0.128	2: 0.084	2: 0.092
Total URL Extensions - URL	3: 0.113	3: 0.071	4: 0.062
Count of 'w' character – URL	4: 0.073	5: 0.057	5: 0.056
Count of '.' Character - URL	5: 0.049	7: 0.041	9: 0.040
content-encoding gzip - HTTP	6: 0.038	6: 0.048	8: 0.044
<a href> relative – WC	7: 0.037	21: 0.017	13: 0.026
Count of <meta> tag - WC	8: 0.032	4: 0.059	7: 0.049
<a href> OoD – WC	9: 0.030	17: 0.021	6: 0.049
server apache age - HTTP	10: 0.024	20: 0.018	16: 0.022
Count of 'z' character - URL	11: 0.023	25: 0.014	21: 0.013
<link type="text/css"> - WC	12: 0.023	9: 0.034	10: 0.038
<img src="http*"> - WC	13: 0.021	24: 0.014	32: 0.007
Count of 'i' character - URL	14: 0.019	15: 0.026	22: 0.013
Count of <p> tag – WC	15: 0.019	18: 0.019	14: 0.025
push()- WC	16: 0.017	14: 0.027	15: 0.023
Count of 'l' character - URL	17: 0.016	11: 0.031	34: 0.007
url extension is .com - URL	18: 0.015	22: 0.016	23: 0.013
Count of 'y' character - URL	19: 0.015	34: 0.003	30: 0.007
vary user-agent - HTTP	20: 0.015	27: 0.012	27: 0.010
Total href attributes – WC	21: 0.014	10: 0.033	3: 0.073
<a href="https*"> - WC	22: 0.014	30: 0.008	11: 0.033
<link href> OoD – WC	23: 0.013	8: 0.037	12: 0.027
cache-control max-age - HTTP	24: 0.013	13: 0.027	18: 0.018
Count of 'p' character - URL	25: 0.012	32: 0.005	26: 0.011
<form action="http*"> - WC	26: 0.012	26: 0.013	33: 0.007
Count of <a> tag – WC	27: 0.011	16: 0.025	19: 0.018
transfer-encoding chunked - HTTP	28: 0.011	12: 0.028	17: 0.022
Count of 'f' character – URL	29: 0.010	29: 0.009	28: 0.009
<script async="true"> - WC	30: 0.009	23: 0.016	20: 0.016
via 1.1 - HTTP	31: 0.008	28: 0.012	24: 0.012
<a href="http*"> - WC	32: 0.008	33: 0.004	31: 0.007
Count of tag - WC	33: 0.006	19: 0.018	25: 0.011
<iframe src="*..html"> - WC	34: 0.006	31: 0.008	29: 0.008

C2 URLs (Ahluwalia et al., 2017) when used in conjunction with unsupervised learning algorithms such as clustering or anomaly detection. Lastly, character counts and related metrics are needed to further study URLs generated by domain generation algorithms. Hence, character counts (and features derived from them) are relevant.

The first HTTP header *discovered* feature we observed has been used in specific attacks and includes features that describe how data are packaged in a `response` and the amount of time data should be kept by the client. The `gzip content-encoding` header extracted from the HTTP `response` has legitimate uses, but it can also be used to evade network scanners (Ullrich, 2013). `Chunked responses`, which also appear on the list, are similar in that they are specified in a `response` header, and have legitimate purposes, but can also be used by attackers (Ullrich, 2013). The `cache-control` header has been studied in (Xu et al., 2013) and `max-age` directive can be used by attackers to instruct a cache to hold a malicious `response` for a long period which enables a cache poisoning attack (OWASP 2020).

Other HTTP headers appear on the list and represent other potential security issues. The `server` header with a value of `apache`. Apache servers have had many well-known security issues with some enabling backdoors (control) for attackers (Apache, 2019)- (ArsTechnica, 2013), (Info Security, 2013), (ZdNet, 2013). The `via` (MDN web docs 2020) header is added by proxies which have legitimate uses however are also well-known to be used by attackers (Security Boulevard, 2019) and (Zaborowski, 2017). The `vary` header can be used by `requests` or `responses`. When used by `responses` with a value of `user-agent`, it specifies whether `responses` will be cached based on the `user-agent` string. This feature is well-known to be misunderstood and misused (SiteGround, 2017) by developers.

We also observe many webpage content features related to links and URLs on the webpage. The total number of relative links or resources, as opposed to absolute links, point to resources within the page (or relative to the page). This is one of the higher ranked *discovered* features and is an extension of features used in (Ma et al., 2011), and is known to be used by attackers. It is also a measure of *content-complexity* (Butkiewicz et al., 2011). One advantage of using relative links is that it lowers the chances of detection for attackers. For example, if the webpage is fetched and successfully loaded, it has gotten past some network-level defenses or other security solutions (Security Week, 2018). Furthermore, relative URLs are known to be leveraged by attackers. Recently, Microsoft's ATP for phishing detection has been bypassed using relative URLs (Security Week, 2018). Additionally, the number of out-of-domain (OoD) links (URLs that are out of the current domain) on the page ranks as a feature. The more links on a webpage, the more chances for potential malicious URLs – only one of them needs to be effective to cause an infection. We also find that the structure of the URLs on the webpage are present in our features. For example, we observe that the protocol (`http` vs `https`) for certain resource references (such as image, links) help with detection, however they are not highly ranked. The `<iframe>` is well-known for its ability to detect drive-by downloads (Provos et al., 2008) and we observe that `.html` files in the `<iframe> src` attribute make the list.

The next *discovered* webpage content features deal with specific tags (elements on the webpage). The `<meta>` tag is known to be associated with malicious redirects (Avanan, 2019). Two other tags that appear are counts of `<p>` and `<center>`. Both are formatting tags and specify how text should be rendered. Although they have no known relation to malicious websites, the counts may provide additional insight as to the level of care attackers place in formatting their text which would be of interest to phishing detection. The other *discovered* feature, `<link type>`, identifies references to `.css` resources. Although this has legitimate uses, CSS files have also been used for attacks (Gualtieri, 2018).

We observe that one JavaScript *discovered* feature, part of webpage content, makes our list – counts of the `push()` method. Certain JavaScript methods such as `escape()` and `charCodeAt()` are relevant to detecting malicious JavaScript. Unlike the `escape()` and `charCodeAt()` methods, the `push()` method is not highly related to malicious JavaScript. Although it makes our list, it does not rank highly and it should be noted that this study performs static analysis, however dynamic analysis can provide better insight into which methods are being executed since malicious

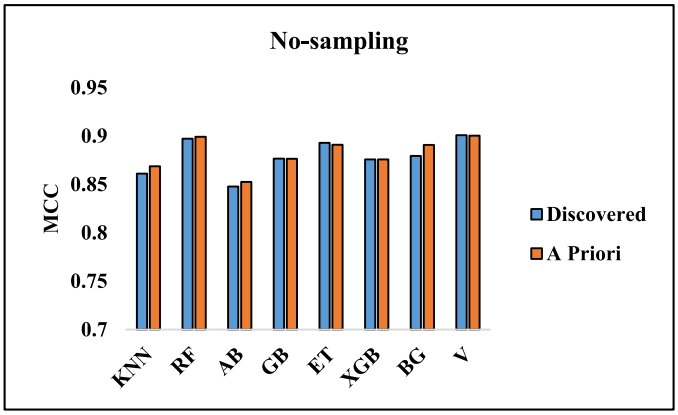

Fig. 4 – Supervised Classifiers from *Discovered* vs. *A Priori* Features, No- sampling Scenario

JavaScript is often obfuscated (most prior research requires de-obfuscation before analysis of code as in (Cova et al., 2010) and (Curtsinger et al., 2011). Although we only find one count of a JavaScript method, the `async=true` attribute on the `<script>` tag is a potential attack vector as it instructs the browser to continue rendering third party libraries in the JavaScript.

We then performed feature ranking of the 99 *a priori* features. Webpage content features account for 40% of total feature importance, URL features account for 48% of total feature importance, and HTTP features account for 12% of total feature importance. Whereas the top two *a priori* features in Table 2 are consistent, none of the top *a priori* features in Table 3 (with an importance greater than 0) are consistent. Also, we observe that some *a priori* features have little to no importance in our study. Full rankings for the 99 *a priori* features in sampling scenarios are shown in Table 3. A feature importance of less than 0.001 is considered 0.

5.3. *Sampling Scenarios*

In the no-sampling scenario, we observe that three out of the eight models improve when using *discovered* features versus *a priori* features. The average MCC changes by -0.0029 when using *discovered* features instead of the *a priori* features. Full results are shown in Fig. 4.

In the over- and under-sampling scenarios, we observe that three and four out of the eight models improve with *discovered* features versus *a priori* features respectively. The average MCC changes by -0.0064, in the over-sampling scenario and -0.0086 in the under-sampling scenario with *discovered* features instead of *a priori* features. We observe similar behavior as in the no-sampling scenario. Full results are shown in Fig. 5 and Fig. 6.

5.4. *Feature Transformation*

We now discuss the results of using features created through feature transformation with feature selection and from feature transformation with PCA for the *discovered* and *a priori* features. For the *transformed* features (2,278 features were created from the 34 *discovered* features, of which 37 were selected, and 19,503 were created from the 99 *a priori* features, of which 27 were selected) we observe a change in MCC of 0.0034 and five of eight models improved when using features created from *discovered* features versus *a priori* features. The effect of the feature transformation with feature selection is shown in Fig. 7.

One hundred and twenty-five features were created with PCA from the 34 *discovered* features, compared to 750 components created from the 99 *a priori* features. With components from *discovered* features, we observed a change in MCC of 0.0346 and all eight models improved compared to using models built with the components from the *a priori* features. The effect of the feature transformation with PCA is shown in Fig. 8.

5.5. *Hyperparameter Tuning and Cross-Validation*

We tuned the hyperparameters of the models in each scenario built from an 80:20 split of train-to-test data (no-sampling, over-sampling, under-sampling, FT w/FS, and FT w/PCA). In this section, we compare the results of the tuned models to their respective non-tuned counterparts and we compare the tuned models built from the *discovered* features to their respective counterparts built with the *a priori* features. Results are shown in Fig.s 9 through 13.

We first compare the effect of tuning. For the *discovered* features – two of eight models improved in the no-sampling scenario, four of eight improved in the over-sampling scenario and under-sampling scenarios, five of eight improved in the FT w/FS scenario, and six of eight improved in the FT w/PCA scenario resulting in an average change in MCC of -0.0003, 0.0035, -0.009, 0.0042, and 0.0031 respectively to their non-tuned counterparts. For the *a priori* features – two of eight models improved in the no-sampling scenario, three of eight improved in the over-sampling scenario and under-sampling scenarios, and two of eight improved in the FT w/ FS and FT w/PCA scenarios scenario resulting in an average change in MCC of -0.0011, 0.0053, -0.0225, -0.0024, and -0.0016 respectively.

We now compare the tuned models' performance built from the *discovered* to the tuned models built from the *a priori* features. In the no-sampling scenario, three of the eight

Table 3 – A Priori Features Ranked

Feature – Type	*No*	*Over*	*Under*
Total HTML Tags – WC	1: 0.115	6: 0.038	1: 0.135
Total Extensions in URL - URL	2: 0.095	3: 0.052	5: 0.049
URL Length - URL	3: 0.092	4: 0.049	3: 0.052
Count of 'w' character - URL	4: 0.058	5: 0.049	4: 0.049
Count of '.' Character - URL	5: 0.046	9: 0.022	11: 0.021
<a href> OoD – WC	6: 0.029	15: 0.018	6: 0.038
Total href – WC	7: 0.028	10: 0.021	2: 0.055
server apache - HTTP	8: 0.026	11: 0.021	10: 0.021
content-encoding gzip - HTTP	9: 0.025	2: 0.055	7: 0.037
Count of <link> tag – WC	10: 0.024	19: 0.017	12: 0.019
Count of <meta> tag – WC	11: 0.022	7: 0.037	8: 0.022
Total TLDs in URL- URL	12: 0.020	33: 0.009	20: 0.017
content-language text/html - HTTP	13: 0.019	18: 0.018	13: 0.019
Count of 'z' character - URL	14: 0.019	23: 0.015	19: 0.017
<link href> OoD – WC	15: 0.015	20: 0.017	22: 0.016
Count of <a> tag – WC	16: 0.014	12: 0.019	9: 0.022
Count of 4-character words - URL	17: 0.014	27: 0.012	23: 0.015
Count of 'y' character - URL'	18: 0.014	49: 0.004	56: 0.003
url extension is .com - URL	19: 0.013	41: 0.005	26: 0.013
Total <img src> - WC	20: 0.013	16: 0.018	15: 0.018
content-length - HTTP	21: 0.013	36: 0.008	17: 0.018
<form action> OoD – WC	22: 0.013	22: 0.016	21: 0.016
Count of tag – WC	23: 0.013	25: 0.013	14: 0.019
cache-control max-age - HTTP	24: 0.012	21: 0.016	16: 0.018
Count of 'i' character- URL	25: 0.012	8: 0.022	24: 0.014
Count of 'l' character- URL	26: 0.012	14: 0.019	49: 0.004
Count of <style> tag – WC	27: 0.011	58: 0.003	34: 0.009
<script src> OoD – WC	28: 0.009	46: 0.004	18: 0.018
Count of 'p' character - URL	29: 0.009	26: 0.0133	27: 0.012
Count of <title> tag – WC	30: 0.008	1: 0.1358	41: 0.005
Count of 'f' character - URL	31: 0.008	30: 0.0104	31: 0.010
<script type=text/javascript> - WC	32: 0.007	37: 0.007	28: 0.012
Count of 'd' character - URL	33: 0.007	17: 0.018	35: 0.008
Count of 's' character - URL	34: 0.007	28: 0.012	32: 0.009
Count of 5-character words - URL	35: 0.007	38: 0.006	62: 0.002
Count of 'o' character - URL	36: 0.006	29: 0.011	44: 0.005
cache-control no-store - HTTP	37: 0.006	57: 0.003	29: 0.011
url extension is .i - URL	38: 0.006	40: 0.006	46: 0.004
replace() - WC	39: 0.006	34: 0.009	40: 0.006
Count of 'u' character - URL	40: 0.006	43: 0.005	30: 0.010
Count of <img> tag – WC	41: 0.005	61: 0.002	36: 0.008
<img srcset> OoD – WC	42: 0.005	39: 0.006	45: 0.005
Count of 'b' character - URL	43: 0.005	32: 0.009	60: 0.002
Count of 'e' character - URL	44: 0.005	45: 0.005	58: 0.003
addEventListener() - WC	45: 0.005	44: 0.005	33: 0.009
<base href> OoD - WC	46: 0.004	76: 0	74: 0
Count of 't' character - URL	47: 0.004	59: 0.003	48: 0.004
Count of <iframe> tag - WC	48: 0.004	63: 0.002	55: 0.0037
Count of 'x' character - URL	49: 0.004	70: 0.001	72: 0
Count of 'c' character - URL	50: 0.003	47: 0.004	47: 0.004
Count of 'r' character - URL	51: 0.003	51: 0.004	65: 0.001
Count of 'm' character - URL	52: 0.003	55: 0.003	52: 0.004
Count of 'h' character - URL	53: 0.003	35: 0.008	66: 0.001
Count of 'a' character - URL	54: 0.003	13: 0.019	61: 0.002
server nginx - HTTP	55: 0.003	54: 0.003	38: 0.006
createElement() - WC	56: 0.003	24: 0.014	25: 0.013
Count of 'n' character - URL	57: 0.002	52: 0.004	50: 0.004
cache-control no-cache - HTTP	58: 0.002	56: 0.003	57: 0.003
Count of 6-character words - URL	59: 0.002	31: 0.010	63: 0.002
Count of 7-character words - URL	60: 0.002	64: 0.001	64: 0.001
Count of <input> tag - WC	61: 0.002	65: 0.001	43: 0.005
Count of 'k' character – URL	62: 0.002	66: 0.001	67: 0.001
Count of 'g' character - URL	63: 0.002	50: 0.004	37: 0.007
<img src> OoD - WC	64: 0.001	62: 0.002	42: 0.005
Count of '-' character - URL	65: 0.001	42: 0.005	39: 0.006
Count of 'v' character r- URL	66: 0.001	69: 0.001	59: 0.003
write() - WC	67: 0.001	74: 0	69: 0.001
cache-control must-revalidate -HTTP	68: 0.001	48: 0.004	54: 0.003
Count of 'j' character - URL	69: 0.001	71: 0	68: 0.001

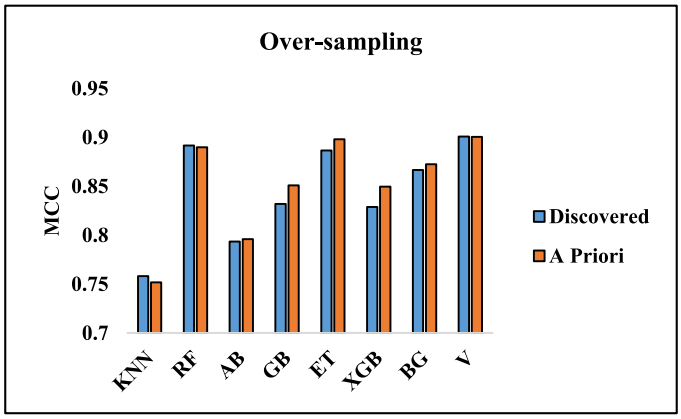

Fig. 5 – Supervised Classifiers from *Discovered* vs. *A Priori* Features, Over-sampling Scenario

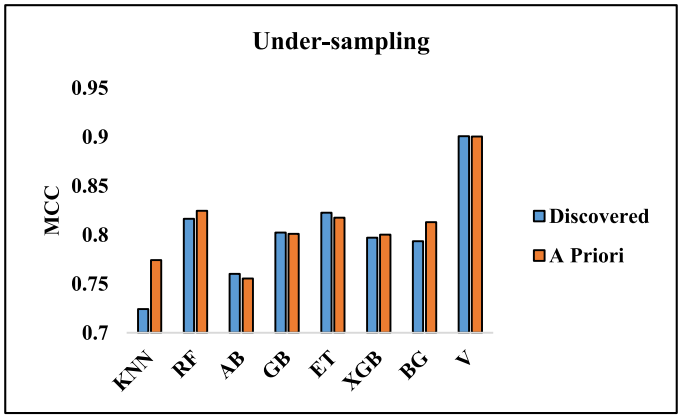

Fig. 6 – Supervised Classifiers from *Discovered* vs. *A Priori* Features, Under-sampling Scenario

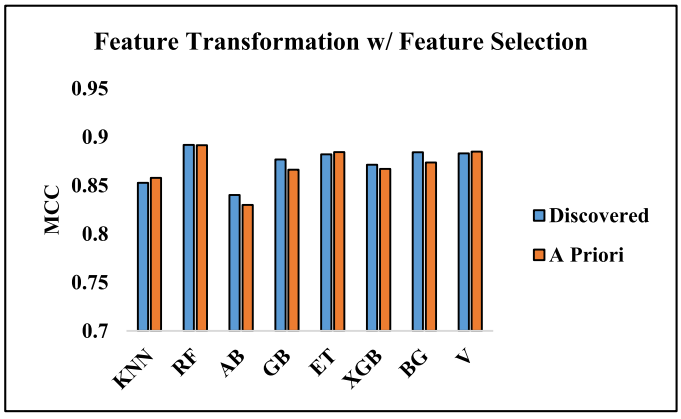

Fig. 7 – Supervised Classifiers from *Discovered* vs. *A Priori* Features, Feature Transformation with Feature Selection Scenario

models improved. In the over-sampling scenario, two of the eight models improved. In the under-sampling scenario, two of the eight models improved. In FT w/FS scenario, all eight models improved. In the FT w/PCA scenario, all eight models improved. When using *discovered* features versus *a priori* features, the MCCs changed by -0.0047, -0.0093, -0.0104, 0.0052, and 0.0314 in these scenarios respectively.

6. Discussion

In Research Question 1 we examined the detection ability of *discovered* features. With the *discovered* features we can obtain a best classifier performance of an MCC 0.7043 using unsupervised learning algorithms, and 0.9008 using supervised learn-

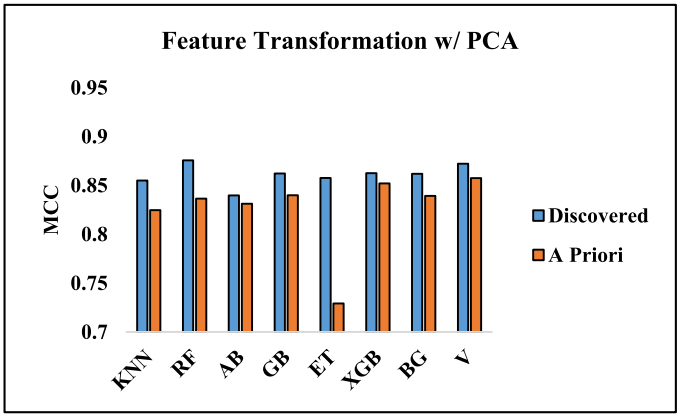

Fig. 8 – Supervised Classifiers from *Discovered* vs. *A Priori* Features, Feature Transformation with PCA Scenaario

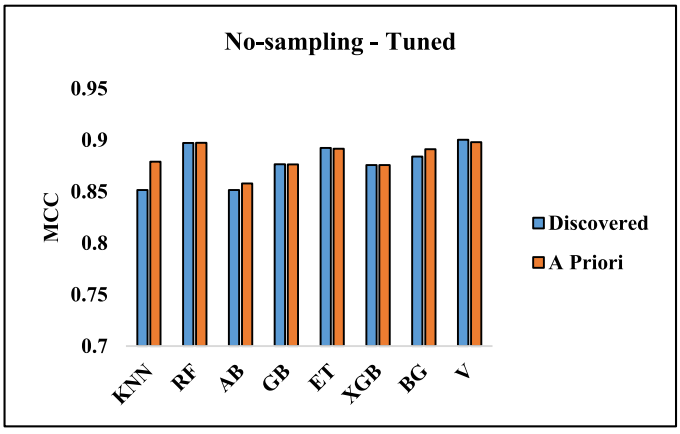

Fig. 9 – Tuned Supervised Classifiers from *Discovered* vs. *A Priori* Features, No-sampling Scenario

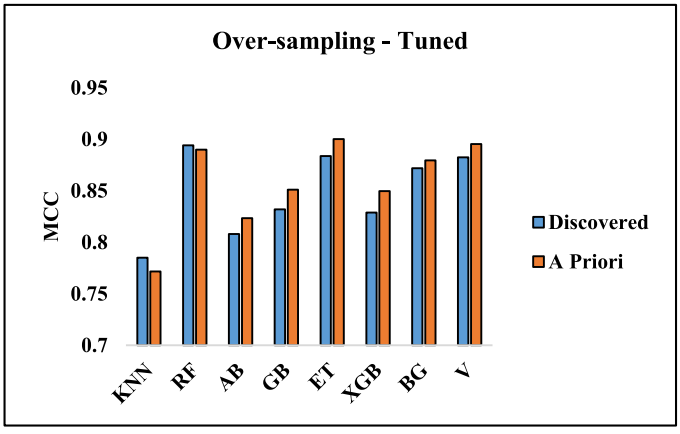

Fig. 10 – Tuned Supervised Classifiers from *Discovered* vs. *A Priori* Features, Over-sampling Scenario

ing algorithms. The unsupervised results are not promising. However, unsupervised learning algorithms are not as common for detecting malicious websites (except for C2 traffic). The results using supervised learning algorithms suggest that a discovery approach can be used to identify features. We envision that this method could be conducted on a routine basis to identify features for detection. However, for further insight into the feasibility of this approach, we need to compare the detection performance of models built from *discovered* features identified in this paper to detection performance of models from prior research that uses the notion of *a priori* features.

Ma in (Ma et al., 2009) leveraged URL and host-based (DNS queries, WHOIS properties, and geographic information) and

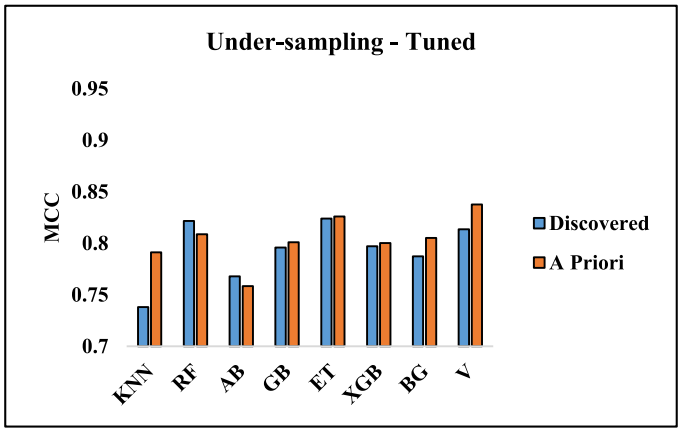

Fig. 11 – Tuned Supervised Classifiers from *Discovered* vs. *A Priori* Features, Under-sampling Scenario

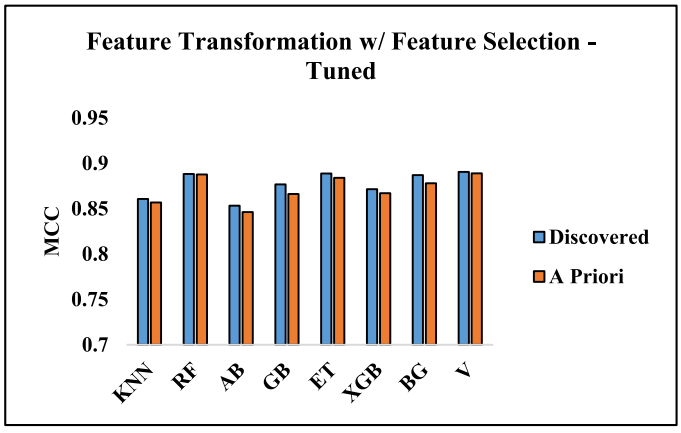

Fig. 12 – Tuned Supervised Classifiers from *Discovered* vs. *A Priori* Features, Feature Transformation with Feature Selection Scenario

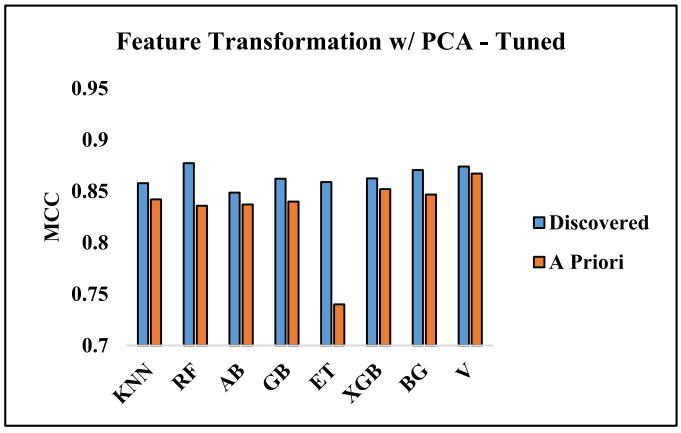

Fig. 13 – Tuned Supervised Classifiers from *Discovered* vs. *A Priori* Features, Feature Transformation with PCA Scenario

incorporated Online Learning in (Ma et al., 2009)-[44] and an SVM classifier to identify malicious URLs. Ma used strictly URL and host-based features and focused on training an SVM several times with Online Learning. We quantify best-case performance differently – Ma with error rate (best-case 2.6%) while we focus on MCC (best-case 0.9008) due to our dataset imbalance. Whittaker (Whittaker et al., 2010) also took an online approach and trains and evaluates a Logistic Regression classifier on millions of webpages (with different algorithms), focused on phishing, and presented their results as a trade-off between precision and recall with their best performance being a precision and recall over 0.95 respectively. Prophiler

(Canali et al., 2010) also focused on classifying many malicious webpages with an evaluation dataset of almost 19 million webpages and achieved a false positive and false negative rate of 9.88% and 0.77% with HTML, JavaScript, URL, and host features. They used Naïve Bayes, Random Forest, Logistic Regression, and other decision tree algorithms.

CANTINA+ (Guang et al., 2011), like us, extracted webpage features and investigated several learning algorithms - Support Vector Machines, Logistic Regression, Bayesian networks, J48 decision tree, Random Forest, and AdaBoost. Unlike us and like Ma, they relied on external sources (for example Page Rank) for features which differs from our approach of only using web `response` features. Their study included two phishing sets of 1,595 and 624 webpages respectively. They achieved a true positive and false positive rate of 4.24% and 1.948% respectively.

Marchal (Marchal et al., 2016) detected phishing with a Gradient Boosting (provided by `Scikit Learn`) classifier with high performance metrics (an AUC up to 0.999 compared to our AUC of 0.9333 in the no-sampling scenario with the Voting classifier) and differentiated themselves by detecting phishing webpages in different languages. They selected 212 features (including the URL and webpage content) for detection, many of which overlap with other prior works. Their dataset consisted of 150,000 legitimate phishing URLS and 3,366 phishing URLs. Like our study, they performed cross-validation.

Phishmon (Niakanlahiji and Al-Shaer, 2018) achieved an accuracy of 95.4% (compared to our best accuracy of 98.03%, which coincides with the 0.9008 MCC, achieved with 34 *discovered* features with the Voting classifier in the no-sampling scenario) with a false positive rate of 1.3% on a dataset of 17,508 legitimate and 4,807 phishing webpages. Their approach, unlike other approaches and like our approach, incorporated all HTTP headers in conjunction with webpage content features for detection. They used four different classifiers (classification and regression trees, K-Nnearest Neighbor, AdaBoost, and Random Forest) and provide a notion of feature importance.

BINSPECT (Eshete et al., 2012) who had a similar approach (aside from feature discovery) to our own used several machine learning classifiers (J48, Random Tree, Random Forest, Naïve Bayes, Bayesian networks, Support Vector Machine, and Logistic Regression) and some of our *discovered* features overlap with their features. However, our study discovered all our features and did not require external sources such as search engine results. We also differed in that our dataset contains C2 URLs while theirs did not. Our accuracies were similar – their accuracy is 97.81% compared to our accuracy of 98.03%. In addition, we observe that 22 of the 34 features *discovered* have been used in prior research. Additionally, most of the features in the *discovered* list have some known association with attacks or malicious techniques. Cova (Cova et al., 2010) leveraged *a priori* features that can identify malicious or suspicious JavaScript and anomaly detection to create JSAND. They evaluated their approach on 823 samples from four different data sources and achieved a false negative rate of 0.2%. Xu (Xu et al., 2013) took the closet approach to our own regarding the features for detection. However, they also included other features like network traffic summaries, which require additional overhead, and they performed feature selection from their *a priori* features for detection. Their approach evaluated four different classification algorithms – Naïve Bayes, Logistic Regression, Support Vector Machine, and J48 and achieved a best-case 99.986% accuracy and they noted the five most selected features.

Basnet (Basnet et al., 2012) used correlation-based feature selection and wrapper feature selection to find relevant features among 177 URL, webpage content, and search engine features along with a Naïve Bayes, Logistic Regression, and Random Forest classifier. They observed wrapper-based feature selection techniques could improve false negative rates by 44.5%. Li (Li et al., 2020), like our study, performed feature transformation during their detection of malicious URLs and did so on seven domain-based, 21 host-based, six reputation-based, and 28 lexical features. Their goal was to demonstrate the benefit of feature transformation when used with different algorithms and noted that feature engineering improved detection rates from 68% to 86%, 58% to 81% and 63% to 82% for K-Nearest Neighbor, Support Vector Machine, and Neural Networks (Bishop, 1995) classifiers respectively and their best accuracy was measured at 97.80%.

In (McGahagan IV et al., 2019) and (McGahagan et al., 2019), the authors discovered and ranked features for malicious website detection but used only webpage content and HTTP headers, performed feature selection, and achieved an MCC of 0.67 and 0.71 respectively which is much lower (and not sufficient for adoption) than our best MCC of 0.9008.

Our approaches, however, are similar but they do not consider URL features which have demonstrated success in prior research. McGahagan (McGahagan et al., 2019) evaluated the performance of several algorithms and used XGBoost for their feature selection.

There are similarities as well as differences between our approach and results from those of prior research. First, we note that our results are comparable (and often better) than those from prior research. However, true comparison is difficult. Marchal (Marchal et al., 2016) and Xu (Xu et al., 2013) have achieved highly accurate results with *a priori* features. Marchal focused solely on phishing, and Xu included additional features like network traffic statistics. Marchal and Xu achieve better results than any prior research we have encountered. Second, we observed that features derived from a web `response` are simpler to gather. Page rank, WHOIS information, network traffic statistics all require additional instrumentation and overhead. Based on our detection metrics as well as on comparisons to prior research, we postulate that feature discovery is feasible for malicious website detection.

Our second research question compares the ability of the *a priori* features from prior research to those found via a discovery approach. In the sampling scenarios we did not see much change in detection performance when using *discovered* features versus *a priori* features. With feature transformation, however, *discovered* features outperformed *a priori* features especially when using PCA. This observation occurred with both the tuned models and the non-tuned models. Hence, we postulate that *discovered* features can be used to create better transformed features for detection, in addition to requiring fewer features for detection. However, we did observe that, on average, the feature transformation and PCA scenarios under-

performed the no-sampling scenario with no feature transformation or PCA. Thus, we answer Research Question 2 in the following fashion. *Discovered* features produced similar performance to those selected *a priori* and did so with 66% fewer features. When incorporating feature transformation, especially PCA, we observed that *discovered* features outperformed the *a priori* features however the performance of the models with transformation was, on average, lower than the performance of models without (in the no-sampling scenario) feature transformation.

Our last research question explores whether the discovery approach can be applied to a set of several threats with a limited number of features (those that can be gathered from the `response` to web `request`). We designed our experiment to simulate the real-world constraints by using a dataset consisting of several threats, and leveraged techniques from prior research. Furthermore, we have limited insight into these threats (we did not hand select them nor are they homogenous) which simulates real-world constraints, and we include C2 URLs in our dataset which is missing from other studies that detect multiple types of threats. To examine this question, we look further into the performance metrics. Overall, our accuracy, AUC, and MCC perform well and are comparable to (and sometimes exceed) the accuracy of other approaches. However, our findings do suggest that this approach alone is not enough. To further examine whether this approach can be a supplement to other detection capabilities, we examine the false positive rate of our models. One of the challenges in practical detection solutions is the large number of false positives (Bloomberg, 2019). The false positive rate of our best performing model in the no-sampling scenario is 0.56% (the non-tuned Voting classifier) and our worst performing classifier in the no-sampling case (AdaBoosting) had a false positive rate of 1.3%. Furthermore, the features extracted in this approach can be extracted from a web `request response` which can be added to other security solutions. As a result, we postulate that a discovery approach, while not sufficient in isolation, can be used as a supplement to other detection techniques.

We observe that we produce models with fewer features that obtain similar (and sometimes better) performance than those from prior research. Fewer features enable us to describe the models in a simpler fashion and reduces the complexity of building and evaluating the models. Although there is no hard limit for how many features are acceptable, fewer features may reduce the risk of over-fitting our models. Further, we demonstrated that while features from prior research have been successful, a re-evaluation can yield positive results and can be helpful given the dynamic nature of the internet and web threats.

7. Limitations

Dataset selection is challenging and presents limitations to our study. First, we use the Alexa Top Domains for our benign websites, and a list of websites containing phishing, drive-by downloads, and C2 infrastructure from Cisco Talos Intelligence Group as our malicious dataset. Both datasets were created by external parties which has both advantages and disadvantages. The first advantage is that we can evaluate our approach as a "black box" and have a smaller chance of unintentionally influencing our dataset. The second advantage is that our malicious dataset mimics an operational scenario's constraints since it consists of several types of threats. However, based on the size of our dataset, we cannot manually verify each entry.

Second, our dataset represents a single snapshot in time. The internet is a dynamic environment and observations made for a single point in time are subject to change. Furthermore, the security realm is challenging because adversaries adapt their attacks and techniques. Thus, examples of malicious websites and techniques (used as training data) may quickly become out of date. Our study does not address whether and how features in benign and malicious websites change over time.

Third, by using the Alexa Top Domains as our benign dataset, we assume that popularity is a trait of a benign website. This is not guaranteed, however, to investigate this assumption, we gathered threat intelligence data from Cymon.io (Cymon.io 2019) for the duration of 2018 to determine how many of our benign websites were included in the threat intelligence data. We found only 5.2% (4.6% of our labeled data) of our benign websites in the threat intelligence data, which suggests that at most 5.2% of our benign data are mislabeled. Prior research (Zhu and Wu, 2004) has demonstrated that noise can have a negative impact on classification accuracy and increases linearly with noise. A noise level of ˜5% is on the lower end of this study.

Fourth, comparing our results with prior research is challenging. Prior research uses different datasets, features, and performance metrics, collects their data at different times, or focuses on different aspects such as the speed of website classification or other metrics. Prior research may also present various measurements and selecting a measurement on which to compare to is subjective. These factors make direct comparison with prior approaches challenging.

Fifth, we collect features statically from the JavaScript on the webpage. However, malicious JavaScript is often obfuscated, which makes the analysis challenging. Furthermore, static extraction of JavaScript provides less insight than recording features from the actual JavaScript execution. Recording JavaScript execution requires additional overhead and was intentionally omitted to make our approach feasible solely from web `request` features. Although there is some limitation, static JavaScript features have been included in prior research which motivates their inclusion in our study.

8. Conclusions

We performed a comprehensive evaluation of discovering features for malicious website detection. We built two models based on unsupervised learning algorithms and eight based on supervised learning algorithms over various sampling and feature transformation scenarios. Based on our study, we postulate that discovering features (versus selecting features *a priori*) is feasible and performs nearly as well as the features from prior research and does so with fewer features.

Table A – unsupervised confusion matrices.

Autoencoders - 34 Features

Scenario	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
No Sampling	0.8517	400	6275	487	675	0.6769	0.3258	0.3721	0.4510
FT w/ FS	0.9373	693	6653	194	297	0.8693	0.7043	0.7000	0.7813
FT w/PCA	0.8428	787	5818	100	1132	0.8622	0.5337	0.4101	0.8873

9. Credit statement

The following individuals have all contributed to our journal article entitled **"Discovering Features for Detecting Malicious Websites: An Empirical Study"** in the following ways:

Dr. John McGahagan: Conceptualization, Related Work, Methodology, Data Collection, Data Cleaning, Data Analysis, Writing, Model Building.

Darshan Bhansali: Methodology, Data Cleaning, Data Analysis, Model Building, Writing.

Ciro Pinto-Coelho: Reviewing and Editing.

Dr. Michel Cukier: Supervision, Reviewing, and Editing.

10. Additional tables

(Tables A)

One Class SVM - 34 Features

Scenario	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
No Sampling	0.9116	6642	502	308	385	0.7608	0.5429	0.9452	0.9557
FT w/ FS	0.9331	6710	603	240	284	0.8693	0.6598	0.9594	0.9655
FT w/PCA	0.8466	5869	766	1081	121	0.8540	0.5285	0.9798	0.8445

Autoencoders - 99 Features

Scenario	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
No Sampling	0.7884	615	5564	272	1386	0.7470	0.3589	0.3073	0.6933
FT w/ FS	0.9505	627	6822	260	128	0.8442	0.7392	0.8305	0.7069
FT w/PCA	0.8829	696	6223	191	727	0.8400	0.5589	0.4891	0.7847

One Class SVM – 99 Features

Scenario	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
No Sampling	0.8503	6279	385	671	502	0.6688	0.3131	0.9260	0.9035
FT w/ FS	0.9203	6592	620	358	267	0.8237	0.6207	0.9611	0.9485
FT w/PCA	0.8509	6405	264	545	623	0.6096	0.2283	0.9114	0.9216

B

No-sampling 34 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9703	767	6837	133	100	0.9189	0.8516	0.8847	0.8522
RF	0.9796	789	6888	111	49	0.9348	0.8972	0.9415	0.8767
AB	0.9704	761	6844	139	93	0.9161	0.8514	0.8911	0.8456
GB	0.9756	765	6881	135	56	0.9210	0.8765	0.9318	0.8500
ET	0.9787	781	6889	119	48	0.9304	0.8924	0.9421	0.8678
XGB	0.9755	762	6883	138	54	0.9194	0.8757	0.9338	0.8467
BG	0.9770	776	6881	124	56	0.9271	0.8840	0.9327	0.8622
V	0.9802	789	6893	111	44	0.9352	0.9003	0.9472	0.8767

Over-sampling 34 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9421	813	6570	87	367	0.9252	0.7582	0.6890	0.9033
RF	0.9783	797	6870	103	67	0.9379	0.8916	0.9225	0.8856
AB	0.9511	834	6620	66	317	0.9405	0.7935	0.7246	0.9267
GB	0.9633	819	6730	81	207	0.9401	0.8319	0.7982	0.9100
ET	0.9773	794	6865	106	72	0.9359	0.8866	0.9169	0.8822
XGB	0.9622	822	6719	78	218	0.9410	0.8288	0.7904	0.9133
BG	0.9732	782	6845	118	92	0.9278	0.8666	0.8947	0.8689
V	0.9803	785	6898	115	39	0.9333	0.9008	0.9527	0.8722

Over-sampling 34 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9516	799	6659	101	278	0.9239	0.7850	0.7419	0.8878
RF	0.9788	798	6873	102	64	0.9387	0.8941	0.9258	0.8867
AB	0.9564	823	6672	77	265	0.9381	0.8080	0.7564	0.9144
GB	0.9633	819	6730	81	207	0.9401	0.8319	0.7982	0.9100
ET	0.9766	794	6860	106	77	0.9356	0.8837	0.9116	0.8822
XGB	0.9622	822	6719	78	218	0.9410	0.8288	0.7904	0.9133
BG	0.9741	793	6841	107	96	0.9336	0.8719	0.8920	0.8811
V	0.9763	800	6851	100	86	0.9382	0.8825	0.9029	0.8889

Under-sampling 34 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9289	824	6456	76	481	0.9231	0.7242	0.6314	0.9156
RF	0.9570	846	6654	54	283	0.9496	0.8164	0.7493	0.9400
AB	0.9391	844	6516	56	421	0.9385	0.7601	0.6672	0.9378
GB	0.9527	847	6619	53	318	0.9476	0.8024	0.7270	0.9411
ET	0.9590	843	6673	57	264	0.9493	0.8226	0.7615	0.9367
XGB	0.9514	843	6613	57	324	0.9450	0.7970	0.7224	0.9367
BG	0.9515	830	6627	70	310	0.9388	0.7936	0.7281	0.9222
V	0.9803	785	6898	115	39	0.9333	0.9008	0.9527	0.8722

Table B – supervised confusion matrices.

No-sampling 34 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9724	763	6858	137	79	0.9182	0.8611	0.9062	0.8478
RF	0.9796	789	6888	111	49	0.9348	0.8972	0.9415	0.8767
AB	0.9698	755	6845	145	92	0.9128	0.8478	0.8914	0.8389
GB	0.9756	765	6881	135	56	0.9210	0.8765	0.9318	0.8500
ET	0.9788	780	6891	120	46	0.9300	0.8930	0.9443	0.8667
XGB	0.9755	762	6883	138	54	0.9194	0.8757	0.9338	0.8467
BG	0.9761	771	6879	129	58	0.9242	0.8794	0.9300	0.8567
V	0.9803	785	6898	115	39	0.9333	0.9008	0.9527	0.8722

Under-sampling 34 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9329	832	6479	68	458	0.9292	0.7381	0.6450	0.9244
RF	0.9583	849	6661	51	276	0.9518	0.8216	0.7547	0.9433
AB	0.9417	845	6535	55	402	0.9405	0.7679	0.6776	0.9389
GB	0.9510	843	6610	57	327	0.9448	0.7958	0.7205	0.9367
ET	0.9592	846	6671	54	266	0.9508	0.8239	0.7608	0.9400
XGB	0.9514	843	6613	57	324	0.9450	0.7970	0.7224	0.9367
BG	0.9483	843	6589	57	348	0.9433	0.7873	0.7078	0.9367
V	0.9564	843	6652	57	285	0.9478	0.8134	0.7473	0.9367

FT w/FS 34 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9709	751	6858	149	79	0.9115	0.8528	0.9048	0.8344
RF	0.9786	783	6886	117	51	0.9313	0.8919	0.9388	0.8700
AB	0.9682	750	6838	150	99	0.9095	0.8402	0.8834	0.8333
GB	0.9758	759	6888	141	49	0.9181	0.8768	0.9394	0.8433
ET	0.9766	776	6878	124	59	0.9269	0.8822	0.9293	0.8622
XGB	0.9747	753	6886	147	51	0.9147	0.8714	0.9366	0.8367
BG	0.9770	780	6877	120	60	0.9290	0.8843	0.9286	0.8667
V	0.9769	771	6885	129	52	0.9246	0.8831	0.9368	0.8567

FT w/FS 34 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9723	765	6855	135	82	0.9191	0.8607	0.9032	0.8500
RF	0.9779	773	6891	127	46	0.9261	0.8882	0.9438	0.8589
AB	0.9710	749	6861	151	76	0.9106	0.8532	0.9079	0.8322
GB	0.9758	759	6888	141	49	0.9181	0.8768	0.9394	0.8433
ET	0.9781	771	6894	129	43	0.9252	0.8888	0.9472	0.8567
XGB	0.9747	753	6886	147	51	0.9147	0.8714	0.9366	0.8367
BG	0.9774	790	6870	110	67	0.9341	0.8869	0.9218	0.8778
V	0.9783	780	6887	120	50	0.9297	0.8905	0.9398	0.8667

FT w/PCA 34 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9715	742	6872	158	65	0.9075	0.8550	0.9195	0.8244
RF	0.9755	760	6885	140	52	0.9185	0.8756	0.9360	0.8444
AB	0.9680	756	6830	144	107	0.9123	0.8398	0.8760	0.8400
GB	0.9728	754	6870	146	67	0.9141	0.8622	0.9184	0.8378
ET	0.9722	733	6886	167	51	0.9035	0.8576	0.9349	0.8144
XGB	0.9729	751	6874	149	63	0.9127	0.8625	0.9226	0.8344
BG	0.9728	750	6874	150	63	0.9121	0.8618	0.9225	0.8333
V	0.9749	755	6885	145	52	0.9157	0.8722	0.9356	0.8389

FT w/PCA 34 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9718	760	6856	140	81	0.9164	0.8578	0.9037	0.8444
RF	0.9759	756	6892	144	45	0.9168	0.8773	0.9438	0.8400
AB	0.9700	751	6851	149	86	0.9110	0.8486	0.8973	0.8344
GB	0.9728	754	6870	146	67	0.9141	0.8622	0.9184	0.8378
ET	0.9724	733	6888	167	49	0.9037	0.8589	0.9373	0.8144
XGB	0.9729	751	6874	149	63	0.9127	0.8625	0.9226	0.8344
BG	0.9744	767	6869	133	68	0.9212	0.8705	0.9186	0.8522
V	0.9751	763	6879	137	58	0.9197	0.8739	0.9294	0.8478

No-sampling 99 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9744	734	6902	166	35	0.9053	0.8687	0.9545	0.8156
RF	0.9801	778	6903	122	34	0.9298	0.8992	0.9581	0.8644
AB	0.9707	760	6847	140	90	0.9157	0.8525	0.8941	0.8444
GB	0.9756	763	6883	137	54	0.9200	0.8764	0.9339	0.8478
ET	0.9786	759	6910	141	27	0.9197	0.8910	0.9656	0.8433
XGB	0.9755	763	6882	137	55	0.9199	0.8758	0.9328	0.8478
BG	0.9784	775	6893	125	44	0.9274	0.8909	0.9463	0.8611
V	0.9803	774	6909	126	28	0.9280	0.9004	0.9651	0.8600

No-sampling 99 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9763	753	6898	147	39	0.9155	0.8791	0.9508	0.8367
RF	0.9797	778	6900	122	37	0.9296	0.8973	0.9546	0.8644
AB	0.9718	760	6856	140	81	0.9164	0.8578	0.9037	0.8444
GB	0.9756	763	6883	137	54	0.9200	0.8764	0.9339	0.8478
ET	0.9787	759	6911	141	26	0.9198	0.8916	0.9669	0.8433
XGB	0.9755	763	6882	137	55	0.9199	0.8758	0.9328	0.8478
BG	0.9784	779	6889	121	48	0.9293	0.8911	0.9420	0.8656
V	0.9798	777	6902	123	35	0.9291	0.8979	0.9569	0.8633

Over-sampling 99 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Prec	Rec
KNN	0.9376	833	6515	67	422	0.9324	0.7518	0.6637	0.9256
RF	0.9782	781	6885	119	52	0.9301	0.8899	0.9376	0.8678
AB	0.9552	793	6693	107	244	0.9230	0.7960	0.7647	0.8811
GB	0.9695	787	6811	113	126	0.9281	0.8510	0.8620	0.8744
ET	0.9798	782	6897	118	40	0.9316	0.8981	0.9513	0.8689
XGB	0.9692	785	6811	115	126	0.9270	0.8496	0.8617	0.8722
BG	0.9746	777	6861	123	76	0.9262	0.8726	0.9109	0.8633
V	0.9803	774	6909	126	28	0.9280	0.9004	0.9651	0.8600

Over-sampling 99 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Rec	Prec
KNN	0.9446	830	6573	70	364	0.9349	0.7716	0.6951	0.9222
RF	0.9782	781	6885	119	52	0.9301	0.8899	0.9376	0.8678
AB	0.9631	781	6767	119	170	0.9216	0.8234	0.8212	0.8678
GB	0.9695	787	6811	113	126	0.9281	0.8510	0.8620	0.8744
ET	0.9802	786	6896	114	41	0.9337	0.9002	0.9504	0.8733
XGB	0.9692	785	6811	115	126	0.9270	0.8496	0.8617	0.8722
BG	0.9759	787	6861	113	76	0.9317	0.8795	0.9119	0.8744
V	0.9789	806	6866	94	71	0.9427	0.8954	0.9190	0.8956

Under-sampling 99 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Rec	Prec
KNN	0.9470	814	6608	86	329	0.9285	0.7742	0.7122	0.9044
RF	0.9594	845	6674	55	263	0.9505	0.8245	0.7626	0.9389
AB	0.9376	843	6505	57	432	0.9372	0.7553	0.6612	0.9367
GB	0.9524	845	6619	55	318	0.9465	0.8009	0.7266	0.9389
ET	0.9579	839	6668	61	269	0.9467	0.8176	0.7572	0.9322
XGB	0.9525	841	6624	59	313	0.9447	0.8002	0.7288	0.9344
BG	0.9570	833	6667	67	270	0.9433	0.8129	0.7552	0.9256
V	0.9803	774	6909	126	28	0.9280	0.9004	0.9651	0.8600

Under-sampling 99 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Rec	Prec
KNN	0.9525	810	6655	90	282	0.9297	0.7912	0.7418	0.9000
RF	0.9547	846	6636	54	301	0.9483	0.8088	0.7376	0.9400
AB	0.9389	841	6517	59	420	0.9369	0.7583	0.6669	0.9344
GB	0.9524	845	6619	55	318	0.9465	0.8009	0.7266	0.9389
ET	0.9599	844	6679	56	258	0.9503	0.8260	0.7659	0.9378
XGB	0.9525	841	6624	59	313	0.9447	0.8002	0.7288	0.9344
BG	0.9537	845	6629	55	308	0.9472	0.8051	0.7329	0.9389
V	0.9635	839	6712	61	225	0.9499	0.8375	0.7885	0.9322

FT w/FS 99 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Rec	Prec
KNN	0.9721	746	6872	154	65	0.9098	0.8578	0.9199	0.8289
RF	0.9786	776	6893	124	44	0.9279	0.8915	0.9463	0.8622
AB	0.9668	717	6860	183	77	0.8928	0.8300	0.9030	0.7967
GB	0.9737	750	6881	150	56	0.9126	0.8662	0.9305	0.8333
ET	0.9772	772	6886	128	51	0.9252	0.8844	0.9380	0.8578
XGB	0.9740	742	6891	158	46	0.9089	0.8670	0.9416	0.8244
BG	0.9751	759	6883	141	54	0.9178	0.8737	0.9336	0.8433
V	0.9773	769	6890	131	47	0.9238	0.8849	0.9424	0.8544

FT w/FS 99 Features - Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Rec	Prec
KNN	0.9717	755	6860	145	77	0.9139	0.8568	0.9075	0.8389
RF	0.9778	775	6888	125	49	0.9270	0.8877	0.9405	0.8611
AB	0.9698	740	6860	160	77	0.9056	0.8463	0.9058	0.8222
GB	0.9737	750	6881	150	56	0.9126	0.8662	0.9305	0.8333
ET	0.9770	775	6882	125	55	0.9266	0.8840	0.9337	0.8611
XGB	0.9740	742	6891	158	46	0.9089	0.8670	0.9416	0.8244
BG	0.9759	768	6880	132	57	0.9226	0.8780	0.9309	0.8533
V	0.9781	772	6893	128	44	0.9257	0.8888	0.9461	0.8578

FT w/PCA 99 Features - Default Parameters

Model	ACC	TP	TN	FN	FP	AUC	MCC	Rec	Prec
KNN	0.9663	648	6925	252	12	0.8591	0.8246	0.9818	0.7200
RF	0.9685	678	6912	222	25	0.8749	0.8365	0.9644	0.7533
AB	0.9666	739	6836	161	101	0.9033	0.8313	0.8798	0.8211
GB	0.9686	730	6861	170	76	0.9001	0.8399	0.9057	0.8111
ET	0.9497	508	6935	392	2	0.7821	0.7292	0.9961	0.5644
XGB	0.9710	735	6875	165	62	0.9039	0.8520	0.9222	0.8167
BG	0.9685	730	6860	170	77	0.9000	0.8392	0.9046	0.8111
V	0.9723	705	6915	195	22	0.8901	0.8574	0.9697	0.7833

FT w/PCA 99 Features -Tuned

Model	ACC	TP	TN	FN	FP	AUC	MCC	Rec	Prec
KNN	0.9695	679	6919	221	18	0.8759	0.8421	0.9742	0.7544
RF	0.9684	677	6912	223	25	0.8743	0.8358	0.9644	0.7522
AB	0.9677	744	6840	156	97	0.9063	0.8372	0.8847	0.8267
GB	0.9686	730	6861	170	76	0.9001	0.8399	0.9057	0.8111
ET	0.9515	522	6935	378	2	0.7899	0.7400	0.9962	0.5800
XGB	0.9710	735	6875	165	62	0.9039	0.8520	0.9222	0.8167
BG	0.9696	750	6849	150	88	0.9103	0.8467	0.8950	0.8333
V	0.9741	720	6914	180	23	0.8983	0.8671	0.9690	0.8000

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: We acknowledge the support of the U.S. Department of Defense. The views and conclusions expressed in this paper are those of the authors and do not necessarily represent those of the Department of Defense or U.S. Government. We thank the Cisco Talos Intelligence Group for having created and shared the list of malicious websites used in this study.

Acknowledgements

We thank the Cisco Talos Intelligence Group for having created and shared the list of 7039 malicious websites. We also acknowledge the support of the U.S. Department of Defense. The views and conclusions expressed in this paper are those of the authors and do not necessarily represent those of the Department of Defense or U.S. Government.

REFERENCES

3sharp, 2019. [Online]. Available: http://www.3sharp.com/projects/antiphishing/. [Accessed September. 19, 2019].

Ahluwalia ITraore, Ganame K, Agarwal N. Detecting broad length algorithmically generated domains. In: Proc. International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments; 2017. p. 19–34.

Amazon.com, "The top 500 sites on the web," [Online]. Available: https://www.alexa.com/topsite/. [Accessed August. 2018].

Apache, 2019, "Apache web server bug grants root access on shared hosting environments," [Online]. Available: https://www.zdnet.com/article/apache-web-server-bug-grants-root-access-on-shared-hosting-environme [Accessed March. 31, 2020].

Baldi P. Autoencoders, unsupervised learning, and deep architectures. In: Proc. ICML Workshop on Unsupervised and Transfer Learning; 2012. p. 37–49.

Basnet RB, Sung AH, Liu Q. Feature selection for improved phishing detection. In: Proc. International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems; 2012. p. 252–61.

Beigi EB, Jazi HH, Stakhanova N, Ghorbani AA. Towards effective feature selection in machine learning-based botnet detection approaches. In: Proc. 2014 IEEE Conference on Communications and Network Security; 2014. p. 247–55.

Bishop CM. Neural Networks for Pattern Recognition. New York, NY: Oxford University Press; 1995.

Bloomberg. Ponemon Institute Reveals Security Teams Spend Approximately 25 Percent of Their Time Chasing False Positives. Response Times 2019. [Online]. Available https://www.bloomberg.com/press-releases/2019-08-01/ponemon.

Bottou L, LeCun Y. Large scale online learning. In: Proc. Advances in Neural Information Processing Systems; 2004. p. 217–24.

Breiman L. Random forests. Machine Learning 2001;45(1):5–32 Oct.

Breiman L. Bagging predictors. Machine Learning 1996;24(2):123–40 Aug.

Butkiewicz M, Madhyastha HV, Sekar V. Understanding website complexity: measurements, metrics, and implications. In: Proc. of the 2011 ACM SIGCOMM Conference on Internet Measurement; 2011. p. 313–28.

Canali D, Cova M, Vigna G, Kruegel C. Prophiler: a fast filter for the large-scale detection of malicious web pages. In: Proc. 20th International Conference on World wide web ACM; 2010. p. 197–206.

Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. J. Artificial Intelligence Res. 2002;16:321–57 Jun.

Avanan, 2019, "MetaMorph HTML Obfuscation Phishing Attack," [Online]. Available: https://www.avanan.com/blog/metamorph-html-obfuscation-phishing-attack. [Accessed March. 31, 2020].

ArsTechnica, 2013, "Rampant Apache website attack hits visitors with highly malicious software," [Online]. Available: https://arstechnica.com/information-technology/2013/07/darkleech-infects-40k-apache-site-addresses/. [Accessed March. 31, 2020].

Cisco Talos Intelligence Group, 2021 [Online]. Available: https://talosintelligence.com/

Cova M, Kruegel C, Vigna G. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In: Proc. 19th International Conference on World Wide Web; 2010. p. 281–90.

Cymon.io, "Open Threat Intelligence," 2019. [Online]. Available: https://cymon.io/. [Accessed January. 15, 2019].

Curtsinger C, Livshits B, Zorn BG, Seifert C. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In: Proc. 2011 USENIX Security Symposium. USENIX; 2011. p. 33–48.

Dietterich TG. Ensemble methods in Machine Learning. In: Proc. International Workshop on Multiple Classifier Systems; 2000. p. 1–15.

Durumeric Z, Adrian D, Mirian A, Bailey M, Halderman JA. A Search Engine Backed by Internet-Wide Scanning. In: Proc. 22nd ACM Conference on Computer and Communications Security; 2015. p. 542–53.
Eshete B, Villafiorita A, Weldemariam K. Binspect: Holistic analysis and detection of malicious web pages. In: Proc. International Conference on Security and Privacy in Communication Systems; 2012. p. 149–66.
Featuretools, "An open source Python framework for automated feature engineering," [Online]. Available: https://www.featuretools.com/. [Accessed February. 8, 2019].
Friedman J, Hastie T, Tibshirani R. *The Elements of Statistical* Learning. New York, NY: Springer Series in Statistics; 2009.
Friedman JH. Stochastic gradient boosting. Computat. Statistic. Data Analysis 2002;38(4):367–78 Feb.
Geurts P, Damien E, Wehenkel L. Extremely randomized trees. Machine Learning 2006;63:3–42 no.1Apr.
Github, "English-words," [Online]. Available: https://github.com/dwyl/english-words/. [Accessed August. 1, 2018].
Guang X, Jason O, RCarolyn P, Lorrie C. CANTINA+: A feature-rich machine learning framework for detecting phishing websites. In: Proc. ACM Transactions on Information and System Security; 2011. p. 1–28.
He M, Horng SJ, Fan P, Khan MK, Run RS, Lai JL, Chen RJ, Sutanto A. An efficient phishing webpage detector. Expert Syst. Appl. 2011;38(10):12018–27.
James G, Witten D, Hastie T, Tibshirani R. An Introduction to Statistical Learning. New York: springer; 2013.
Kapravelos YShoshitaishvili, Cova M, Kruegel C, Vigna G. Revolver: an automated approach to the detection of evasive web-based malware, 14–16; 2013. p. 637–52 Aug.
M E, Karabulut SA Özel, Ibrikci T. A comparative study on the effect of feature selection on classification accuracy. Procedia Technol. 2012;1:323–7.
Info Security, 2013, "Malicious Apache server and Blackhold provide stealthy backdoor," [Online]. Available: https://www.infosecurity-magazine.com/news/malicious-apache-server-and-blackhole-provide /[Accessed March. 31, 2020].
M. Gualtieri, 2018, "Stealing Data with CSS: Attack and Defense," [Online]. Available: https://www.mike-gualtieri.com/posts/stealing-data-with-css-attack-and-defense /[Accessed March. 31, 2020].
B. Keating, "The frequency of the letters of the alphabet in English," [Online]. Available: https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html. [Accessed March. 31, 2020].
Keras, 2020. [Online]. Available: https://keras.io /[Accessed March 1, 2020].
Kheir N, Blanc G, Debar H, Garcia-Alfaro J, Yang D. Automated classification of C&C connections through malware URL clustering. In: Proc. ICT Systems Security and Privacy Protection; 2015. p. 252–66.
Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proc. 14th International Joint Conference on Artificial Intelligence; 1995. p. 1137–45.
Li T, Kou G, Peng Y. Improving malicious URLs detection via feature engineering: linear and nonlinear space transformation methods. Information Systems 2020(91).
Ma J, Saul LK, Savage S, Voelker GM. Beyond blacklists: learning to detect malicious websites from suspicious URLs. In: Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2009. p. 1245–54.
Ma J, Saul LK, Savage S, Voelker GM. Identifying suspicious urls: An application of large-scale online learning. In: Proc. International Conference on Machine Learning; 2009. p. 681–8.
Ma J, Saul LK, Savage S, Voelker GM. Learning to detect malicious urls. ACM Trans. Intell. Syst. Technol. 2011;2:30 no.3.
Marchal S, Saari K, Singh N, Asokan N. Know your phish: Novel techniques for detecting phishing sites and their targets. In: IEEE 36th International Conference on Distributed Computing Systems (ICDCS); 2016. p. 323–33.
McGahagan IV J, Bhansali D, Gratian M, Cukier M. A Comprehensive Evaluation of HTTP header features for detecting malicious websites. In: Proc 2019 15th European Dependable Computing Conference (EDCC); 2019. p. 75–82.
McGahagan J, Bhansali D, Pinto-Coelho C, Cukier M. A comprehensive evaluation of webpage content features for detecting malicious websites. In: Proc. 2019 9th Latin-American Symposium on Dependable Computing (LADC); 2019. p. 1–10.
McGrath DK, Gupta M. Behind phishing: an examination of phisher Modi operandi. Proc. Of USENIX, 2008.
McCullagh P, Nelder JA. Generalized Linear Models. Boca Raton, FL: Chapman & Hall; 1989.
MalwareBytes Labs, 2016, "Explained: Domain Generating Algorithms." [Online]. Available: https://blog.malwarebytes.com/security-world/2016/12/explained-domain-generating-algorithm/. [Accessed September. 19, 2019].
MDN web docs, "HTTP headers," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers /[Accessed March. 31, 2020].
MDN web docs, "JavaScript reference," https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference. [Accessed August 10, 2018].
Murphy KP. In: Naïve Bayes Classifiers. University of British Columbia; 2006. p. 60.
Nelms T, Perdisci R, Ahamad M. Execscent: mining for new c&c domains in live networks with adaptive control protocol templates. In: Proc. 22nd USENIX Security Symposium; 2013. p. 589–604.
Niakanlahiji BTChu, Al-Shaer E. PhishMon: a machine learning framework for detecting phishing webpages. In: Proc. 2018 IEEE International Conference on Intelligence and Security Informatics (ISI); 2018. p. 220–5.
Open DNS, "PhishTank. Out of the Net, into the Tank" 2019. [Online]. Available: https://www.phishtank.com/. [Accessed April. 8, 2019].
OWASP, "Cache Poisoning," [Online]. Available: https://owasp.org/www-community/attacks/Cache_Poisoning. [Accessed March. 31, 2020].
Perdisci R, Davide A, Giacinto G. Scalable fine-grained behavioral clustering of http-based malware. Computer Networks 2013;57(2):487–500.
Perdisci R, Lee W, Feamster N. Behavioral clustering of http-based malware and signature generation using malicious network traces. In: Proc. NSDI'10; 2010. p. 391–404.
Peterson LE. K-nearest neighbor. Scholarpedia 2009;4(2):1883. Available: Scholarpedia.org http://www.scholarpedia.org/article/K-nearest_neighbor. [Accessed August 1, 2018].
Pixabay "Stunning free images and royalty free stock." Available: https://pixabay.com/. [Accessed: Sept. 19, 2019]
N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, "All your iframes point to us," 2008.
Pyselenium, 2018. [Online]. https://pypi.org/project/PySelenium/. [Accessed August 10, 2018].
Raileanu LE, Stoffel K. Theoretical comparison between the gini index and information gain criteria. Ann. Mathemat. Artific. Intell. 2004;41(no.1):77–93 vo..
K. Reitz, I. Cordasco, and N. Prewitt, "Requests: HTTP for Humans," 2019. [Online]. Available:

http://docs.Python-requests.org/en/master/. [Accessed August. 10, 2018].
Rieck K, Krueger T, Dewald A. Cujo: efficient detection and prevention of drive-by download attacks. In: Proc. 26th Annual Computer Security Applications Conference; 2010. p. 31–9.
Ruta D, Gabrys B. Classifier selection for majority voting. Inf. Fusion 2005;6:63–81 vol.no.1.
Schapire RE. A brief introduction to boosting, 2; 1999. p. 1401–6.
Scikit-Learn, "Machine Learning in Python," [Online]. Available: https://scikit-learn.org/stable/. [Accessed March. 31, 2020].
Security Week, 2018, "Phishers Use New Method to Bypass Office 365 Safe Links," [Online]. Available: https://www.securityweek.com/phishers-use-new-method-bypass-office-365-safe-links. [Accessed March. 31, 2020]
Syarif APrugel-Bennett, Wills G. SVM parameter optimization using grid search and genetic algorithm to improve classification performance. Telkomnika 2016;14(4):1502–9.
SMOTE - Synthetic Minority Over-sampling Technique, 2019, [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html. [Accessed February. 8, 2019].
SiteGround, 2017, "How The Vary HTTP Header Can Be Bad," [Online]. Available: https://www.siteground.com/blog/vary-http-header/. [Accessed March. 31, 2020].
Security Boulevard, 2019, "Malicious Bot Detection Through a Complex Proxy Network," [Online]. Available: https://securityboulevard.com/2019/04/malicious-bot-detection-through-a-complex-proxy-network/. [Accessed March. 31, 2020].
The Artificial Imposter, "Feature Importance Measures for Tree Models." [Online]. Available https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3. [Accessed September. 19, 2019].
S. Ullrich, 2013, "HTTP Evasions Explained," [Online]. Available: https://noxxi.de/research/. [Accessed August. 7, 2020].
W3School, "HTML5 Introduction," [Online]. Available: https://www.w3schools.com/html/html5_intro.asp. [Accessed March. 31, 2020].
Whittaker C, Ryner B, Nazif M. Large-scale automatic classification of phishing pages. Proc. of NDSS '10, 2010.
Willard G. Understanding the co-evolution of cyber defenses and attacks to achieve enhanced cybersecurity. J. Inf. Warfare 2015;14:17–31.
XGBoost, 2019. [Online]. https://xgboost.readthedocs.io/en/latest/. [Accessed June. 20, 2019].
Xu L, Zhan Z, Xu S, Ye K. Cross-layer detection of malicious websites. In: Proc. Third ACM conference on Data and Application Security and Privacy; 2013. p. 141–52.
Yadav S, Reddy AKK, Reddy AN, Ranjan S. Detecting algorithmically generated malicious domain names. In: Proc. 10th ACM SIGCOMM Conference on Internet Measurement; 2010. p. 48–61.
Yan X, Xu Y, Cui B, Zhang S, Guo T, Li C. Learning url embedding for malicious website detection. IEEE Trans. Ind. Inf. 2020;16:6673–81 no.10.
Zaborowski EJ. Malicious Proxies. Defcon 2017. [Online]. Available https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-edward_zaborowski-doppelganger.pdf. [Accessed March. 31, 2020].
ZdNet, 2013, "'Sophisticated backdoor malware opens up seruvriy balckhole in Apache webservers," [Online]. Available: https://www.zdnet.com/article/sophisticated-backdoor-malware-opens-up-security-blackhole-in-apache-web-servers [Accessed March. 31, 2020].
Zarras APapadogiannakis, Gawlik R, Holz T. Automated generation of models for fast and precise detection of HTTP-based malware. In: Proc. 12th Annual International Conference on Privacy, Security and Trust; 2014. p. 249–56.
Zhang J, Seifert C, Stokes JW, Lee W. Arrow: Generating signatures to detect drive-by downloads. In: Proc. 20th International Conference on World Wide Web, WWW '11; 2011. p. 187–96.
Zhang Y, Hong JI, Cranor LF. Cantina: a content-based approach to detecting phishing websites. In: Proc. 16th International Conference on World Wide Web; 2007. p. 639–48.
Zhou Y, GCheng JShanqing, Dai M. Building an efficient intrusion detection system based on feature selection and ensemble classifier. Computer Networks 2020;174.
Zhu X, Wu X. Class noise vs. attribute noise: a quantitative study. Artificial Intelligence Review 2004;22(no.3):177–210.
The Honeypot Project, 2009, "Capture HPC," [Online]. Available: https://www.honeynet.org/projects/old/capture-hpc/. [Accessed March. 31, 2020].

John McGahagan is a software engineer with a PhD in Reliability Engineering from the University of Maryland, College Park. He has an MS Systems Engineering and BS Electrical Engineering from the University of Maryland, College Park with a focus on computer security and software systems. His research concentration includes detecting cyber threats through machine learning and statistics.

Darshan Bhansali currently works at Purdue University as a Data Scientist. He assists the University in deriving actionable insights from huge amounts of student data. Current projects include designing and maintaining a methodology to sample students for testing Covid-19. Prior to that, he worked with researchers at University of Maryland on a wide range of topics revolving around integrating Data Science in Cybersecurity.

Ciro Pinto-Coelho is a graduate student pursuing a PhD in Reliability Engineering at the University of Maryland, College Park. He has a BS degree in Biochemistry and a MS degree in Chemical Engineering, both from the University of Maryland. His research interests are in the field of social cybersecurity with a focus on the susceptibility of online communities to influence.

Michel Cukier is a professor of reliability engineering with a joint appointment in the Department of Mechanical Engineering at the University of Maryland, College Park. He is also the director for the Advanced Cybersecurity Experience for Students (ACES). His research covers dependability and security issues. Dr. Cukier has published more than 70 papers in journals and refereed conference proceedings in those areas