CM30082: Dissertation

Large-Scale Lexical Classification of Phishing Websites

Author
David Medzinskii

Supervisor
Dr. Julian Padget

Second Marker
Dr. Russell Bradford

Course
BSc. (Hons) Computer Science
The University of Bath

May 2, 2017

Access

This dissertation may be made available for consultation within the University Library, and may be photocopied or lent to other libraries for the purposes of consultation.

Large-Scale Lexical Classification of Phishing Websites

Submitted by: David Medzinskii

DECLARATION
This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Cover sheet

This page is replaced by the departmental cover sheet in the bound version of the document.

Abstract

The prominence of phishing has risen over the past years, with the number of unique attacks reaching an all time high in 2016. Attacks can be deployed with minimal cost and effort, enabling attackers to launch large volumes of attacks in short spaces of time. The fast-paced nature of phishing makes automated detection processes critical for the safe-guarding of Internet users.

This study investigates the use of machine learning for phishing detection, with features extracted from the URL only. Through experimentation, a set of 87 effective features were identified, including a significant number of novel features not found in existing research. An evaluation of classification algorithms identified that a Random Forest model with 150 trees maximized classification performance, obtaining an F1 score of 0.92 and ROC AUC of 0.97 when testing on a noisy data set of URLs obtained from spam email - a major communication channel where phishing attacks are found. A comparison against existing research indicated that the model built in this study outperforms state-of-the-art lexical classifiers, and often outperforms classifiers that use external features too.

The obtained results were used to build a large-scale lexical classifier, Poseidon, that is able to accelerate the classification of phishing sites, reducing the load on a more expensive classification process by 99%. It is shown that Poseidon outperforms existing systems of this nature with respect to various evaluation metrics. Testing on a live feed of 2 million unlabelled URLs/day, Poseidon is able to detect 6000 phishing attacks/month, costing $0.01 per true positive when using a mainstream cloud services provider.

This study is one of the few to evaluate classification in a real-life scenario, using phishing and benign URLs retrieved from an environment in which a large proportion of phishing attacks operate.

Contents

List of Figures

Acknowledgements

I would like to thank Dr. Julian Padget for his invaluable guidance and encouragement throughout this project - it has been a great pleasure. I would also like to thank the individuals at Netcraft (especially Mike, Andy, Matt) who have provided me with the extensive insight, support and resources to make this project feasible.

1 Introduction

Phishing is a lucrative crime. For a mere $20, one can obtain the resources necessary to deploy a phishing attack with minimal human intervention (Ramzan, 2010), enabling the collection of sensitive information (passwords, bank-details, credit-card information) from those who fall victim. While some of the stolen credentials can facilitate direct financial theft by the attacker, they are also a highly-demanded commodity in the underground, with particular credentials being bought in the region of $1000 (Ramzan, 2010).

2016 saw phishing reach an all-time high (Anti-Phishing Working Group, 2016), with the number of unique phishing sites increasing by 300% from Q4 2015. The rising volume in phishing attacks indicates the need for intelligent systems to protect individuals. Existing measures of phishing mitigation rest on the shoulders of effective detection mechanisms - a constantly evolving field due to novel approaches introduced by attackers in attempts to evade detection.

User-facing applications such as web-browsers and email-clients have built in mechanisms to shield users from phishing attacks, though these rely on the principle of having a blacklist of known phishing URLs. Reliable detection processes are essential for ensuring blacklists are frequently maintained, which is critical due to the fast-paced nature of phishing attacks, with individual phishing attacks often only lasting for a few hours (Sheng et al., 2009).

It has been shown that automated detection through the use of machine learning is effective for the purposes of maintaining blacklists (Whittaker et al., 2010). However, a large proportion of these systems pose caveats due to the method of data retrieval. Accessing potential phishing websites in order to perform classification introduces increased latency and high-memory usage, as well as providing the attacker with the opportunity to evade the systems by masking their attacks (Le et al., 2011). While these limitations can be overcome, this incurs an economic cost that can make processing large feeds of websites infeasible.

A possible solution to this is to perform detection purely using information extracted from the URL, without the dependency of external data retrieval. Research on lexical classification of phishing URLs is limited, though experiments have shown that lexical classification can be fast and reliable (Blum et al., 2010, Darling et al., 2015, Gyawali et al., 2011), with minimal reduction in predictive performance (Le et al., 2011).

A lightweight classifier also has uses as a filter for a more expensive classification process, with existing systems enabling up to 90% load reduction (Gyawali et al., 2011), allowing large-volume feeds to be processed by complex classifiers. This can subsequently support the frequent updating of blacklists, aiding applications to effectively safeguard users from phishing attacks, as well as enable take-down efforts to be launched quicker, killing phishing campaigns earlier in their life cycle.

Unfortunately, there is significant disparity amongst the recommended features and classification algorithms in existing literature, highlighting the need for an investigation to take place to evaluate these under the same conditions and constraints. The results of this can inform the implementation of a classification system that can be used to tackle the increasingly growing number of phishing attacks.

1.1 Aims and Contributions

This study consists of three distinct phases. The initial phase involves evaluating lexical features identified by existing studies, as well evaluating a significant number of novel features to identify an optimal feature vector.

The second phase of the study involves identifying the optimal classification algorithm for this domain, producing a classification model that is subsequently compared against existing studies. An investigation is also performed to observe the impact of the evolution of phishing attacks on classification performance of the produced model.

The results of the first two phase of the study are used to build and evaluate a large-scale classification system, which processes a live feed of spam URLs obtained from Netcraft (Netcraft, 2017*b*).

Existing research in lexical-only classification is limited, with little evidence of investigations undertaken in real-life scenarios. This study aims to improve upon this by evaluating the classification system against live phishing attacks, in a real-time environment that exhibits the true conditions and constraints under which a system of this nature would be used.

2 Literature and Technology Survey

Phishing is a method by which confidential or sensitive data is stolen through "both *social engineering* and *technical subterfuge*" (Abu-Nimeh et al., 2007, Abunadi et al., 2013, Anti-Phishing Working Group, 2016). Through various communication channels (email, IM, social media and more (Vasava and Mangrule, 2015)), the attacker will lure the victim to a website that mimics a website of a legitimate organisation (Likarish et al., 2008), and present input fields that request the victim's confidential details, e.g. login details for an individual's online banking account. Data entered by the user will then be stored, and once collected by the attacker, may be used to commit fraud or sold to a bidder (Ramanathan and Wechsler, 2012).

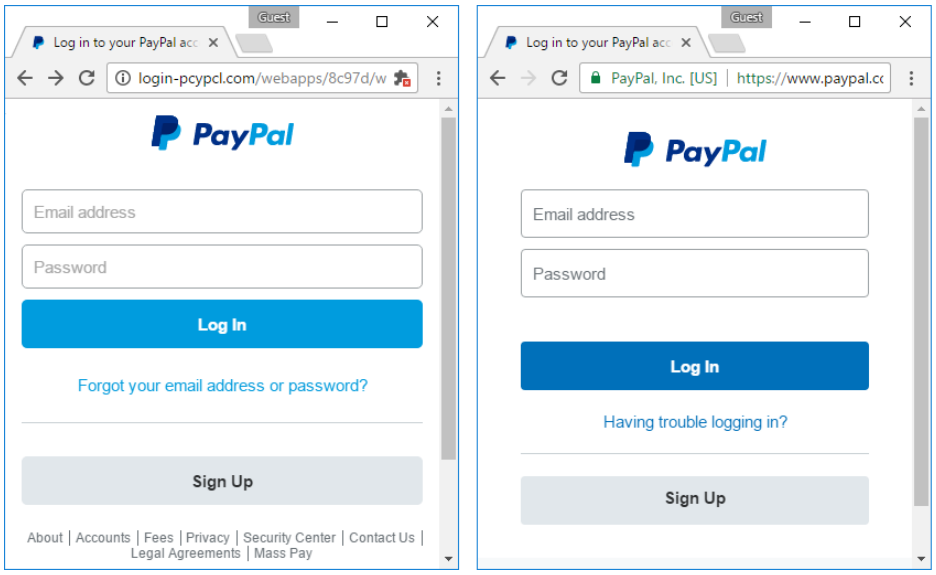

Figure 2.1: An example of a phishing page (left) mimicking a real PayPal (PayPal, 2017) login page (right).

Figure 2.1 shows the degree of deceptiveness possessed by phishing sites. The phishing site on the left is almost structurally identical to the real login page (right). Submitting the form on the phishing site will store the entered credentials on the server, and the victim will be presented with a fake page indicating an issue with their PayPal account. In this instance, the attacker has employed several methods to create an illusion of legitimacy:

- Using elements which are visually similar to the legitimate page.
- Using a domain (`login-pcypcl.com`) with visually similar characters of the target brand substituted. In this case, the `a` in `paypal` is switched to a `c`.
- Using graphical assets that are sourced from the target brand's web-server. The PayPal logo in this case is loaded from one of PayPal's webservers.

While indicators to detect phishing exist (the misspelled brand name in the URL, invalid anchor links within the page, lack of HTTPS indicator), many users will ignore these cues when presented with a phishing site. Links to phishing sites will also often be accompanied with content that is alarming in nature, e.g. informing the user that failing to submit their details will lead to the suspension of their bank account (Ludl et al., 2007). Under the pressure of the warning, the user may overlook certain features that could help indicate that the site is a phishing site, and therefore fall victim to the scheme.

With the ease of acquiring large email lists on the black market (Stringhini et al., 2014, Vömel et al., 2010), accessibility to users' social media profiles and other communication channels, phishing attacks can reach a huge audience. Vömel et al. (2010), for example, found the sale of an email list containing 5 million target email addresses through their investigation of criminal services offered online. This indicates the importance of anti-phishing tools in safe-guarding potential victims from financial loss, theft of confidential material, industrial espionage and more (Weider et al., 2008).

2.1 Tackling Phishing

Approaches to tackling phishing fall into four categories: *Detection*, *Offensive Defence*, *Correction* and *Prevention* (Abu-Nimeh et al., 2007, Balusamy et al., 2016, Khonji et al., 2013). Through these approaches, the effects of phishing campaigns can be mitigated at difference stages of the campaign lifecycle, and ultimately lead to the end of the campaign. Figure 2.2 visualizes the process by which anti-phishing techniques kill phishing campaigns.

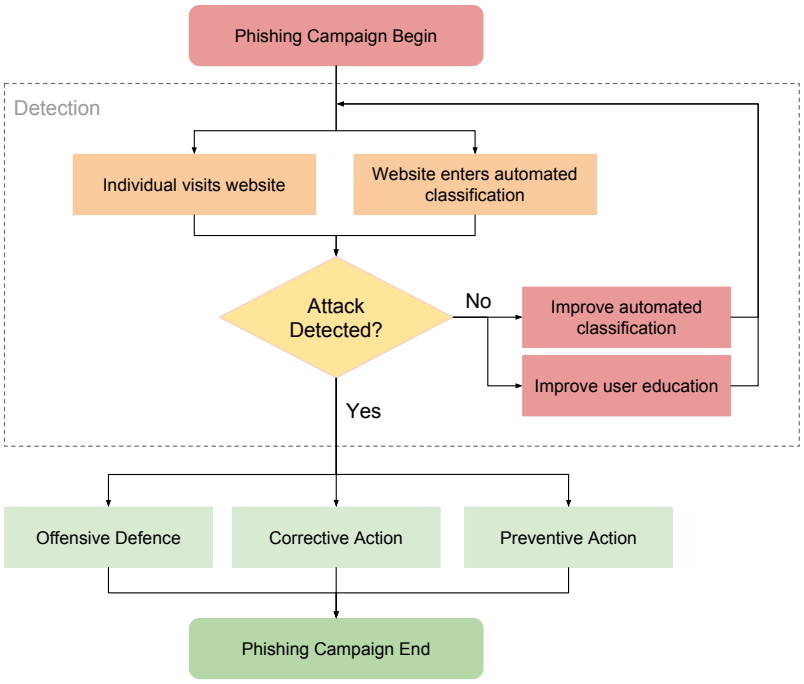

Figure 2.2: Process by which anti-phishing techniques bring phishing campaigns to an end. Based on the process outlined by Khonji et al. (2013).

Through an analysis of the length of phishing campaigns, Sheng et al. (2009) observed that 63% of phishing campaigns in their data set lasted for only 2 hours, however only 7.9% of these were detected and had corrective measures put into place. Attackers will often undertake short campaigns in order to avoid detection, since a shorter campaign reduces the exposure of the phishing website to existing detection tools and diminishes the advantages provided by blacklists (Blum et al., 2010).

2.2 Detection

The detection phase is the first process involved in weakening and mitigating a phishing campaign, which highlights its importance since further downstream process rely on the success of detection. Methods of detecting phishing sites generally fall into 2 distinct categories: User Training and Automated Classification (Khonji et al., 2013).

2.2.1 User Training

A phishing website is usually first visited by a human, since the link is distributed via communication channels such as email and social media. Since the phishing website will not exist in any blacklists at that point in time, the classification process carried out by the human is therefore the only barrier that can prevent the human from disclosing their personal details. Sheng et al. (2010) identified that user education does in fact lead to the user being "less susceptible" to phishing attacks. Through exposure to various phishing-education resources (web-based materials, cartoons), users were found to be 40% less likely to divulge their confidential details on a phishing site. Robila and Ragucci (2006) found that their method of user-education for improving phishing identification was successful, and that their participants in the study "were able to identify most of the threats."

To further support this, Kumaraguru et al. (2007) carried out a study to identify the effectiveness of different methods of phishing-education (security-notices, comic strips and a poster). The comic-strip proved to be most effective, with only 23% of participants falling for a phishing attack post-education (100% fell for the phishing attack pre-education). There is, however, an issue of credibility in this particular study as it only involved 30 participants. While there were efforts to diversify the demographics, the study did not ensure that each group for each method of education had the same gender split, and mean age. Sheng et al. (2010) expresses the fact that both gender and age can determine the effectiveness of user-education, therefore these results may not be as reliable.

User-education is not, however, regarded as the *complete* solution to being protected from phishing. Görling (2006) explains that a user's behaviour in a particular situation (within the context of computer-security) is dependent on much more than solely the "education" that they have received. The real challenge lies with ensuring that the user applies their knowledge, the likelihood of which is often impacted due to security being a "secondary task"(Görling, 2006), when compared to the actual task the user is undertaking. A user may choose to ignore certain cues in order to ensure that the primary task is completed, and therefore leave themselves vulnerable to attack despite having received education.

Similarily, in the study by Robila and Ragucci (2006), it is highlighted that many attacks are successful as they appeal to the user emotionally. Despite the success achieved by Robila and Ragucci (2006) through their methods of education, the manipulative nature of phishing attacks can prevent a user applying their phishing-education effectively, thus leaving them susceptible to losing their confidential information.

2.2.2 Automated Classification

The drawbacks of user-education can be tackled through the use of preventative tools that make use of automated classification. These tools are discussed further in Section 2.5, however the majority will perform either real-time detection using heuristics, such as *SpoofGuard* (Stanford Security Lab, 1998), use a blacklist, or a combination of both (Zhang et al., 2007). Regardless of the time of detection (real-time versus relying on a blacklist of phishing sites detected in the past), an automated classification process is involved to perform the detection. Performing detection through software is advantageous since human error is removed as a factor from the effectiveness of detection (Khonji et al., 2013), and is less vulnerable to the social-engineering aspect of a phishing attack (Robila and Ragucci, 2006).

In terms of cost, automated classification can also be a more effective financial investment, as it is less expensive than user-education (Khonji et al., 2013).

Another benefit of automated classification is its adaptability to the evolution of phishing attacks (Abdelhamid et al., 2014). Attackers continuously modify the techniques used in phishing websites to evade detection, and therefore detection mechanisms need to be able to be able to evolve, too, in order to ensure that new attacks are successfully detected. Machine-learning is a widely-used approach that supports adaptation to the evolution of attacks - Section 2.6 reviews machine-learning algorithms used prominently in this field.

2.3 Offensive Defence

The purpose of Offensive Defence is to reduce the effectiveness of a phishing campaign (Karthika and Perumal, 2016) by the means of a counter-attack. While preventative measures intend to stop a user from submitting their confidential information, Offensive Defence aims to protect users for whom detective and preventative measures failed, and have subsequently revealed their credentials (Knickerbocker et al., 2009). This is usually done by submitting multiple fake credentials to the phishing site, hence making it difficult for the attacker to identify the real credentials within their data set (Kumaraguru et al., 2007). Offensive defence has several advantages, one of which is that it does not rely on a user's technical aptitude or ability to detect phishing sites. As explained in Subsection 2.2.1, it can be difficult to ensure that a user acts appropriately when faced with a phishing website, regardless of the education they have received in this context. Furthermore, a user may ignore a warning of a phishing website if they feel that their primary task takes precedence (Whalen and Inkpen, 2005), and hence Offensive Defence can be used to protect in such a scenario. The attack process is transparent to the user, and therefore does not usually interfere with a user's primary task or workflow (Ake-Johnson et al., 2010).

Offensive Defence mechanisms can also take advantage of NAT technology to protect users. If an attacker recognizes that submissions from certain IP addresses are due to Offensive Defence tools, they may decide to block those IP addresses. NAT is a widespread technology that allows many systems on a network to share the same IP address, therefore if one system causes the IP address to be blocked, then all systems behind the same IP will be unable to interact with the phishing site (Khonji et al., 2013).

BogusBiter (Yue and Wang, 2010) is a tool that falls in the category of Offensive Defence. It is built to be agnostic to detection methods, so that various blacklists and detection services can be used to identify and attack phishing websites. This means that as detection methods improve, the performance of *BogusBiter* will also improve proportionally (Yue and Wang, 2010). The tool is implemented as a browser extension, and leverages the use of detection tools that are already installed on the browser.

If a user follows the warning provided by a browser when navigating to a phishing site and leaves the web-page, *BogusBiter* will generate and submit a set of n credentials. If a user ignores the warning and submits data to the phishing site, *BogusBiter* will generate and submit $n-1$ credentials, therefore hiding the user's credentials amongst the generated set. While the former does not directly protect the user, this will still inject fake credentials to the attacker's collected data, thereby increasing the amount of labour required to identify real victims' credentials. *BogusBiter* also attempts to perform all HTTP requests in parallel, aiming to reduce the delay from performing many requests simultaneously.

The client-side nature of *BogusBiter*, however, has several disadvantages. If *BogusBiter* attacks a legitimate website, then the user could be prevented from logging in due to the large submission of requests to the website. Many websites implement a policy of maximum login attempts from a

particular IP address, and may lock or even suspend an account. Furthermore, many websites also make use of tests such as a CAPTCHA (Von Ahn et al., 2003) to prevent automated logins. These tests can be triggered by many login attempts too, and hence negatively impact a user's work-flow. In addition, *BogusBiter* does not provide protection against scenarios where the user's credentials are retrieved through the use of key-logging in JavaScript.

Humboldt is the implementation of a distributed approach to Offensive Defence (Knickerbocker et al., 2009). It overcomes some of the drawbacks of *BogusBiter*, such as the pattern produced by the multiple submissions from a single user and the reliance on the browser's detection mechanisms. The system requires a network of *Humboldt* clients, each of which submit small amounts of 'poisonous' data that creates a significant disruption in the attacker's data set. The pattern created by *Humboldt*'s submissions are more difficult to detect than those of *BogusBiter*, and therefore make the phisher's task of cleaning their data set more of a costly task.

Attackers are able to evade *Humboldt* by the means of using URLs with unique IDs. In order for *Humboldt*'s submissions to be effective and not be filtered out easily, *Humboldt* will need to recognize URL parameters which are used for unique IDs, and make these parameters unique across all submissions. *Humboldt* could overcome this by collaborating with email providers to gain a feed of detected phishing URLs from users emails, and being the first to submit data to those links. This is difficult, however, as it requires the end-user's agreement to extraction and transmission of information from their emails. *Humboldt* also relies on blacklists which can often be slow to update, especially due to short campaign lengths (Marchal et al., 2012) and if submissions to the blacklist require human intervention (Xiang et al., 2011).

The Offensive Defence technique has a heavy reliance on existing detection mechanisms. Through empirical evaluations, it has been shown that these techniques are effective (Knickerbocker et al., 2009, Yue and Wang, 2010), however the performance can be limited by the effectiveness of phishing detection. Therefore, an improvement to detection processes and services will positively impact the effects of Offensive Defence.

2.4 Corrective Action

The correction process involves using take-down methods to end phishing campaigns. Take-down methods focus on compromising the resources that phishing campaigns take advantage of (Khonji et al., 2013). The following describes ways in which campaigns can be disrupted and rendered non-functional once detected:

- Suspension of Hosting and Domain — Hosting service providers and domain registrars can suspend the attacker's accounts and therefore make websites inaccessible. This requires prompt cooperation from the service providers in order to be effective, and can be evaded through the 'Fast-Flux' technique (Moore and Clayton, 2007). If attackers have access to multiple hosts, then if one is taken down, the domain can be instructed to point at another host instantly.
- Suspension of Communication Channels — Similar to the method above, suspension of channels by which phishing URLs are distributed will limit the number of individuals exposed to the attack. These channels include Email, Social Media accounts, Phone Numbers (Abunadi et al., 2013).
- Removal of Hotlinked resources — Many phishing sites will link directly to resources on the legitimate pages, such as logos and background images (Mohammad et al., 2015). These can be amended on the system hosting the resources, therefore crippling the visual similarity between the phishing site and the target organisation.

- Tracing Botnet Command & Control Centers — As many phishing websites are also distributed through the use of botnets, the compromised system can be analysed to identify the command and control centre for the botnet, and therefore undertake processes to take these down too. This is effective in scenarios where the zombie[1] machine is either hosting the phishing website (Moore and Clayton, 2007), or the zombie machine itself is used to distribute the attacker's message (Khonji et al., 2013).

2.5 Preventive Action

Preventive Action ultimately aims to stop the submission of a user's credentials to a phishing website. Yue and Wang (2010) argue that users are the most unreliable component in the process of avoiding a phishing attack, and therefore services exist that take advantage of the detection process and block access to detected phishing sites, without requiring any detection work to be performed by the user.

Many of these services operate on the user's browser, some integrated by the developers of the browser (Ludl et al., 2007) as well as third party solutions such as the Netcraft Extension (Netcraft, 2017*a*), B-APT (Likarish et al., 2008) and SpoofGuard (Stanford Security Lab, 1998). When these tools detect that the user is accessing a phishing site (through various means such as using a maintained blacklist, heuristics, etc.), the user is shown a warning indicating the dangerous nature of the site. The performance of toolbars depends on both the reliability of the detection component, as well as the effectiveness of the warnings presented to the user from a usability perspective.

There has been a large focus on enhancing detection mechanisms, particularly through the use of machine learning (Abu-Nimeh et al., 2007, Aydin and Baykal, 2015, Khonji et al., 2011, Whittaker et al., 2010). These techniques can result in creating blacklists that are "comprehensive, error-free, and timely" (Whittaker et al., 2010), and therefore the preventative tools that are built upon these can better protect users from falling victim to phishing attacks. Sections 2.6 and 2.7 investigate the development of classifiers including choosing classification algorithms and performing feature selection.

2.6 Detection Through Machine Learning

In the context of phishing, the purpose of a classifier is to predict whether or not an input website is a phishing website, based on its characteristics (Abu-Nimeh et al., 2007). Initially, a set of properties are extracted from the website and are grouped together as a set of 'features'. Phishing website features are characteristics that, when observed, provide evidence of the website as being phishing or not. In order to perform an informed prediction, a classifier requires a 'training' phase where it 'learns' the characteristics that belong in each category. Two of the ways in which this can be done is through *supervised* and *unsupervised* learning.

Supervised learning is performed by supplying the classifier with feature vectors that are already labelled with the correct category (Mohri et al., 2012). The unsupervised approach, on the other hand, involves supplying unlabelled feature vectors, and the classifier infers the categories through processes such as clustering (Zander et al., 2005). Once the learning phase has occurred, the classifier will be able to categorise unseen inputs.

Choosing which classification algorithm to use depends on nature of the data being classified, as well as the behavioural requirements. It is evident that different classifiers will have different predictive performance, as investigated by Abu-Nimeh et al. (2007). In the context of phishing website classifi-

[1] A machine that has been compromised for malicious purposes and is under remote control.

cation, poor predictive performance can lead to benign sites being blocked or phishing websites not being blacklisted. This chapter will compare existing implementations of phishing website classifiers to understand the features which are most effective, and analyse the performance of various classification algorithms. The classifiers reviewed in this chapter were chosen due to their prominence in phishing detection literature, as well as the availability of sufficient data from their testing.

2.6.1 Classification Evaluation Metrics

The process of optimizing a classifier can be seen as a maximisation problem of various evaluation metrics. Initially, the results of classification involving n classes can be represented by a $n \times n$ 'confusion matrix' (Davis and Goadrich, 2006). The confusion matrix for a binary problem (Figure 2.3) allows the extraction of metrics such as accuracy, precision, and recall (Batista et al., 2004), as shown below.

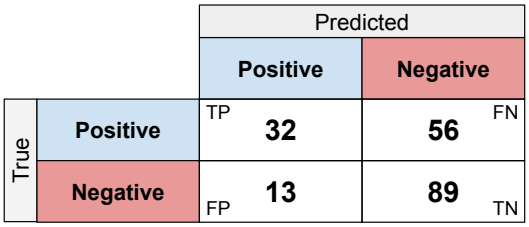

Figure 2.3: A confusion matrix for a binary classification problem where the classes are 'Positive' and 'Negative', with various metrics labelled. An example interpretation would be that 32 of truly 'Positive' samples were classified as 'Positive'.

Accuracy (Batista et al., 2004)
Accuracy represents the proportion of samples that were classified correctly, and can be calculated as:

$$\frac{TP + TN}{TP + FN + FP + TN} \tag{2.1}$$

Error Rate (Batista et al., 2004)
The error rate is the converse of accuracy and represents the proportion of samples that were classified incorrectly, and is calculated as:

$$\frac{FN + FP}{TP + FN + FP + TN} \tag{2.2}$$

Batista et al. (2004) indicate that, however, accuracy and error rate can be misleading as they favour the 'majority class'. For example, if 95% of the samples belonged to the positive class, then labelling all samples as belonging to the positive class would generate a classification accuracy of 95%. In the context of phishing classification, this is particularly important due to the imbalanced distribution of phishing urls in various url feeds. Gyawali et al. (2011) state that in their investigation of phishing URLs in a 'realistic-scenario', it was identified that the average ratio of phishing:non-phishing in the feed they used was $1 : 654$.

Precision (Davis and Goadrich, 2006)
Precision measures the number of samples labelled as positive that are in fact positive. This is particularly important as labelling a benign website as phishing can lead to a disruption in the service provided by the benign website. It is given by:

$$\frac{TP}{TP + FP} \tag{2.3}$$

Recall (Davis and Goadrich, 2006)
Recall, also known as the True Positive Rate, measures the proportion of positive samples that were correctly labelled as positive. It is calculated as:

$$\frac{TP}{TP + FN} \tag{2.4}$$

F_1 **Score** (Yang and Liu, 1999)
The F_1 measure is a balanced mean of precision and recall, and ranges from 0 to 1. It is given by:

$$\frac{2rp}{r + p} \tag{2.5}$$

where r is recall and p is precision.

Receiver Operating Characteristic
A ROC curve can be generated for a classifier by plotting True Positive Rate, TPR, against False Positive Rate, FPR, at various decision thresholds. The generated curve thus indicates the trade-off between the TPR and FPR for various decision thresholds (Fawcett, 2006). The ability of the classifier to discriminate between classes is depicted by the area under the curve, AUC. A classifier that performs random classification of samples would have an AUC of around 0.5 (dashed red line on Figure 2.4). A classifier that is worse than this would have an $AUC < 0.5$. A better ability to discriminate between the positive and negative class is indicated by a larger AUC.

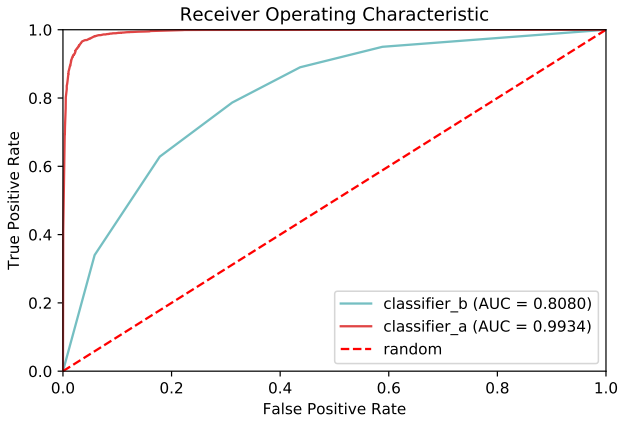

Figure 2.4: ROC curves for two classifiers, "classifier_a" and "classifier_b".

2.6.2 Classification Algorithm Review

Several classification algorithms were found to be prominent in existing literature, including Support Vector Machines, Random Forests and Neural Networks. The following section will aim to provide an overview of these algorithms, and identify the performance achieved when applied to phishing classification.

2.6.2.1 Naive Bayes

The Naive Bayes classifier assumes that the values of features in the feature vectors are independent (Nunan et al., 2012). The algorithm computes the posterior probability that a vector of features, x, belongs to a particular class, and the class with the highest probability indicates the category that the feature vector belongs to (Ma, Saul, Savage and Voelker, 2009a). Nunan et al. (2012) stated that Naive Bayes can display a "high recognition rate", and has "low computation cost", though within the context of phishing classification, this algorithm is often the worst performing (Ma, Saul, Savage and Voelker, 2009a, Miyamoto et al., 2008).

2.6.2.2 Support Vector Machines (SVM)

Support Vector Machines operate by identifying the largest margin (a hyperplane) by which a collection of feature vectors can be split into two classes (James et al., 2013). Due to this nature, SVMs are well suited towards binary classification problems, such as whether or not a particular website is a phishing website. In order to classify an unseen feature vector, the distance between the hyperplane and the feature vector is calculated, and subsequently used to predict the class of the feature vector.

Abu-Nimeh et al. (2007) indicated that the training process for SVMs is computationally expensive, and SVMs can also suffer from overfitting due to noisy data. Training of an SVM is performed by using a kernel function, the purpose of which is to map the vectors into a dimension where linear separation is possible. Various kernel functions exist, though the best SVM performance was observed when the 'Radial Basis Function (RBF) kernel' was used (Bergholz et al., 2008, Chu et al., 2013, Ma, Saul, Savage and Voelker, 2009*a*). Unfortunately, Ma, Saul, Savage and Voelker (2009*a*) also identified that using the RBF kernel resulted in slow performance when compared to using a linear kernel, with an increase in computational time by almost a factor of 2. Abu-Nimeh et al. (2007), Gyawali et al. (2011), Miyamoto et al. (2008) found that SVMs performed consistently worse than other algorithms in terms of precision, recall and accuracy.

2.6.2.3 Random Forest

The Random Forest algorithm is based on the principle of constructing multiple decision trees, such that each tree is generated from a random sample of training data (Breiman, 2001). This is similar to "Bootstrap Aggregation" where multiple decision trees are created too, however, the difference lies with the logic used to perform splits at nodes in the trees. With "Bootstrap Aggregation" all features from the feature vector are considered to find the best split of data. With Random Forest, however, only a random sample of features are considered at the point of splitting in order to reduce the correlation between trees (Liaw and Wiener, 2002). This has proven to increase both classification accuracy and classification speed (Breiman, 2001). Once the trees have been created, each tree generates a predicted class for a particular unseen feature vector. The class of the feature vector is determined by an average of all outcomes. Liaw and Wiener (2002) also describe a weighted approach when averaging the classes produced by the forest.

This algorithm exhibits characteristics that are desirable for the purposes of phishing website classification, and has been implemented and tested with success in various studies (Abu-Nimeh et al., 2007, Mohammad et al., 2014, Ramanathan and Wechsler, 2012). Abu-Nimeh et al. (2007) identified that Random Forest had the lowest prediction error and lowest false negative rate amongst all the algorithms that were investigated. Sanglerdsinlapachai and Rungsawang (2010) also identified a similar result with the performance of the algorithm.

Whittaker et al. (2010) also state the suitability of Random Forest for phishing website classification. When compared against their proprietary algorithm (which maintains Google's phishing blacklist), it was identified that Random Forest performs similarly, and would sufficiently "substitute" the proprietary algorithm.

Ramanathan and Wechsler (2012) analysed the computation time of the algorithms investigated, and Random Forest was found to be the fastest when performing multi-class classification, and second-fastest when performing binary classification. Abu-Nimeh et al. (2007) mentions that Random Forests have the disadvantage of being difficult to reproduce due to the random generation of trees. However, Ramanathan and Wechsler (2012) found similar performance when testing the performance of the algorithm, indicating that while the trees may be generated differently, the output remains similar.

2.6.2.4 AdaBoost

AdaBoost is a classification algorithm that is similar to Random Forest due to the principle of using multiple "weak" classifiers to estimate the class of a sample. Using the training samples, an initial decision tree is constructed. An analysis is then performed to identify the incorrectly classified samples, and a new iteration of the tree is constructed such that incorrectly classified samples carry a greater weight when impacting the model (Zhu et al., 2009). The process of assigning increased weights to these samples is known as "boosting", and the iterative construction of trees is repeated until a desired number of classifiers is reached, often in the numbers of 500-1000 (Zhu et al., 2009). In order to classify a sample, a weighted average is taken of the decisions of all trees.

This algorithm has been used in various studies (Sanglerdsinlapachai and Rungsawang, 2010, Xiang et al., 2011), though the observed performance was worse than the performance observed for other algorithms using the same data sets. While not performing as poorly as Naive Bayes, AdaBoost achieved an error rate of 9-10% when evaluated by Sanglerdsinlapachai and Rungsawang (2010), and F1-scores of 89.5-90.5%. Other algorithms in the same study, however, achieved error rates as low as 7.5% and F1-scores as high as 91.7%.

Miyamoto et al. (2008), however, found that using AdaBoost yielded results that were very similar to the best-performing algorithms in terms of error rate, F1-score and ROC AUC. This suggests that AdaBoost may be beneficial for particular heuristics, as some of the features used by Miyamoto et al. (2008) are very different to those used by Sanglerdsinlapachai and Rungsawang (2010).

2.6.2.5 Artificial Neural Network

An Artificial Neural Network consists of interconnected nodes (neurons). The connections (synapses) send signals to nodes based on a weighting that is determined during the training phase (Abu-Nimeh et al., 2007). The nodes apply a summation function to all input values from synapses, and apply an activation function using the resultant value. A neural network consists of 3 types of layers - an input layer, $n > 0$ hidden layers and an output layer. The input layer depends on the size of the feature vector, and the output layer contains a single node which outputs the predicted class.

Sanglerdsinlapachai and Rungsawang (2010) observed the performance of a Neural Network for the purposes of phishing website classification, and identified that it had the lowest error rate of 8% when compared against other algorithms. Abu-Nimeh et al. (2007), however, observed a much higher error rate with the majority of false detections being false negatives. The discrepancy between the results may be due to the differences in nature of the feature vectors used; Abu-Nimeh et al. (2007) used a feature vector of size 48, whereas Sanglerdsinlapachai and Rungsawang (2010) only used a feature vector of size 8.

2.6.2.6 Logistic Regression

Logistic Regression is often used for binary classification (Friedman et al., 2001). The model uses linear functions to predict the probablity of a sample belonging to a class, and therefore is useful when there exist linear relationships between the feature values and sample classes (Ma, Saul, Savage and Voelker, 2009*a*). Abu-Nimeh et al. (2007) identified that using Logistic Regression resulted in the highest precision, and closely followed the performance of Random Forest when observing the F1-score. Ma, Saul, Savage and Voelker (2009*a*) also had success when using Logistic Regression, achieving the lowest error rate on 2/4 data sets, only being marginally defeated when other data sets were used. Garera et al. (2007) interestingly remark that the algorithm is "computationally efficient to evaluate", indicating its possible suitability for large-scale systems.

Miyamoto et al. (2008), however, found mixed results when using LR, often ranking in the middle when compared against eight other classification algorithms. Random Forest, SVM, Adaboost and Neural Network surpassed Logistic Regression when considering the various evaluation metrics described in Section 2.6.1.

2.7 Features

A feature vector contains pieces of information (features) which can be used to distinguish between phishing and non-phishing websites. The nature of these features can range from elements from within the page content (Abdelhamid et al., 2014, Fette et al., 2006, Mohammad et al., 2014) to information extracted from the domain (Abunadi et al., 2013, Whittaker et al., 2010), and therefore must be considered in terms of their effect on classification performance. The number of features being used is also a factor of consideration, as using large numbers of features could lead to over-fitting (Abu-Nimeh et al., 2007).

Le et al. (2011) provide a distinction that separates features into two categories: *lexical* and *external*. Lexical features are extracted solely from URL, and do not require accessing external resources. External features encompasses features that are extracted from the webpage and any third-party services that can provide useful information for classification purposes. While external features provide a larger amount of information regarding the phishing attack, retrieving these features comes at the following costs (Le et al., 2011):

- Additional latency during the feature extraction period.
- Introduction of a dependency on the availability of mentioned external services.
- Increased usage of client resources, e.g. network bandwidth.
- Side-effects may occur as a result of making the request, e.g. submitting a form with GET parameters in the URL (Garera et al., 2007).

Aburrous et al. (2010) proposed an initial categorization of features, however do not explicitly make the distinction between *lexical* and *external*. The following sections are a refinement of the original categorization proposed by Aburrous et al. (2010) with the above taken into consideration. A sample of features, examples and the studies that used the features can be found in Table A.1.

2.7.1 Lexical Characteristics of URLs

Lexical features range from whether or not a suspicious TLD [2] is used, to counting the number of special characters. Figure 2.5 is an example of a phishing website URL with some of the features identified that could be used for classification:

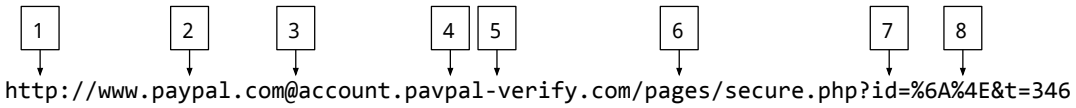

Figure 2.5: An example of features that could indicate phishing activity. Table 2.1 contains explanations for each feature.

[2] Top-Level Domain, e.g. *.com*

#	Description
1	Phishing attackers often do not acquire SSL certificates, which is reflected by the lack of HTTPS (Mohammad et al., 2015).
2	The presence of the name of an organisation is suspicious if the domain itself is not owned by the organisation (Khonji et al., 2011).
3	Using an @ symbol is suspicious since a web-browser would ignore parts of the URL before the @ symbol (Aburrous et al., 2010).
4	`pavpal` is visually similar to `paypal`. The swapping of the 'y' to a 'v' could mislead a user, and hence is suspicious (Aburrous et al., 2010).
5	The presence of a character from a set of 'suspicious characters' (Vasava and Mangrule, 2015) could indicate phishing activity.
6	Le et al. (2011) found that the number of sub-directories could indicate phishing activity.
7	Phishing URLS often contain large numbers of arguments (Le et al., 2011).
8	Characters can be encoded using hexadecimal, in order to mask the actual URL. This is a form of obfuscation that many phishing attackers employ to mislead victims, and therefore the presence of hexadecimal values is suspicious (Shekokar et al., 2015).

Table 2.1: Description of features identified in Figure 2.5.

Use of N-Grams

Gyawali et al. (2011), Blum et al. (2010) and Darling et al. (2015) investigated the use of n-gram modelling for the purposes of phishing classification. N-grams are overlapping chains of n consecutive tokens extracted from the URL. For example, from the word `PHISH` we can generate 4 *bigrams* using individual characters: `PH`, `HI`, `IS`, `SH`.

The approach taken by Gyawali et al. (2011) and Blum et al. (2010) involves using n-grams of URL tokens to contribute to their bag-of-words representation. Unigrams and bigrams of various segments of URLs are extracted and used as binary features if more than 1 URL from the same class share the particular unigram/bigram. It is important to note that this causes the overall feature vector to be very large.

Darling et al. (2015), however, use characters extracted from the URL to form their n-gram models. The investigation extracted n-grams of up to 4 characters from only benign URLs, and calculated the probability of the n-grams appearing in both the benign class and phishing class. The rationale behind collection from only benign URLs lies on the author's assumption that the phishing data set had a "higher degree of variability" and thus is more difficult to characterise as n-gram models. A set P was created:

$$P = \{p_1, p_2 ... p_n\}$$

where each element of P is the probability of a particular n-gram occurring in the benign class. Thus, for each URL, a score was calculated as the sum of probabilities in P for every n-gram that appeared in the URL. While this approach appeared to be effective, a concern regarding credibility can be raised. Gyawali et al. (2011) and Whittaker et al. (2010) both observed that in a real-life scenario, there is a large imbalance in the ratio of phishing to benign URLs. However, this imbalance is the converse of the data set used by Darling et al. (2015) - the volume of phishing URLs is in fact much lower than the volume of benign URLs. Gyawali et al. (2011) observed a ratio of 1:650, and Whittaker et al. (2010) observed a ratio of 1:100 of the number of phishing to benign URLs. The effectiveness of the approach taken by Darling et al. (2015) could perhaps be attributed to overfitting of their smaller benign data set.

It has been shown that lexical features of URLs alone can help successfully detect phishing activity (Blum et al., 2010, Darling et al., 2015, Le et al., 2011). A primary advantage of classification based on lexical features is the fact that no further content needs to be fetched, hence decreasing the overall computation time.

Furthermore, the security risks that are involved when fetching content is eliminated, as is the overhead with having to process extra content (Blum et al., 2010). This is particularly important since attackers often "cloak" their websites by ensuring that requests from certain clients are served with benign content (Ma, Saul, Savage and Voelker, 2009*a*), and therefore performing a content-based analysis may result in failure to detect phishing activity. Figure 2.6 is an extract from a php script found in a phishing-kit, demonstrating the mechanism used to "cloak".

```
$blocked_words = array(
  'google',
    ...
  'cloudflare'
);
$bannedIP = array(
  '^104.236.153.*',
  '^107.170.*.*',
    ...
  '^95.76.156.*'
);
if (in_array($_SERVER['REMOTE_ADDR'], $bannedIP)) {
    header('HTTP/1.0 404 Not Found');
    exit();
} else {
    foreach ($bannedIP as $ip) {
        if (preg_match('/' . $ip . '/', $_SERVER['REMOTE_ADDR'])) {
            header('HTTP/1.0 404 Not Found');
            die('<h1>404 Not Found</h1>The page that you have
                 requested could not be found.');
        }
    }
}
foreach ($blocked_words as $word) {
    if (substr_count($hostname, $word) > 0) {
        header('HTTP/1.0 404 Not Found');
        die('<h1>404 Not Found</h1>The page that you have
             requested could not be found.');

    }
}
```

Figure 2.6: Extract from a phishing-kit demonstrating the mechanism behind cloaking. The kit was found during collection of phishing URLs.

The code in Figure 2.6 shows that the website will serve a "404" message to clients requesting content from a *banned IP range* or from a *banned host*. A host is considered "banned" if it contains a "banned token" (the list of banned tokens includes names of many cyber-security organisations), and therefore directly requesting this content from such hosts would result in benign content being served, and possibly leading to misclassification. Using lexical features from the URL, however, is not susceptible to this form of evasion.

Le et al. (2011) conducted a study to understand the differences in performance between using just lexical features, and in combination with external features (e.g. domain information, page content). A very small decrease (1%) was found when using solely lexical features of URLs, supporting the notion that lexical features alone are sufficient for classifying phishing-websites. While Whittaker et al. (2010) have indicated that lexical characteristics of urls are much more easily manipulated by the attacker, many common indicators of phishing are still prevalent in URLs as they play a role in the social-engineering aspect of the phishing attack. Garera et al. (2007) categorize these indicators as methods by which attackers "obfuscate" the true nature of a phishing URL.

A comprehensive list of lexical-features used in literature can be found in Table A.2.

2.7.2 Host and Domain Based

While it has been proven that it is possible to use lexical features only from a URL to distinguish between phishing and benign websites, external information regarding the host and the domain is useful too (Ma, Saul, Savage and Voelker, 2009*a*). Pan and Ding (2006), for example, describe that an empty/incomplete DNS record should raise suspicion, especially if the identity claimed does not match the WHOIS identity (ICANN, 2016).

Information regarding the popularity of the domain and traffic statistics can also help distinguish between legitimate and phishing sites. As mentioned earlier, phishing campaigns are usually very short (Marchal et al., 2012), and therefore websites that have receive low traffic can be considered suspicious (Xiang et al., 2011). Various third party services such as the traffic ranking provided by Alexa (Alexa, 2017) and Google's PageRank (Brin and Page, 2016) provide measurable statistics that can be used for this purpose. Xiang et al. (2011) used external information regarding the website heavily in their classifier which was able to achieve a true positive rate of 92%.

2.7.3 Security

A common identifier that is used in various studies (Aburrous et al., 2010, Zhang et al., 2007) is to observe if SSL is being used, and if not, the website should be flagged as a potential phishing website. However, Mohammad et al. (2014) mention that some phishing sites may use SSL but obtain their certificate from an untrusted certificate authority. The study subsequently states that a website is only considered 'legitimate' if it meets the following criteria:

- The HTPPS protocol is being used.
- The issuer of the SSL Certificate is from a set of trusted Certificate Authorities.
- The certificate is over 2 years old.

If the website does not meet any of the above conditions, it is considered either 'Phishing' or 'Suspicious' depending on how many conditions weren't met.

2.7.4 Script Analysis

Mohammad et al. (2015) and Pan and Ding (2006) provide several JavaScript-related features that phishing websites implement in order to deceive potential victims. For example, phishing sites may disable the right click option in a web-browser to prevent a visitor from observing the source code, which could reveal a phishing website's true nature.

Phishing pages usually extract personally information through the use of HTML forms, as this is the method by which users log in to legitimate services (Suriya et al., 2009). HTML forms have an `action` attribute which specifies the location of the resource which will process the data from the form. It has

been found that phishing websites often override the value in the `action` attribute to perform a void action, or point to a resource on a different domain (Alkhozae and Batarfi, 2011). Legitimate websites, on the other hand, will process results of a form submission on the same domain, therefore the value of the `action` attribute will point to a resource on the same domain.

Some features found in certain studies, however, are no longer very effective in detecting phishing sites. Many studies, for example, identify the use of the JavaScript `onMouseOver` function as an indicator of a phishing site. When a user hovers over a link, the web-browser will show the destination of the link in the 'status' textbox. Nguyen (2013) explains that this value is override-able using the `onMouseOver` function, therefore allowing an attacker to mask the real destination of a link. As most web-browsers have now prevented the modification of the 'status' textbox through code, this feature would be less prominent in new phishing campaigns.

2.7.5 Page Contents

Analysing the content of a web-page has proven to be a successful method by which phishing sites can be distinguished from legitimate sites. While phishing sites aim to imitate the target organisation, the importance of similarity is mainly on the visual aspect of the website as opposed to the DOM [3], i.e. the underlying elements of the web-page (Pan and Ding, 2006). An example of this is the use of external resources, such as images and links (Alkhozae and Batarfi, 2011). Since phishing websites will make use of existing images such as logos of the target organisation, it is easier for the attacker to embed the image directly from its location on the target organisation's web-servers. Furthermore, in order to better imitate the target organisation's website, the author may link to certain pages on the target organisation's site, such as Contact and Help pages. Therefore, by observing the number of resources that are hosted on a different domain, it is possible to distinguish between legitimate and phishing websites.

Moreover, analysing the page content can also reveal abnormal behaviour that a user may experience on the phishing website. An example of this would be anchor elements (`<a>`) that do not provide any action when clicked on, or link to a blank page through the use of the `about:blank` URI (Pan and Ding, 2006).

It is important to note that one particular study carried out by Ludl et al. (2007) determined that an analysis of page features did *not* in fact allow them to differentiate between phishing and non-phishing webpages. A review of this study shows that the reasoning behind picking some of the features is not in fact well founded. A large portion of the feature vector comprised of identifying the number of different types of form elements, due to the "prevalence of web forms on phishing pages". While it is true that phishing pages will most certainly include a form into which the user can enter their personal data, the target organisation will most likely have a web form too. In order to convince a potential victim that they are on a legitimate website, the attacker may structure the form similarly to the target organisation. Therefore, the number of form elements is not directly related to whether nor not the website is a phishing page. Such an error decreases the reliability of the results obtained by Ludl et al. (2007) and therefore page content can still be considered an effective resource to use for feature selection.

[3] Document Object Model (Robie, 1998)

2.7.6 Feature Selection

Several statistical methods also exist to support the selection of features, and quantify the ability for a particular feature to distinguish between classes. The *chi-squared* statistic is a popular method of measuring the performance of a feature (Novaković et al., 2011). An initial null hypothesis, H_0, is assumed, i.e. The feature cannot distinguish between the classes. An $I \times J$ "contingency-table" can then be created, where the value in n_{ij} is the frequency of a sample in class i having the value j for this feature (Stanberry, 2013). The chi-squared statistic (χ^2) can subsequently be computed using the following formula:

$$\chi^2 = \sum_{i=1}^{I} \sum_{i=1}^{J} \frac{n_{ij} - \mu_{ij}}{\mu_{ij}} \tag{2.6}$$

where μ_{ij} is the expected frequency under the null hypothesis (Stanberry, 2013).

The value for χ^2 indicates the strength of the feature to reject the null-hypothesis, i.e. a strong feature will have a larger χ^2 value. A corresponding *p-value*, p, can also be generated for the χ^2 value, which indicates the probability that a same or larger χ^2 value would be observed if H_0 is true. If $p < \alpha$, where α is a pre-determined threshold of statistical significance, then the test is considered statistically significant and the null hypothesis can be rejected (Pennsylvania State University, 2017). It is important to note that the *chi-squared* statistic does not perform well for small occurrences of features (Yang and Pedersen, 1997).

Related to the *chi-squared* statistic is *Matthew's Correlation Coefficient* (Matthews, 1975), also known as the *Phi-Coefficient*. It provides an indicator of the accuracy of a binary-classifier in relation to the accuracy of randomly labelling the data (Matthews, 1975). It can also be used to measure the performance of a feature, as the metric can be calculated for binary-classifier that uses a single feature, i.e. the feature being investigated. The value can be generated from the resultant confusion matrix (see Section 2.6.1), using the following equation:

$$C = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.7}$$

C will always lie between 1 and -1, where $C = 0$ indicates performance that is the same as a random-classifier, and $C < 0$ indicates performance that is worse than a random-classifier.

Mutual Information is another useful statistic for feature selection (Kwak and Choi, 2002). In the context of feature selection, this statistic can quantify how much information a feature provides in determining the class of a sample (Tourassi et al., 2001). The mutual information, I, of two variables X and Y (where X is the class of a sample, and Y is the value of the feature) can be calculated as (Cover, 1991):

$$I(X;Y) = \sum_{x,y} p(x,y) log \frac{p(x,y)}{p(x)p(y)} \tag{2.8}$$

where $p(x)$ and $p(y)$ are the probability distribution functions of X and Y respectively. A weakness of *Mutual Information*, as indicated by Yang and Pedersen (1997) is that for two features that have the same conditional probability, the rarer feature will have a higher score.

2.7.7 Feature Weighting

As certain features will be more effective than others to distinguish between classes, a form of weighting may be useful where certain features can carry a different weighting than others when contributing to the overall classification (Zhang et al., 2007). A simple method of calculating weights for features, as proposed by Zhang et al. (2007), is as follows:

1. For each feature, x_i, the True Positive (TP_i) and False Positive (FP_i) rate is calculated.
2. An effect, e_i, is calculated for each feature by $TP_i - FP_i$.
3. The weight, w_i, is calculated for x_i using: $\frac{e_i}{\sum e_i}$

An advantage of this method is that the weighting can be recalculated easily on demand, and therefore the weights can be kept up to date relative to the population of websites that flow through the classifier. Mohammad et al. (2014) also employ a similar form of weighting - the weight w_i of feature x_i is the ratio of that feature occurring in the data set of phishing sites.

Weighting can also occur at classifier-level, as opposed to feature-level. For example, the Random Forest algorithm involves generating a number of decision trees, each of which use a random selection of features to perform decisions. As the effectiveness of each tree is dependent on the features used, weights could also be assigned to the trees themselves when calculating the output class (Winham et al., 2013).

2.8 Use in a Large-scale Environment

A few studies have been performed to understand the usefulness of a machine-learning classifier in a large-scale environment. Prophiler is a malware classifier built by Canali et al. (2011) to reduce the load on a more expensive classification process. Prophiler used *full* features, and an incoming feed of crawled URLs as well as URLs extracted from a spam feed. Over 60 days, Prophiler analysed 18,939,908 web-pages, determining 14.3% as malicious (and subsequently reducing the load on the more expensive downstream process by 85.7%). During this period, a 13.7% false positive rate was observed. Prophiler is able to classify URLs at a rate of 3.7 pages/s[4] after creating a cache for retrieving external information.

The lexical classifier developed by Gyawali et al. (2011) reduced the load on their downstream process by 90%, and observed a 7.5% cumulative error rate over 90 days. This 90-day trial involved processing a combination of previously verified phishing sites, and a feed of spam URLs in which 0.01% URLs are phishing, with an overall ratio of 1:651 (0.0015) of phishing to non-phishing URLs.

Whittaker et al. (2010) tested their classifier with a large number of URLs (9,388,395) where 1.1% of the URLs are phishing. Using five-fold cross validation, a 0.03% false positive rate was observed. This classifier is much more computationally expensive, however, with each URL taking a median of 76s for classification. In relation to Prophiler (Canali et al., 2011), this classification system is more similar to the downstream system that Prophiler filtered the initial set of URLs to. This investigation also uses a proprietary classification algorithm, though the authors have stated that a Random Forest model consisting of 100 trees trained on 25% of the training data performs similarly.

As mentioned by Canali et al. (2011), a fast classifier can support a slower, more intensive classifier by acting as an initial filter. Due to the short-life nature of phishing campaigns, detection needs to be as fast as possible. The architecture of using a two-step classification process described in this section makes tackling phishing in a large-scale environment possible.

[4] A system with an 8-core Intel Xeon Processor and 8GB RAM was used.

3 Data Collection

The following is a description of the data sets used for both the experimentation phase and the implementation phase.

Data set Name	Source	Size (Urls)	Notes
Benign URLs			
dmoz†	DMOZ (2016)	100,000	Archive of reviewed benign websites.
feroz	Feroz and Mengel (2015)	8,023	The benign data set used by Feroz and Mengel (2015).
alexa	Alexa (2017)	200,000	Taken from the top 1 million sites list Alexa (2017).
google	Google (2016)	2380	URLs taken from Google Search using the `allinurl:` operator. Used by Ludl et al. (2007), Sanglerdsinlapachai and Rungsawang (2010).
startingpoint	Google (2016)	200,000	As used by Mohammad et al. (2014).
spam_benign†	Netcraft (2017*a*)	100,000	A collection of spam urls verified as non-phishing by Netcraft (2017*a*). This is a noisy data set where some phishing-urls may exist, however were categorized as benign as at the time of detection the sites were offline. It is also important to note that these URLs in general are much more similar to the syntax of phishing URLs, with complex path and query components.
Phishing URLs			
phishtank†	PhishTank (2017)	26,508	Archive of manually-verified phishing URLs.
cleanmx	net4sec (2016)	47,547	A public list of verified phishing URLs.
openphish	OpenPhish (2016)	2,345	A public list of verified phishing URLs.
spam_phish†	Netcraft (2017*a*)	100,000	A collection of verified phishing URLs sourced from a feed of spam URLs.

Table 3.1: A description of the data sets used in this report. Sets indicated with † are used primarily, with the remaining used to aid with comparisons against existing studies.

For experiments that aided with the identification of an optimal feature vector (Chapter 4), and optimal classification algorithm (Chapter 5), two distinct pairs of balanced data sets were used:

- **SBSP**: Spam-Phish-Spam-Benign, consisting of 100,000 URLS from **spam_benign** and 100,000 URLs from **spam_phish**.
- **DP**: Dmoz-Phishtank, consisting of 25,000 URLS from **dmoz** and 25,000 URLs from **phishtank**.

4 Optimal Feature Vector

The aim of this investigation is to identify features that can effectively distinguish between phishing and benign URLs, where the "effectiveness" is calculated and empirically evaluated. Initially, the structure of a URL is defined to a level of granularity appropriate for URL classification. A set of features is then constructed, which consists of features found in existing literature, as well as novel features. Finally, the features are evaluated using appropriate statistical tests, enabling the identification of the features that have the greatest ability to distinguish between phishing and benign URLs. The most effective features are used to compose an "optimal feature vector". The following section outlines this process in detail.

4.1 URL Structure

A URL can be defined as having the following syntax, as per RFC 1738 (Berners-Lee et al., 1994):

```
http://<host>:<port>/<path>?<searchpart>
```

Figure 4.1: Structure of a HTTP URL as per RFC 1738 (Berners-Lee et al., 1994).

The following diagram shows the way in which a URL can be decomposed to a finer level, therefore allowing more granular feature extraction:

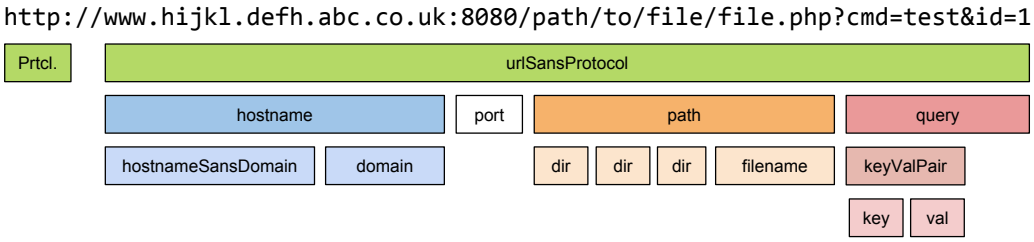

Figure 4.2: The decomposition of a URL to a finner level than RFC 1738 (Berners-Lee et al., 1994). Figure A.1 shows the grammar for this decomposition.

From the aggregated list of features (Table A.1), it is evident that many studies decompose a URL to a similar level, differentiating between the host and path components of a URL for example. However, there is a need to decompose these parts further, due to the discrepancies that can arise when evaluating properties such as number of tokens. For example, a feature used by many studies (Le et al., 2011, Ma, Saul, Savage and Voelker, 2009*a*, Zhang et al., 2007) is to count the number of dots in the hostname part of a URL, as phishing attackers will often use multiple subdomains to mislead users, e.g. `verify.paypal.acc0unt.com`. A higher value for this feature indicates that the URL is suspicious. However, a legitimate URL may use a generic Second-Level-Domain (SLD), such as `.co.uk` which immediately increases the number of dots by 1 in the hostname. Being able to account for this will enable better differentiation of situations when an increased number of subdomains is suspicious rather than due to using a generic SLD. A similar principle can be applied to instances where path portions of URLs contain default file names such as `index.html` which increases the token count but may refer to the same resource had the file name not been explicitly defined.

4.2 Initial Feature Vector

An initial feature vector was chosen from the aggregation of features used in literature (Table A.2). All features were included, except those that aim to extract the brand that a phishing attack is imitating, and n-gram-based features for which a novel approach was taken. Brand extraction features were avoided as the purpose of the classifier is to highlight URLs that contain cues indicative of phishing, as opposed to cues that relate to a particular brand. It is important to note that features that indicate that "brand-imitation" in general might be taking place do not fall under this category, because these features still retain the generic nature of classification. The work by Pan and Ding (2006) indicate that brand extraction is best performed when taking into account external content such as the web-page, and therefore attempting to extract the brand using lexical features only may not be effective.

4.3 Novel Features

The following section describes the novel features that were identified. The features fall into three distinct categories - token-frequency-based features, binary features, and frequency/continuous features.

4.3.1 N-gram and Token Frequency Based Features

As shown in Section 2.7.1, n-grams have been used by other studies when extracting features from a URL. The bag-of-words approach by Gyawali et al. (2011) and Blum et al. (2010) involved allocating a value in the feature vector for each n-gram, where the value indicated the frequency of the particular n-gram appearing in the URL. This results in a very large, sparse feature vector (Blum et al. (2010) obtained a feature vector of size 369,585 features).

While Darling et al. (2015) describe that their approach of using n-grams is more effective than the bag-of-words approach, concerns can be raised about real-world performance as these n-grams are extracted from benign URLs only and it is based on the assumption that highly-frequent tokens in the benign set are not frequent in the phishing set.

The approach taken in this investigation was to use n-gram data to generate a score for a URL, indicating its likeness to a particular class based on the presence of previously observed n-grams extracted from various parts of the URL. A description of the algorithm to generate the score is as follows:

1. Define a tokenizer function, g, that will convert a URL into the list of tokens, and return a list of n-grams. For example, this function may extract all directory elements of a URL's path, and convert it into bigrams ($n = 2$), as shown in Figure 4.3.

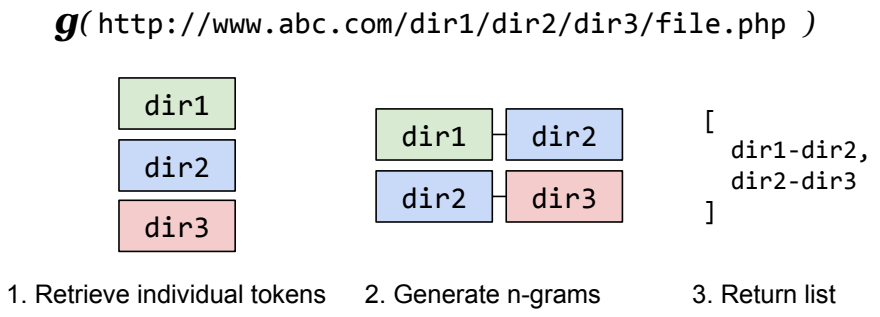

Figure 4.3: An example of a tokenizer function that generates bigrams of directory elements.

2. Obtain a set phishing URLs, C_p, and a set of benign URLs, C_b.

3. Define two new sets, T_p and T_b. For each URL in C_p, tokenize using g and update T_p with the tokens obtained. Repeat for C_b. As a result, we have a set of tokens for each class.

$$T_p = t_0^p, t_1^p, ..., t_n^p \tag{4.1}$$
$$T_b = t_0^b, t_1^b, ..., t_n^b \tag{4.2}$$

4. For each token t_i^c, from T_c, a score s_i^c is calculated, and stored in set S_c. The score indicates the ability of the particular token to distinguish between classes based on its presence/absence.

$$S_p = s_0^p, s_1^p, ..., s_n^p \tag{4.3}$$
$$S_b = s_0^b, s_1^b, ..., s_n^b \tag{4.4}$$

Consider a classifier that labels URLs as belonging to class c based on the presence of t_i^c. A 2x2 confusion matrix can be created, and the score for t_i^c is calculated as the Matthew's Correlation Coefficient (MCC) of this classification (see Section 2.7.6). For example, if the token "login" appeared in 1923/2000 phishing URLs, and 73/1400 benign URLs, the confusion matrix would be constructed as shown in Figure 4.4, and the MCC would be extracted using Equation 2.7. Section 2.6.1 describes the method by which confusion matrices are constructed.

	P	B
P	1923	77
B	73	1327

MCC(`login`) = `0.913`

Figure 4.4: Calculating the score for the token "login".

Tokens which appear with similar frequencies in both classes will obtain a low score, whereas tokens with a larger difference in frequency between the classes will have proportionally higher scores. A advantage of using MCC as the scoring function is that the effects of class imbalance are mitigated.

5. Given an out-of-sample URL, the likeness of the URL to a particular class, c, can now be calculated. The URL is initially converted into a list of tokens, $W = w_1, ..., w_n$ using g. $W \cap T_c$ provides a set of tokens from the URL for which respective scores exist in S_c. The values in S_c corresponding to the tokens in $W \cap T_c$ are summed and returned. Thus, the likeness L_c, of a tokenized URL, W, to a class, c, is defined as:

$$L_c = \sum_{t_i^c \in W} s_i^c \tag{4.5}$$

This algorithm is particularly advantageous over the efforts by Darling et al. (2015) as it takes into account the occurence of the same tokens in both phishing URLs and benign URLs (unlike Darling et al. (2015) who only look at the benign class), and therefore can provide a higher weight to tokens that appear frequently in one class, and lower in the other.

Finally, this algorithm is very loosely-coupled with the scoring metric used (in this case MCC). Other statistical methods exist that can also provide a value for discriminatory ability, as discussed in Section 2.7.6, which can be integrated with ease.

Tokenizers

The above algorithm requires a tokenized representation of the URL. Some methods of tokenization can be extracted from literature, such as obtaining tokens from the entire URL by splitting using special characters as delimiters (Blum et al., 2010). Table 4.1 is a list of tokenization methods used with the above algorithm in this study. The sizes of the subsequent n-grams generated from the tokens are shown in the 'n' column.

For most methods, likeness scores were generated for the phishing class only, based on the assumption that benign URLs exhibit more variation, and thus generating a representative list of tokens for benign URLs is more difficult. This assumption is justified as instances where phishing detection has been observed in a real-life scenario, the number of benign URLs encountered has been significantly larger than the number of phishing URLs (Gyawali et al., 2011, Le et al., 2011, Whittaker et al., 2010).

Tokenizer	n	Description	Example
`URL_TOKENS`	1, 2, 3, 4	A region from the URL is split using a list of special characters, {`?, -, _, ., =, &, /` } Blum et al. (2010), Darling et al. (2015), Garera et al. (2007).	`www, example, website, abc, com, 3232, path, to, the, file, php, cmd, test, id, 1`
`HOSTNAME_ TOKENS`	1, 2, 3, 4	The `URL_TOKENS` tokenization applied to the host portion of the URL only.	`www, example, website`
`DOMAIN_ TOKENS`	1, 2, 3, 4	The `URL_TOKENS` tokenization applied to the domain portion of the URL only.	`abc, com`
`PATH_TOKENS`	1, 2, 3, 4	The `URL_TOKENS` tokenization applied to the path portion of the URL only.	`3232, path, to, the, file, php`
`PATH_MODEL`	3, 4, max[1]	The path region is split into a list of segments using '/', and each segment is encoded using a symbol representing the types of characters in the segment. A particular distinction is made for segments that contain only letters, only digits, only symbols, and mixtures of both. Table 4.2 is a table showing the symbols used to encode particular path segments. The justification for this arises from the fact that phishing urls will often have common paths due to the re-use of phishig kits (Wardman et al., 2009), and therefore identifying prominent paths in phishing URLs may be beneficial. Furthermore, phishing attackers will use vulnerabilities in existing applications running on web-servers to upload their phishing content, and therefore avoiding the need to invest in a web-server themselves (Wardman et al., 2009). Certain vulnerabilities may be the same across web-servers, and therefore can be picked up using this method.	`d, n, a, n`
`QUERY_KEYS`	1	The query part of a URL is list of keys and values. This tokenizer generates a list of keys.	`cmd, id`
`QUERY_VALS`	1	Same as `QUERY_KEYS`, except the values of the query string are used instead.	`test, 1`
`FILE_NAME`	1	The file name without the extension.	`file`
`FILE_EXT`	1	The file extension from the URL is retrieved.	`php`
`LAST_PATH_ SEGMENT`	1	The last element of the path is retrieved. If a file is specified then this is usually the file name, otherwise this will be the last directory.	`file.php`

Table 4.1: The different methods of tokenization, using which likeness scores are calculated.

[1] *max*: A concatenation of all elements.

Type of Characters Contained			Encoded As
Letter	Digit	Symbol	-
•	-	-	a
-	•	-	d
-	-	•	s
•	•	•	n
•	-	•	n
-	•	•	n
•	•	-	m
-	-	-	x

Table 4.2: The scheme used to encode a path as a sequence of symbols, each symbol representing the type of characters present in each path segment. This is used with the `PATH_MODEL` tokenizer described in Table 4.1.

4.3.2 Binary Features

Binary features indicate either the presence or absence of a particular quality. Table 4.3 is a list of the novel binary features that were identified.

Feature	Description
`path_php`	`1` if the URL points to a php file, `0` otherwise. In an evaluation of 584 phishing kits by Cova et al. (2008), all kits were found to be using PHP. The reason for this is believed to be because PHP is widely offered by web-hosting providers and is "supported by most web-servers" if it needs to be installed (Cova et al., 2008). Using a more obscure technology would limit the hosts that the phishing attack could be deployed on.
`path_sensitive`	`1` if the path contains sensitive terms, `0` otherwise. While conceptually this is not a novel feature, several studies devise separate sets of "sensitive terms". This feature looks from a pool of sensitive terms aggregated from all studies (Bergholz et al., 2008, Lee et al., 2015, Ma, Saul, Savage and Voelker, 2009*a*, Zhang et al., 2007). The full list of sensitive terms can be found in Figure A.2.
`host_sensitive`	Same as `path_sensitive`, however applied to the hostname region of the URL only.
`more_than_one_dot_path`	`1` if the path contains more than one dot ('`.`'), `0` otherwise. Based on the obfuscation techniques attackers use, as investigated by Le et al. (2011).
`underscore_path`	`1` if an underscore ('`_`') is present in the path, `0` otherwise. Based on the obfuscation techniques attackers use, as investigated by Le et al. (2011).

Feature	Description
`suspicious_prefix`	`1` if a token contains a sensitive-term AND an underscore or hyphen, `0` otherwise. Tokens are found by splitting the url by special characters (except underscores and hyphens). Based on the findings of (Mohammad et al., 2015) where phishing URLs were found to exhibit this characteristic, though (Mohammad et al., 2015) only checked for the presence of a hyphen.
`suspicious_brand_prefix`	`1` if a token contains a brand name AND an underscore or hyphen, `0` otherwise. Tokens are generated in the same way as for `suspicious_prefix`. The reasoning behind this is the same as for `suspicious_prefix`. A list of brand names that are used by phishing attacks was obtained from Netcraft (2017*b*).
`brand_in_<region>`	`1` if a brand name is embedded in a particular region of a URL, `0` otherwise. This is calculated for domain, host and path regions. A further feature is included for presence of a brand name in a list of path tokens, requiring an entire token to match the brand name. This allows differentiation of when a brand name is merely embedded due to being part of a word, as opposed to forming an entire token. Looking for a presence of a brand has been shown to be effective (Chu et al., 2013).

Table 4.3: The list of novel binary features identified.

4.3.3 Frequency/Continuous Features

These features are either means or frequencies of particular characteristics.

Feature	Description
`numbers_in_path`	Number of individual digits found in the path.
`max_dots_path_segment`	Largest number of dots (‘`.`’) found in a path segment.
`num_dots_host`	Number of dots found in the hostname (excluding the domain name).
`hyphens_host`	Number of hyphens (‘`-`’) found in the hostname.
`hyphens_path`	Number of hyphens (‘`-`’) found in the path.
`num_tokens_url`	Number of tokens found in the URL.
`longest_sub_dir`	The length of the longest sub-directory found in the path region.
`shortest_<host|path>_token`	The length of the shortest token from the hostname tokens and path tokens.
`num_special_chars`	Number of special characters in the full URL.
`num_special_chars_path`	Number of special characters in the path region.
`max_arg_value_length`	The largest url argument length.
`mean_arg_value_length`	The mean url argument value length.
`full_host_name_length`	The length of the hostname.
`path_length`	The length of the path.

Table 4.4: The list of novel frequency/continuous features identified.

4.4 Testing

The next step in the process is to empirically evaluate the performance of the current set of features, and remove those that are not effective at distinguishing between classes. Each feature was evaluated using the *chi-squared* test (See Section 2.7.6), to obtain a score and p-value indicating the strength and statistical significance of a feature's ability to determine the class of a sample.

For each data set, SBSP and DP, meta-data (such as MCC scores of tokens) was collected from a training partition and a Feature Extractor was initialized. Using the Feature Extractor, a feature vector was generated for every URL in the test set. *Chi-squared* scores were subsequently generated for each feature.

This process was carried out using 5-fold cross validation, where the data is split into 5 partitions, and 4/5 partitions are used for training and the remaining partition is used for testing. Each fold uses a different partition as the testing set. The scores from each combination of partitions were subsequently averaged. Figure 4.5 outlines the process of generating scores for one fold.

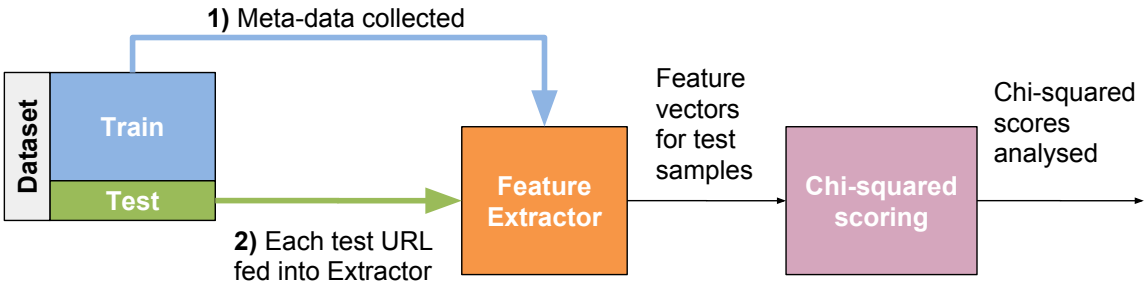

Figure 4.5: The process of evaluating the features, given a data set.

4.4.1 Results

The following is the results of generating chi-squared (χ^2) values (and associated p-values) for all features on both data sets, using 5-fold cross validation. The features are shown in descending order of χ^2 values for the **SBSP** data set.

	SBSP				DP				
Feature	**R**[2]	χ^2	$p-val$	**Sig**[3]	**R**	χ^2	$p-val$	**Sig**	OR[4]
length_hostname	1	124592.575	0.0000	•	64	81.009	0.0890	-	•
path_length	2	66459.618	0.0000	•	1	117296.169	0.0000	•	•
longest_token_hostname	3	59457.743	0.0000	•	52	257.807	0.0000	•	•
length_url	4	59044.710	0.0000	•	2	93576.529	0.0000	•	•
full_host_name_length	5	44372.928	0.0000	•	63	87.783	0.0189	-	•
avg_token_len_hostname	6	31885.265	0.0000	•	46	484.150	0.0000	•	•
longest_token_path	7	27902.769	0.0000	•	5	40012.369	0.0000	•	•
num_non_alpha_url	8	22122.133	0.0000	•	6	34112.026	0.0000	•	•
longest_query_value	9	17157.785	0.0000	•	7	23510.221	0.0000	•	•
max_arg_value_length	10	17157.785	0.0000	•	8	23510.221	0.0000	•	•
longest_token_url	11	15270.532	0.0000	•	15	8424.214	0.0000	•	•
length_querystr	12	15061.192	0.0000	•	3	52973.586	0.0000	•	•
longest_sub_dir	13	14466.720	0.0000	•	4	48717.006	0.0000	•	•
avg_token_len_path	14	11217.758	0.0000	•	9	19516.500	0.0000	•	•
num_slashes_url	15	10109.021	0.0000	•	17	6486.114	0.0000	•	•
token_count_path	16	8773.012	0.0000	•	12	15782.357	0.0000	•	•
shortest_host_token	17	7996.940	0.0000	•	38	724.019	0.0000	•	•
mean_arg_value_length	18	5582.846	0.0000	•	10	16177.028	0.0000	•	•
shortest_path_token	19	4193.956	0.0000	•	16	8249.082	0.0000	•	•
path_sensitive_count	20	4186.583	0.0000	•	18	2928.144	0.0000	•	•
num_dots_url	21	4121.906	0.0000	•	49	373.407	0.0000	•	•
num_dots_host	22	3594.038	0.0000	•	58	166.261	0.0000	•	•
numbers_in_path	23	3533.073	0.0000	•	13	14939.689	0.0000	•	•
length_filename	24	3238.417	0.0000	•	11	16173.162	0.0000	•	•
path_sensitive	25	3070.399	0.0000	•	22	2061.096	0.0000	•	•
sensitive_term_url	26	2639.047	0.0000	•	21	2209.261	0.0000	•	•
url_ngram_1_matches	27	2545.779	0.0000	•	24	1887.563	0.0000	•	•
num_tokens_url	28	2534.364	0.0000	•	14	10200.130	0.0000	•	•
num_params	29	2504.242	0.0000	•	25	1829.896	0.0000	•	•
brand_in_hostname_sans_domain	30	2466.500	0.0000	•	70	42.837	0.0000	•	•
benign_url_ngram_1_matches	31	2142.657	0.0000	•	43	517.214	0.0000	•	•
dots_in_path	32	1853.564	0.0000	•	20	2389.017	0.0000	•	•
brand_in_host	33	1532.595	0.0000	•	74	27.198	0.0002	•	•
phishy_path_model_matches_2	34	1443.244	0.0000	•	19	2602.647	0.0000	•	•
benign_url_ngram_2_matches	35	1417.604	0.0000	•	89	0.755	0.3854	-	•
url_ngram_2_matches	36	1196.771	0.0000	•	45	489.654	0.0000	•	•
out_of_pos_tld	37	1180.834	0.0000	•	47	432.418	0.0000	•	•
max_dots_path_segment	38	1165.009	0.0000	•	23	2032.283	0.0000	•	•
avg_token_len_url	39	1154.173	0.0000	•	51	279.513	0.0000	•	•
avg_token_len_domain	40	806.537	0.0000	•	71	37.387	0.0000	•	•
longest_token_domain	41	782.377	0.0000	•	76	24.132	0.0006	•	•
suspicious_prefix	42	781.160	0.0000	•	32	1050.831	0.0000	•	•
brand_in_path	43	735.735	0.0000	•	34	925.033	0.0000	•	•
organisation_in_path	44	735.735	0.0000	•	35	925.033	0.0000	•	•
path_ngram_1_matches	45	723.937	0.0000	•	44	505.393	0.0000	•	•

[2]Rank within the data set.
[3]Statistically significant, $p < 1 \times 10^{-}3$.
[4]Statistically significant in either data set.

	SBSP				DP				
Feature	**R**	χ^2	$p-val$	**Sig**	**R**	χ^2	$p-val$	**Sig**	OR
url_ngram_3_matches	46	630.605	0.0000	•	55	180.430	0.0000	•	•
target_brand_garera	47	575.053	0.0000	•	36	909.260	0.0000	•	•
digit_letter_ratio	48	571.610	0.0000	•	40	549.524	0.0000	•	•
phishy_path_model_matches_max	49	540.827	0.0000	•	54	224.523	0.0000	•	•
phishy_path_model_matches_3	50	536.180	0.0000	•	31	1185.643	0.0000	•	•
underscore_path	51	474.867	0.0000	•	33	1011.314	0.0000	•	•
subdomain_presence	52	367.011	0.0000	•	39	588.971	0.0000	•	•
host_ngram_1_matches	53	356.762	0.0000	•	80	4.447	0.0443	-	•
embedded_domain	54	341.953	0.0000	•	75	26.600	0.0000	•	•
suspicious_file_name	55	317.319	0.0000	•	59	158.203	0.0000	•	•
phishy_path_model_matches_4	56	312.389	0.0000	•	48	406.359	0.0000	•	•
contains_at	57	300.497	0.0000	•	56	168.800	0.0000	•	•
url_ngram_4_matches	58	275.377	0.0000	•	61	107.283	0.0000	•	•
query_key_ngram_1_matches	59	272.620	0.0000	•	67	73.005	0.0000	•	•
contains_hex	60	258.268	0.0000	•	60	121.121	0.0000	•	•
hyphen_url	61	237.447	0.0000	•	37	862.066	0.0000	•	•
more_than_1_dot_path	62	233.263	0.0000	•	50	373.242	0.0000	•	•
max_num_delim_value	63	206.063	0.0000	•	41	534.141	0.0000	•	•
suspicious_file_ext	64	185.750	0.0000	•	42	525.119	0.0000	•	•
domain_ngram_1_matches	65	178.141	0.0000	•	79	7.662	0.0102	-	•
suspicious_brand_prefix	66	173.577	0.0000	•	62	91.230	0.0000	•	•
num_special_chars_in_path	67	169.703	0.0000	•	30	1309.511	0.0000	•	•
num_delim_fileName	68	128.194	0.0000	•	27	1568.690	0.0000	•	•
path_ngram_2_matches	69	128.092	0.0000	•	78	16.347	0.0001	•	•
query_val_ngram_1_matches	70	122.687	0.0000	•	77	23.977	0.0000	•	•
path_php	71	119.289	0.0000	•	29	1321.762	0.0000	•	•
vowel_cons_ratio	72	85.690	0.0000	•	66	76.783	0.0000	•	•
hyphens_path	73	79.158	0.0000	•	28	1334.333	0.0000	•	•
whitedomain	74	66.906	0.0000	•	72	34.455	0.0493	-	•
contains_ip	75	66.246	0.0000	•	69	49.400	0.0000	•	•
host_ngram_2_matches	76	64.119	0.0000	•	83	2.091	0.2168	-	•
path_ngram_3_matches	77	52.069	0.0000	•	81	4.279	0.0449	-	•
hyphens_host	78	30.259	0.0003	•	57	166.929	0.0000	•	•
host_ngram_3_matches	79	25.497	0.0000	•	85	1.117	0.3978	-	•
ratio_num_to_non_num_url	80	25.252	0.0000	•	53	225.378	0.0000	•	•
path_ngram_4_matches	81	22.556	0.0000	•	86	0.973	0.3394	-	•
domain_ngram_2_matches	82	17.753	0.0000	•	87	0.870	0.3594	-	•
benign_path_model	83	14.219	0.0002	•	91	0.031	0.8661	-	•
brand_in_domain	84	12.377	0.0023	-	65	79.875	0.0000	•	•
num_special_chars	85	5.218	0.1764	-	26	1750.270	0.0000	•	•
contains_port	86	3.984	0.3076	-	82	3.560	0.0948	-	-
host_sensitive	87	2.592	0.4291	-	68	67.630	0.0000	•	•
host_ngram_4_matches	88	1.887	0.1780	-	88	0.863	0.4034	-	-
token_count_domain	89	1.344	0.2602	-	84	1.835	0.2182	-	-
num_hyphens_domain	90	1.025	0.3631	-	73	31.722	0.0000	•	•
domain_ngram_3_matches	91	0.234	0.6332	-	90	0.150	0.7138	-	-
domain_ngram_4_matches	92	0.034	0.8594	-	92	N/A	N/A	N/A	N/A
slash_redir	93	N/A	N/A	N/A	93	N/A	N/A	N/A	N/A

Table 4.5: Results of evaluating feature performance. A green cell with the • symbol indicates those that have a statistically significant ability to distinguish between phishing and benign URLs for the particular data set.

4.4.2 Discussion

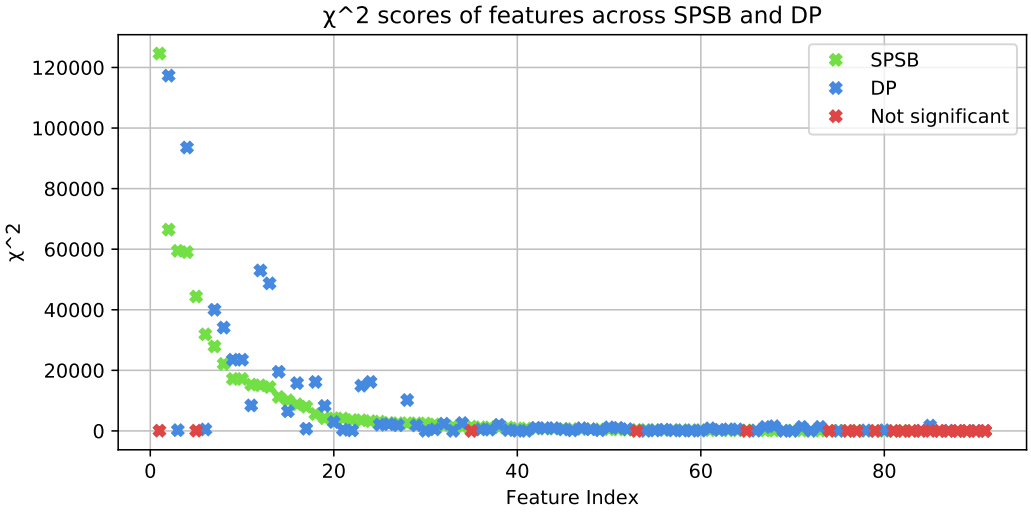

Figure 4.6: χ^2 scores of features tested on each data set. Scores which were found to be statistically significant are shown in green and blue for the SBSP and DP data sets, respectively.

Figure 4.6 shows χ^2 scores of features plotted against their respective indexes for both data sets. Scores that were not statistically significant ($p >= 0.001$) are shown in red. The index assigned to a feature reflects its performance ranking in the SBSP data set, therefore a feature with index 1 indicates that it is the strongest feature for the SPSB data set. 93.5% of features can provide statistically significant evidence for classification in at least 1 data set, with 76.3% of these features being able to do so for both sets. The fact that the scores for the DP set follows the scores for the SBSP set closely indicates that majority of the features are similarly important in both data sets.

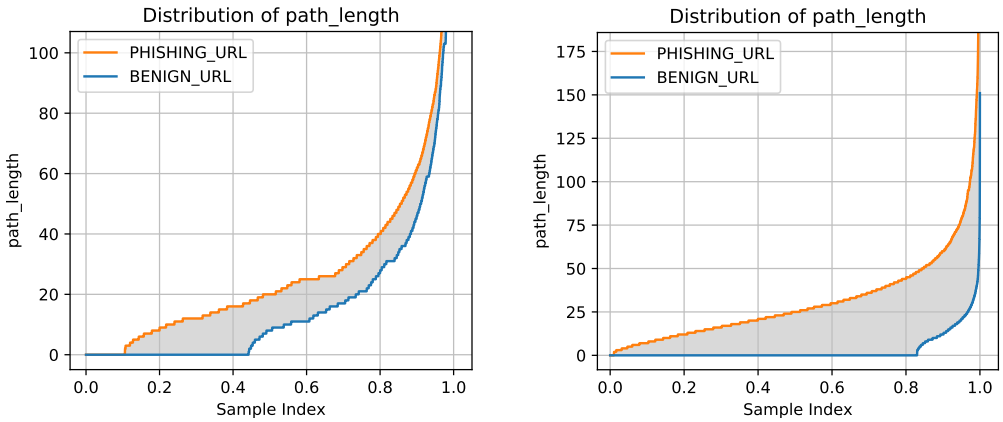

Figure 4.7: Distributions of values for `path_length` of each sample in SBSP (left) and DP (right).

The feature `path_length`, for example, is ranked top for the SBSP data set, and second for the DP data set. The discriminatory ability is evident from Figure 4.7, which highlights the difference in distributions of values for each data set. The difference in area under the two curves (shown by the

area shaded in gray) indicates the degree to which the values of the feature differ between the classes. However, it is important to note that the curves on the graph for SBSP on Figure 4.7 are closer, and all the SBSP URLs are sourced from a communication channel where phishing is prevalent (Fette et al., 2006), whereas the benign URLs for DP are sourced from DMOZ (DMOZ, 2016). This therefore raises concerns regarding the results reported by studies where benign URLs were sourced solely from DMOZ and sources alike (Darling et al., 2015, Feroz and Mengel, 2015, Huang et al., 2012, Le et al., 2011, Ma, Saul, Savage and Voelker, 2009*a*).

Some features that were reported in literature as being effective, however, were not found to provide statistically significant evidence for classification. `contains_port`, for example, is a binary feature indicating if the URL contains a port number after the domain, and only was true for 0.25% and 0.8% of phishing URLs in SBSP and DP respectively. The double-slash redirection technique (`slash_redir`) used by Mohammad et al. (2015) were not found in any URLs at all.

Other features found to be ineffective are: `brand_in_domain`, `num_special_chars`, `host_sensitive`, and `num_hyphens_domain`. It is interesting to note that while `brand_in_domain` was found not to be statistically significant, `brand_in_host` is ranked 33/93 and 74/93 for SBSP and DP respectively. This is most likely due to the fact that domain registrars can monitor domain registrations for the presence of brand names, especially brands that are targeted frequently, whereas inserting a brand in other regions of the URL (subdomain, path) is not susceptible to this and is only revealed once the URL has been distributed.

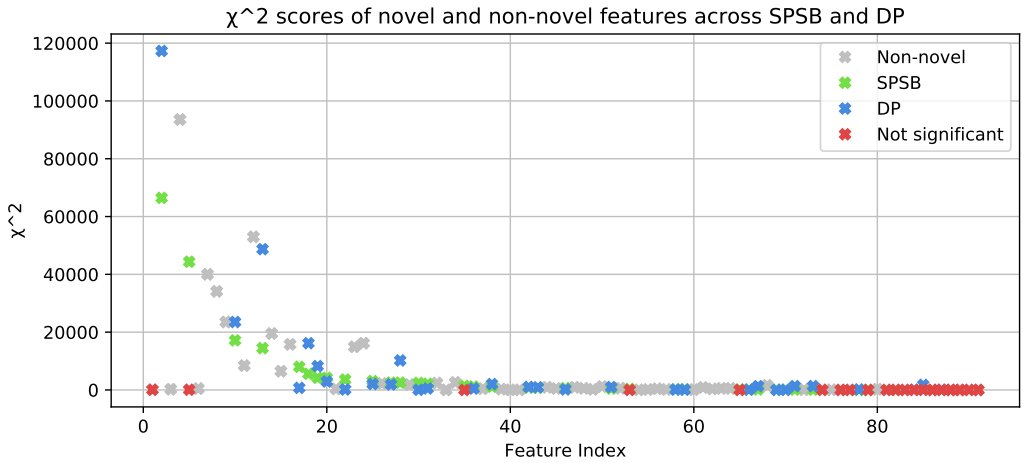

Figure 4.8: χ^2 scores of features tested on each data set. Statistically significant scores for *novel* features are shown in colour, whereas non-novel features are shown in grey.

Figure 4.8 highlights the performance of novel features, amongst non-novel features (shown in grey). 85% and 77.5% of novel features were found to be effective with statistical significance for the SBSP and DP data sets respectively. 92.5% were effective for at least one data set, and 50% of these were ranked in the top 50% features for the SBSP data set. 70% were effective in both data sets, and 45% of these were in the top 50% ranked features for the SBSP data set. This indicates that the novel features are in fact useful for classifying phishing features, and in some instances more effective than many features found in existing literature.

4.4.2.1 Novel Token-frequency-based Features

Most of the token-frequency-based features are effective for classification. Unigrams extracted from full phishing URLs (`url_ngram_1_matches`) were most effective, ranked 27/93 for SBSP and 24/93 for DP. The distribution graph for values of this feature can be seen in Figure 4.9.

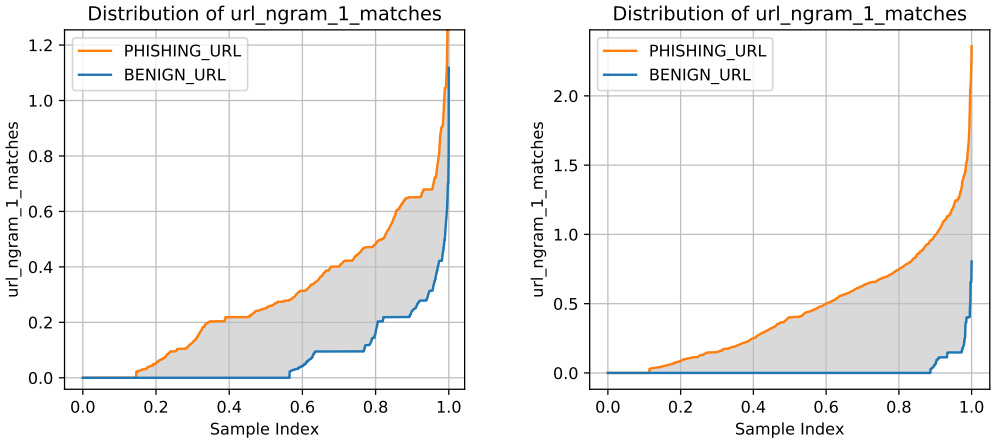

Figure 4.9: Distributions of values for `url_ngram_1_matches` (unigrams from phishing URLs) of each sample in SBSP (left) and DP (right).

Another particularly effective feature in this category is `phishy_path_model_matches_2`, which use bigrams of path model symbols from phishing URLs. This feature was ranked 34/93 for SBSP and 19/93 for DP, therefore indicating that modelling the path of URLs based on the nature of the characters in segments is useful. Figure 4.10 shows the distribution for this feature's values.

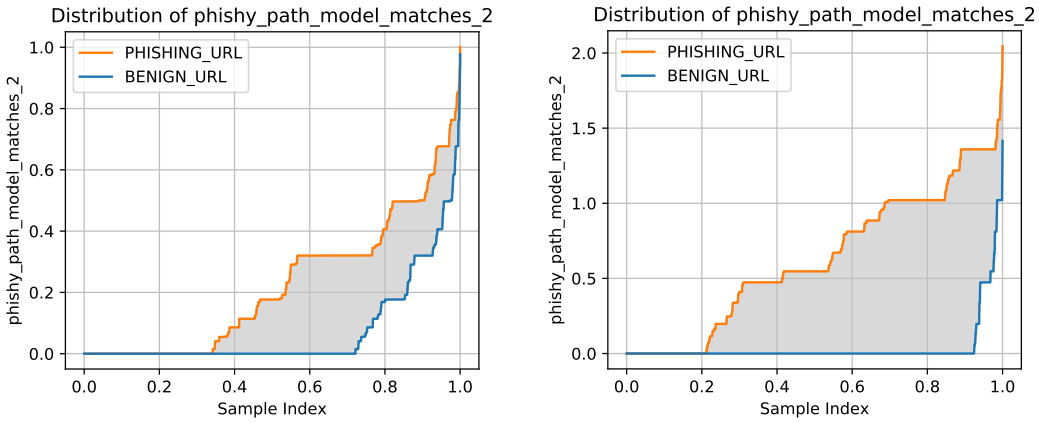

Figure 4.10: Distributions of values for `phishy_path_model_matches_2` (bigrams of path model symbols) of each sample in SBSP (left) and DP (right).

Token-based features which have not been found to be effective are: `host_ngram_4_matches`, `domain_ngram_3_matches`, `domain_ngram_4_matches`. This is due to the fact that the regions from which the n-grams are sourced from often do not contain enough tokens to create n-grams of size 3 and 4, and so these n-grams will rarely be found in out-of-sample data.

4.4.2.2 Other Novel Features

Aside from the token-frequency features, the most effective feature for the SBSP data set was `max_arg_value_length`, ranked 10/93, followed by `longest_sub_dir`, ranked 13/93. From Figure 4.11, it can be inferred that the longest query value found in benign URLs are generally longer than those found in phishing URLs. It can also be inferred that the longest subdirectory in phishing URLs are generally longer than those found in benign URLs. A comparison for the DP data set is not made due to the lack of representative benign URLs, as described earlier in this chapter.

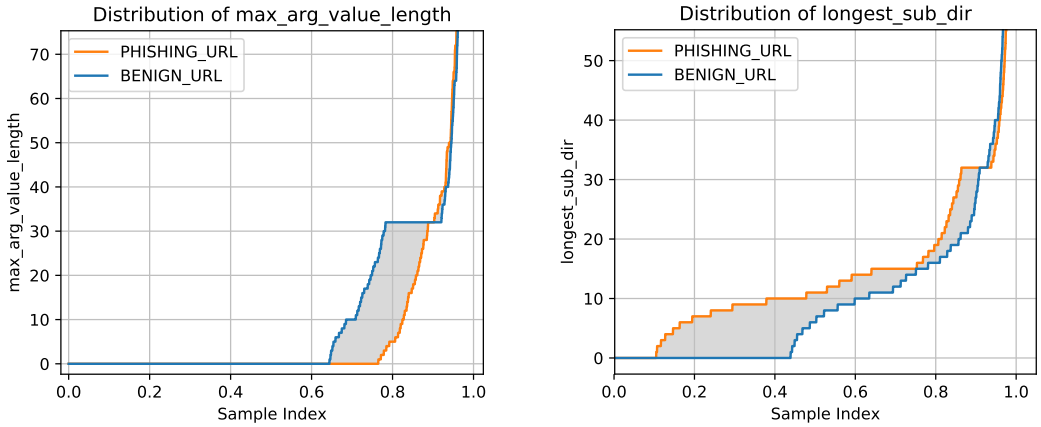

Figure 4.11: Distributions of values for `max_arg_value_length` and `longest_sub_dir` for the SBSP data set.

4.5 Selected Features

With empirical values for feature performance, ineffective features can be discarded, and the feature vector can be reduced to only those features considered "useful" for classification. The initial step in this process involved selecting the subset of features that provided statistically significant evidence for distinguishing between phishing and benign URLs. These features are shown with the • character in the column **Sig** for SBSP in Table 4.5 - a total of 83 features.

The next step was to chose features that may not have been significant for the SBSP data set, but were significant for DP. It is important to consider why these features may not have been significant for SBSP - it may be due to the poor quality of benign URLs in DP, or simply due to the fact that the nature of phishing attacks included in DP are not present in SBSP. These features are: `brand_in_domain`, `num_special_chars`, `host_sensitive`, `num_hyphens_domain`. It was decided that the features should be included as they do not take advantage of the primary deficiency of the DP data sets, i.e. the lack of path components on many benign URLs.

Furthermore, these features may have been highlighted as insignificant for the SBSP data set due to the phishing URLs being sourced from different locations, and therefore the phishing attacks may have varied in nature. Ignoring the indicators that are useful for DP may mean that particular phishing attacks will be ignored by a classifier in the future. In addition, many classification algorithms incorporate feature weighting, and therefore low performing features will have a smaller impact on the classification decision, mitigating possible negative effects of their inclusion.

The final *optimal feature vector* is comprised of features that are have a • character in the column **OR** in Table 4.5 - a total of 87 features.

5 Optimal Classification Algorithm

Section 2.6.2 identified multiple classification algorithms that are prominent in the phishing-classification space. This investigation will aim to identify the best performing classification algorithm, using the SBSP data set and the *optimal feature vector* established in Chapter 4.

The process of this investigation will require an initial selection of algorithms that have been reported as being suitable for phishing classification. As each classification algorithm relies on a number of hyperparameters which influence the construction of the model, hyperparameter tuning is undertaken to identify values for these parameters that maximize classification performance.

With these results, an inter-algorithm comparison can subsequently take place between the algorithms (with their optimal hyperparameters), finding the best-performing algorithm. In order to rank the classifier instances on performance, F1-score and AUC will be used. F1-score weights Precision and Recall equally, which is advantageous as neither of these metrics are significantly more important than the other. The ROC AUC indicates the trade-off between false positives and true positives when adjusting thresholds. A higher TPR comes at the cost of a higher FPR, though a larger ROC AUC indicates that the cost of FPR to obtain a high TPR is lower.

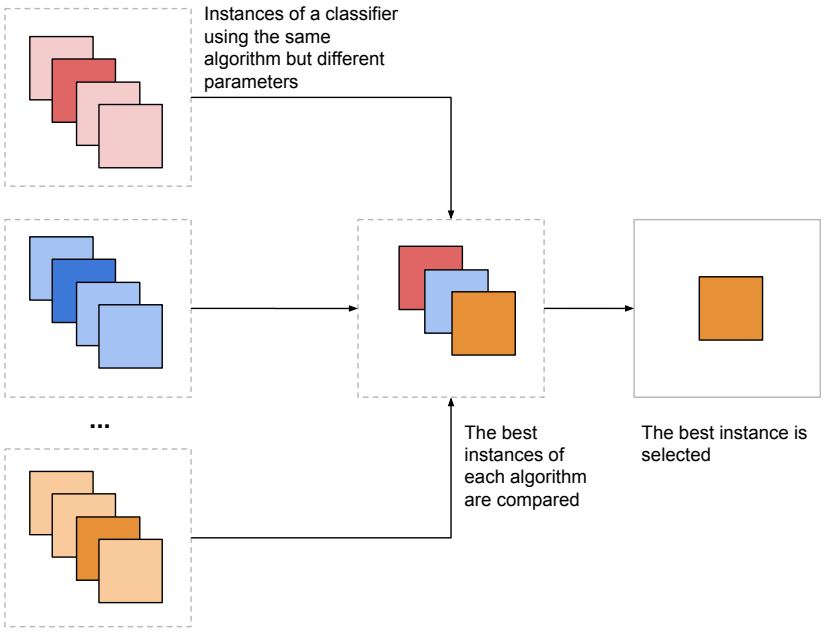

Figure 5.1: The process of selecting the optimal classification algorithm and parameters.

All classifiers were instantiated in Python (64-bit, v2.7) (Python Software Foundation, 2017*a*) using the scikit-learn library (scikit-learn, 2017*a*) (excluding XGBoost which used a separate open-source implementation (DMLC, 2017)). Each classifier was evaluated on the SBSP data set, using 5-fold cross validation.

5.1 Algorithm Selection

All algorithms identified and described in Section 2.6.2 were chosen for the investigation, excluding Naive Bayes and SVM due to their poor performance in this domain (Abu-Nimeh et al., 2007, Gyawali et al., 2011, Ma, Saul, Savage and Voelker, 2009*a*, Miyamoto et al., 2008).

In addition to those, a further classification algorithm known as XGBoost was included. Whilst this algorithm is similar to AdaBoost and Random Forest (See Section 2.6.2) in the sense that an ensemble of multiple trees are created, the method of training the classification model is different. Like Adaboost, the model is created by creating multiple trees consecutively, each next tree aiming to improve classification performance. The difference, however, arises when deciding how to split at new nodes. AdaBoost weights previously misclassified samples higher, whereas gradient boosting aims to minimize an objective function (Chen, 2014). The objective function is a sum of training loss and model complexity, and minimizing this indicates a model that can represent the training data well but is not overfitted. Trees are grown in a greedy-fashion such that at each node, the possible splits are evaluated in terms of impact to the objective function and the best split is chosen (Chen, 2014). XGBoost has been applied to a wide variety of problems successfully - in 2015, 60% of winning solutions at Kaggle (Kaggle, 2017) used XGBoost (Chen and Guestrin, 2016), indicating its potential suitability for this domain.

5.2 Hyperparameter Tuning

The methodology used to find the optimal hyperparameters follow the steps outlined by Hsu et al. (2003), and is summarized below:

1. The hyperparameters to be tuned are chosen. A set of values for each hyperparameter is decided, and hence a "hyperparameter space" is created. As the number of hyperparameters dictates the dimensions of the "hyperparameter space", thus only the most important hyperparameters are used to keep computation costs minimal.

2. An objective function is chosen. Hsu et al. (2003) use "Accuracy" as this measure, though the disadvantages of using accuracy have been discussed in Section 2.6.1. In this case, the F1-score, the harmonic mean of Precision and Recall, is used instead. Maximisation of both of these metrics are desirable for classification performance.

3. Models are trained using every combination of the hyperparameters in the "hyperparameter space", and the combination which maximises the objective function are identified as the "optimal" hyperparameters. This form of exhaustive search is known as *grid-search*. Hsu et al. (2003) describe grid-search as being naive, yet easily parallelised as each combination is independent, and therefore suitable for this problem. It is important to note that sometimes multiple permutations of parameters will yield maximums, and therefore a second goal can be used to order these. In this experiment, the permutation that requires the least computation time will be chosen.

The above processes was undertaken for all algorithms using 5-fold cross validation and 12.5% of SBSP data set (12,500 benign samples and 12,500 phishing samples). Only a fraction of the data set was used to reduce the overall time required for this process. In addition to th F1-score, the mean time taken to classify all the test samples in a fold was recorded (`mean_time_taken`), to understand the impact of the hyperparameters on classification time.

5.2.1 Random Forest

Hyperparameter tuning of the Random Forest model involved varying the number of trees (`num_trees`) in the ensemble, and the maximum number of features (`max_features`) used when finding the best split at a node. A large value for `max_features` would lead to little variance amongst the trees, which is not beneficial as only some of the patterns in the data set will be reflected. Breiman (2003) suggests using the $\sqrt{n}$ for the `max_features` (where n is the total number of features), which translates to around 10% for this feature set. This parameter was subsequently varied from 2% to 60%.

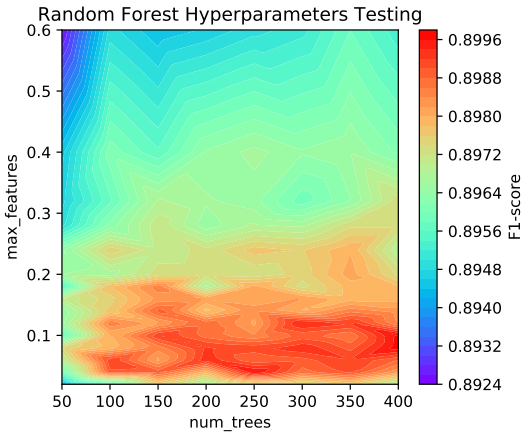

Figure 5.2: The F1-scores achieved when varying the number of trees and maximum number of features.

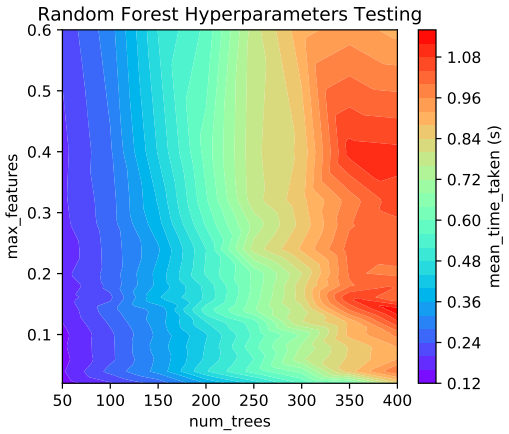

Figure 5.3: The mean classification time for one fold when varying the number of trees and maximum number of features.

The results support the recommendation by Breiman (2003) for the `max_features` property, as Figure 5.2 shows the highest F1-scores being observed around 10% of the features. Figure 5.3 also indicates that classification time increases as the number of trees increases. Using the recommendation of Breiman (2003) for `max_features`=$\sqrt{n}$, and with the aim to maximise F1-score with as few as possible trees, the optimal parameters are found to be: `num_trees`: 150, and `max_features`: 10% ($\sqrt{87} \approx 9$ features (rounded) $\approx 10\%$).

5.2.2 AdaBoost

The number of trees (`num_trees`) was varied for this algorithm from 50-2000 trees, and it can be seen that the F1-score is maximised at *1600 trees*.

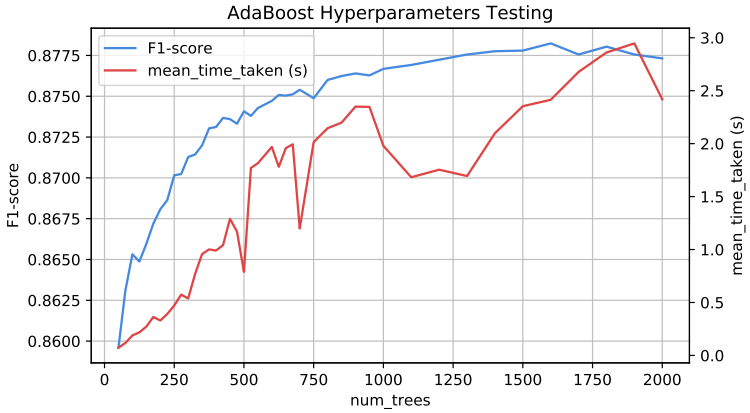

Figure 5.4: The performance obtained when using AdaBoost with 50-2000 trees.

5.2.3 Multi-layer Perceptron (Neural Network)

MLP is a forward-propagating Neural Network, as used by Abu-Nimeh et al. (2007). This experiment will use an MLP with one hidden layer, and vary the number of nodes in the hidden layer (`num_nodes`) as well as the activation function (`activation`).

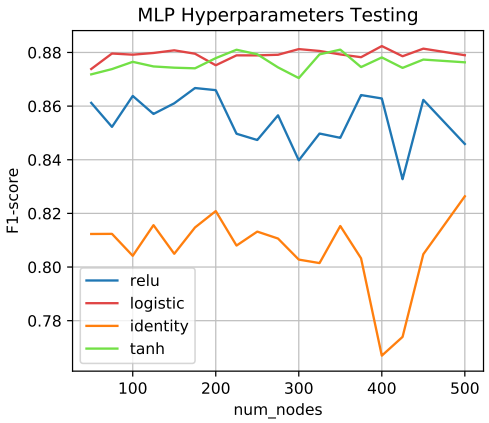

Figure 5.5: The F1-scores achieved when varying the number of nodes and activation function.

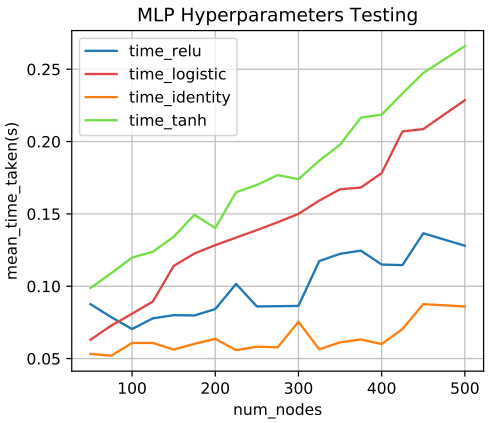

Figure 5.6: The mean classification time for one fold when varying the number of nodes and activation function.

Figure 5.5 indicates that the highest F1 scores were achieved with the *tanh* and *logistic* activation functions, with minimal increases in performance as the number of nodes increased. The classification time, however, increased linearly for both of these functions as the number of nodes was increased. The *logistic* function is more consistent, with the highest performance observed at 400 nodes. While the high number of nodes indicates a higher classification time, the classification time at 400 nodes is still significantly lower than when compared to other algorithms analysed in this section. The optimal parameters are subsequently chosen as: `activation:` *logistic* and `num_nodes:` *400*.

5.2.4 Logistic Regression

The hyperparameter chosen for Logistic Regression was the method of *regularization*, which is used to set the internal weighting of features. Figure 5.7 shows that L1 regularization achieves a marginally higher score than L2 regularization, with very minimal differences in classification time.

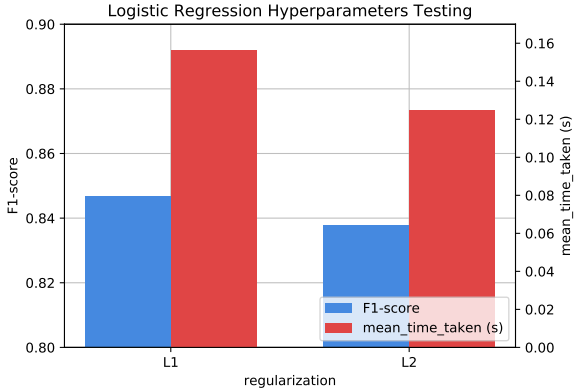

Figure 5.7: The process of selecting the optimal classification algorithm and parameters.

5.2.5 XGBoost

The hyperparameters chosen for tuning for XGBoost are the number of trees (`num_trees`, 50-450) and the maxmimum tree depth (`max_depth` 2-10). Increased tree depth leads to a more complex model, though is prone to overfitting.

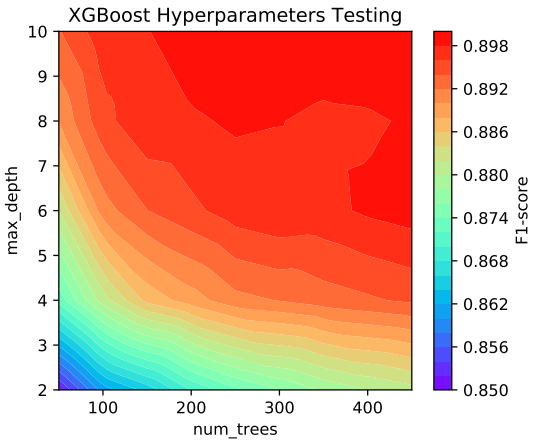

Figure 5.8: The F1-scores achieved when varying the number of trees and tree depth.

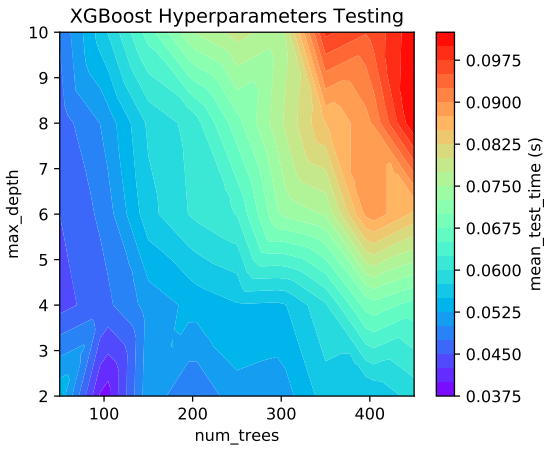

Figure 5.9: The mean classification time for one fold when varying the number of trees and tree depth.

Figure 5.8 shows that the F1-score increases as both `max_depth` and `num_trees` are increased, however the gains begin to diminish from around 250 trees and a max depth of 6. Figure 5.9 indicates that the number of trees is the biggest factor for increasing computation time, as the pattern is more of a horizontal gradient than a vertical one. It is therefore desirable to have a lower number of trees but a larger value for `max_depth`. The optimal parameters are decided to be *250 trees* and a *max depth of 9*, as this ensures the classification time is low but F1-score is in the region where any further increases are insignificant. The fact that the F1-score only increases as the parameters are increased indicates that overfitting is not an issue with these parameter values.

5.3 Inter-algorithm Comparison

Each algorithm with optimal hyperparameters (as identified in Section 5.2) was trained and tested on the full SBSP data set using 10-fold cross validation. The Accuracy, Precision, Recall, F1-score, and ROC AUC was recorded.

5.3.1 Results

Algorithm	Hyperparameters	Accuracy	Precision	Recall	F1-score	AUC
Random Forest	`num_trees`: 150 `max_features`: *sqrt*	0.914865	0.905003	0.927040	0.915889	0.9701
AdaBoost	`num_trees`: 1600	0.878935	0.876708	0.88189	0.879292	0.9490
MLP	`num_nodes`: 400 `activation`: *logistic*	0.89541	0.89547	0.89533	0.8954	0.9582
Logistic Regression	`regularization`: *L1*	0.83877	0.86403	0.80408	0.83298	0.9167
XGBoost	`num_trees`: 250 `max_depth`: 9	0.913435	0.906864	0.92151	0.914128	0.9708

Table 5.1: The results of the inter-algorithm comparison. Related confusion matrices can be found in Appendix B.

5.3.2 Discussion

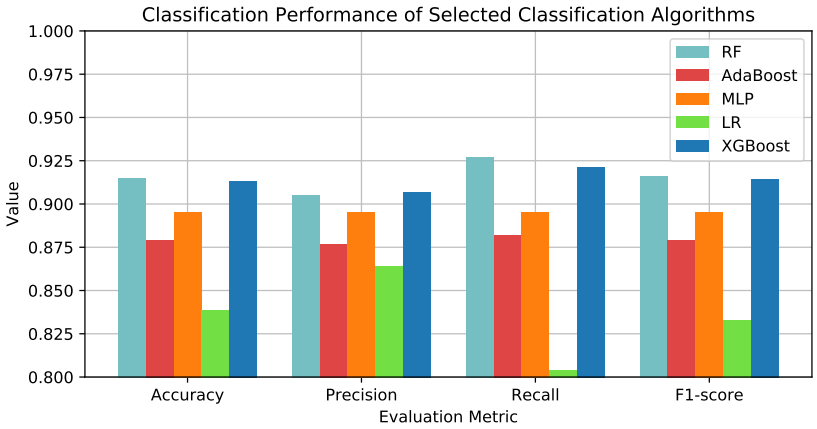

Figure 5.10: The performance obtained when testing all algorithms on the SBSP data set using 10-fold cross validation.

The results show that Random Forest and XGBoost perform the best in terms of the evaluation metrics used, with Random Forest outperforming XGBoost marginally on Accuracy (91.49% vs 91.34%), Recall (92.70% vs 92.15%) and F1-Score (91.59% vs 91.41%). The differences in performance are very minimal, in the order of 10^{-3}, and therefore this indicates that either algorithm would be suitable for building a classification system. Logistic Regression, however, performed the worst in all metrics, with a significantly lower value than other algorithms for Recall and Accuracy.

It is important to note that Logistic Regression assumes that the classes are "linearly separable" (Ng and Jordan, 2002), whereas tree-based classifiers do not. The poor performance of the Logistic Regression model may indicate that the the benign and phishing URLs with features used in this study are *not* linearly separable, hence the disparity in performance with the other models.

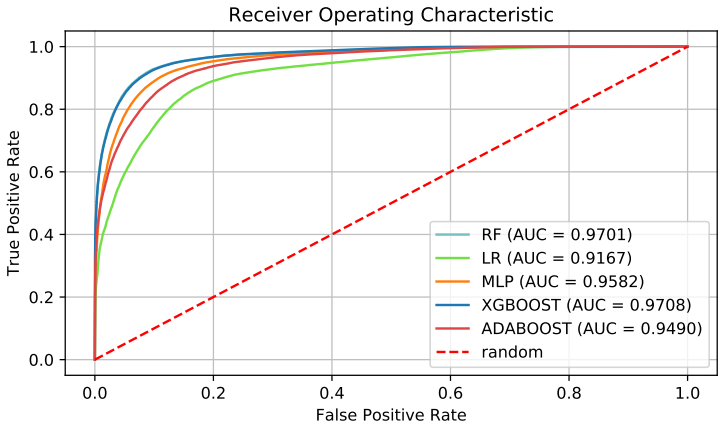

Figure 5.11: ROC curves for the selected algorithms. It is important that the curve for RF lies underneath XGBOOST.

A similar observation can be found when analysing the ROC curves for each of the algorithms (shown in Figure 5.11), with Random Forest and XGBoost achieving the greatest AUC (0.9701 and 0.9708 respectively). Logistic Regression performed the worst, again, for this metric with an AUC of 0.9167.

The high values for Random Forest and XGBoost indicate that using these algorithms involves the least trade-off between True Positives and False Positives when adjusting the decision threshold. Varying the decision threshold will yield different ratios of True Positives to False Positives, though for these two algorithms it is evident that obtaining a high True Positive Rate (e.g. 90%) can be done so while maintaining a low False Positive Rate (7%).

5.4 Comparison Against Existing Studies

As the Random Forest model was shown to perform marginally better than the XGBoost model, a comparison against existing studies was performed. The methodology for this comparison involved creating data sets that best matched the data sets used in each study. Efforts were made to collect the URLs from the same source, and the training and testing process was matched in terms of number of URLs used in each phase and the number of folds used during k-fold cross-validation.

In each comparison, the same evaluation metrics that were used in the study were collected, and the nature of the features used was also noted (*full* vs *lexical*). Each study was also given a "Dataset Weight", which indicates the total number of URLs used to train the classification model. Details and raw results of comparisons can be found in Appendix C. If the training-test split was unknown, then 10-fold cross validation was used.

5.4.1 Accuracy

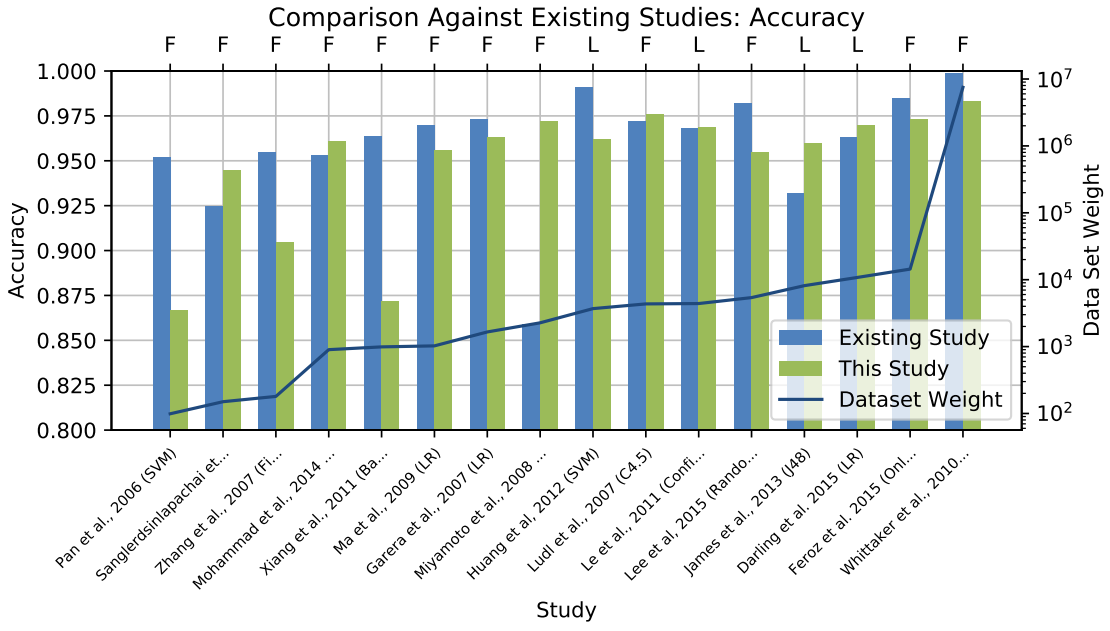

Figure 5.12: A comparison against the accuracies observed in existing studies, ordered by Dataset Weight. Classifiers that use *full* features are denoted by F, whereas *lexical* classifiers are denoted by an L.

While accuracy can often be misleading (Batista et al., 2004), it was the most widely reported statistic, being reported by 16 studies. Figure 5.12 shows that the Random Forest model performed better than 7 studies, where 4 of these studies used *full* features. This supports the notion that lexical classification is competitive with full-feature classification, as reported by Le et al. (2011).

3 out of 4 lexical classifiers were outperformed, which shows that the feature vector used in this study is in fact effective. A possible explanation for the lower accuracy when compared with Huang et al. (2012) is that Huang et al. (2012) focused on using brand-specific features, which made up a large portion of their feature vector (43%). 10 most targeted brands were identified in their analysis of their phishing set obtained from PhishTank, and these were used when constructing their feature vector. While a very high accuracy was achieved (99%), the heavy dependence on the top 10 brands (which are prominent in this feed) may lead to a much worse performance when tested against another feed where such brands are not so prominent. The advantage of using the classifier in this study is that it is not so tightly-coupled with a specific set of brands, and therefore its generic nature is favourable when classifying other feeds.

Other instances where the Random Forest model was outperformed by other studies and substaintial training data[1] was involved are in the comparisons against Garera et al. (2007) (-1%), Lee et al. (2015) (-2.7%), Feroz and Mengel (2015) (-1.2%) and Whittaker et al. (2010) (-1.6%). It is important to note that these classifiers are full-featured classifiers, and that the differences in accuracy are in fact minimal. When the benefits of lexical classification are taken into account (low-latency, low-memory and less susceptibility to methods of evasion), certain use-cases may find lexical classification much more desirable despite the trade-off in accuracy.

It can therefore be concluded that in terms of accuracy, this study is able to outperform state-of-the-art lexical classifiers and some full-featured classifiers, indicating that the novel features are a useful contribution to the field of phishing classification.

5.4.2 F1-score

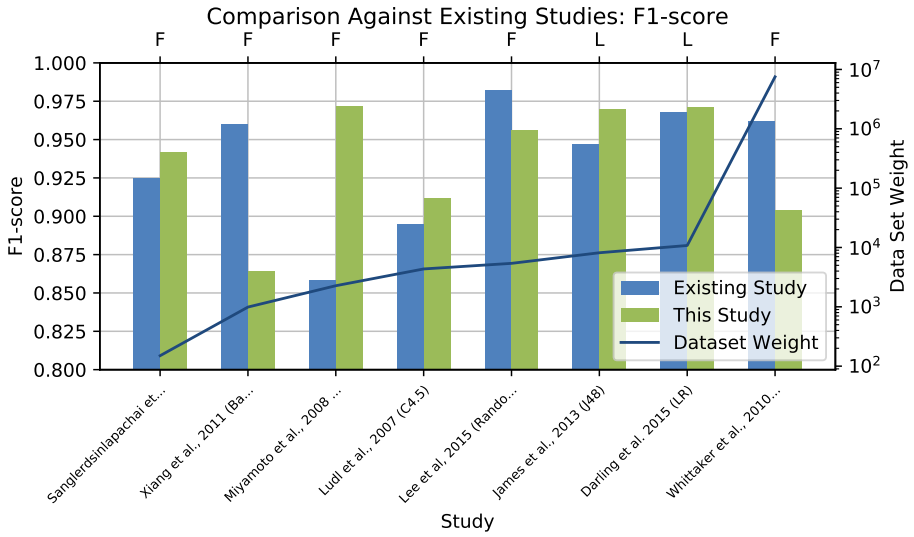

Figure 5.13: A comparison against the F1-scores observed in existing studies. Classifiers that use *full* features are denoted by F, whereas *lexical* classifiers are denoted by an L.

The F1-score, as explained in Section 2.6.1, is a balanced mean of precision and recall. From Figure 5.13, it is evident that this study outperforms 4 out of 8 classifiers for which F1-score was reported or could be calculated. When restricted to training data of size $>$ 1000 URLs, it is only outperformed by two studies, by Lee et al. (2015) (2.6%) and Whittaker et al. (2010) (4.8%), both of which use *full*-features.

[1] $>$ 1000 URLs

5.4.3 ROC AUC

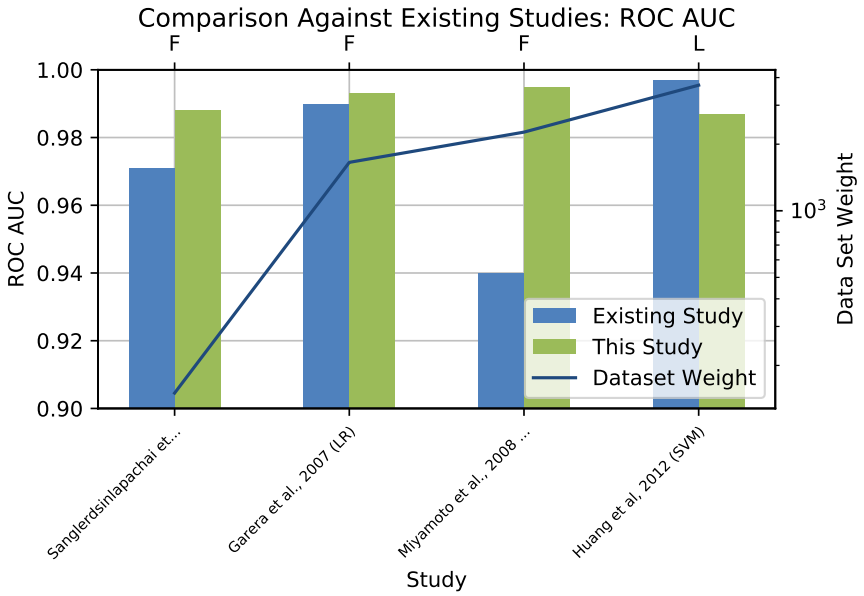

Figure 5.14: A comparison against the ROC AUC observed in existing studies. Classifiers that use *full* features are denoted by F, whereas *lexical* classifiers are denoted by an L.

The ROC AUC comparison revealed that this study was able to outperform all other studies but one, Huang et al. (2012), which achieved a greater AUC by 0.01. While Huang et al. (2012) is indeed a lexical classifier, the increased performance may again be due to the overfitting with the 10 particular brands that Huang et al. (2012) focused on, and therefore the 0.01 trade-off in AUC may be acceptable for increased ability to apply the classifier to a bigger variety of URLs.

5.4.4 Recall

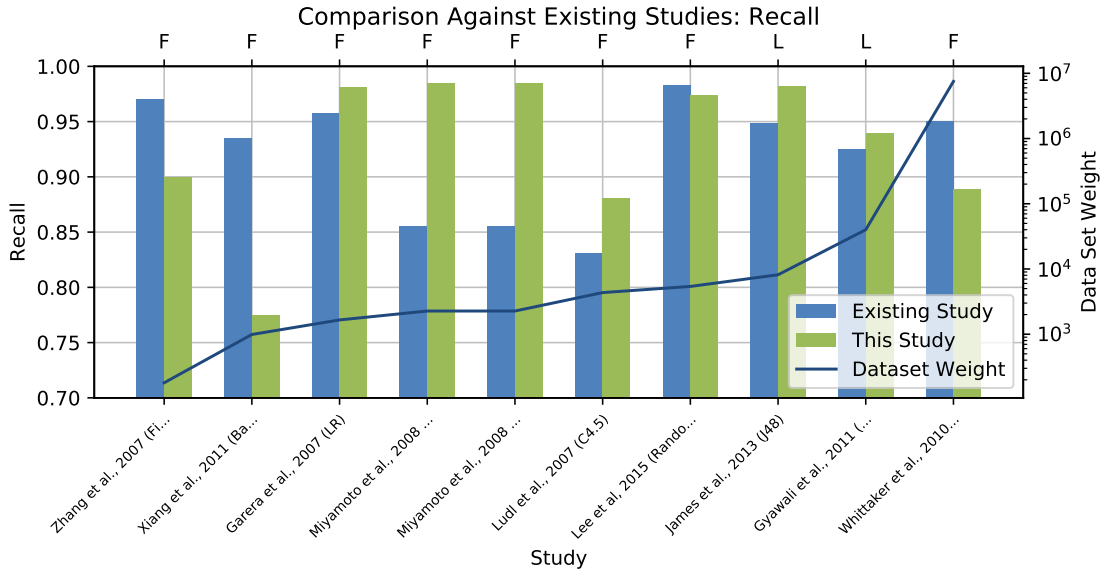

Figure 5.15: A comparison against the recall observed in existing studies. Classifiers that use *full* features are denoted by F, whereas *lexical* classifiers are denoted by an L.

Of the 10 classifiers for which recall was reported, 6 were outperformed which included the lexical classifiers built by James et al. (2013) and Gyawali et al. (2011). Other lexical classifiers did not report recall and therefore a comparison could not be performed. Recall is an important metric as it can indicate the portion of phishing URLs that are misclassified as benign and therefore remain undetected. Instances this study achieved a lower recall are explored below:

- Zhang et al. (2007) and Xiang et al. (2011): Both of these studies used low amounts of training data (100 and 990 samples respectively). On a lexical level, this number of samples is too low to exhibit the variation present in URLs.
- Lee et al. (2015): A minimal difference of 0.9% was observed, possibly due to the extra information available when using a full-featured classifier.
- Whittaker et al. (2010): The difference in recall may be attributed to a lack of data to match the data set used by Whittaker et al. (2010). While the class imbalance was preserved, the data set used in this comparison was 70 times smaller.

5.4.5 Precision

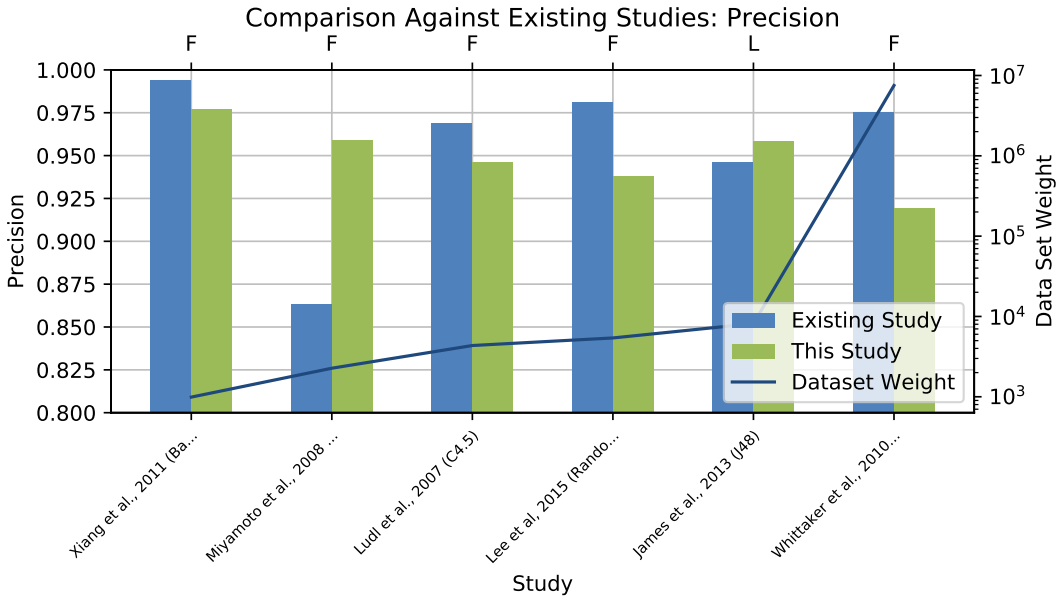

Figure 5.16: A comparison against the precision observed in existing studies. Classifiers that use *full* features are denoted by F, whereas *lexical* classifiers are denoted by an L.

This study outperformed the the lexical classifier built by James et al. (2013), which was the only lexical classifier for which precision was reported. From the remaining classifiers for which precision was reported, only 1 out of 4 was outperformed. The reduced performance in precision indicates that the benefits of lexical classification come at a cost of increased misclassifications of benign URLs as phishing. In particular workflows such as a multi-stage classifier, this may not be significantly detrimental as further classification processes may be able to better identify false positives.

5.5 Resistance to Evolution

The evolution of phishing attacks raises a concern regarding the use of a static model for classification. Random Forest is a batch-learning algorithm, where the same model is used to classify samples each time. Online-learning algorithms, however, are suitable for tackling this problem as they support continuous retraining, enabling the model to be incrementally modified. Each new sample can be used to alter the classification model, without having to rebuild it. Ma, Saul, Savage and Voelker (2009*b*) state that continuous retraining is critical to being able to keep up with evolving phishing attacks.

Existing studies that address the issue of retraining currently identify a training regimen whereby a batch-learning classification model is retrained daily with new data. Xiang et al. (2011) use a two-week sliding window for retraining their batch-learning model, i.e. retraining on day T will use training data from day $T - 14$ to T. A similar approach was also taken by Ma, Saul, Savage and Voelker (2009*b*), whereby the classification model was retrained daily based on samples from the previous 14-17 days. When no retraining was performed, Ma, Saul, Savage and Voelker (2009*b*) identified a progressively worse classification rate every next day.

An experiment was undertaken to identify the susceptibility of the model built in this study to evolution. Phishing URLs were collected for a period of 50 days, from 15-01-2017 (t) to 06-03-2017 (t'), and were separated into individual daily sets. An initial training set of 16,750 phishing URLs (from PhishTank (2017)) and 16,750 benign URLs (from the set spam_benign) was also constructed, whereby the phishing URLs range from attacks observed during 2016 through to the 14-01-2017 inclusive. Benign data was kept static based on the findings of Darling et al. (2015) that benign URLs change at a much slower rate than phishing URLs.

The daily data sets were tested by using two distinct training schemes - *Train-once* and *Train-daily*. In the *Train-once* scheme, the model was trained with the original training data, and the error rates of classifying each daily set were recorded. In the *Train-daily* scheme, before classifying the set for day T, the classifier was retrained with a composite data set of the initial training data and all the URLs collected on the days that fall into the interval $t > x < T$.

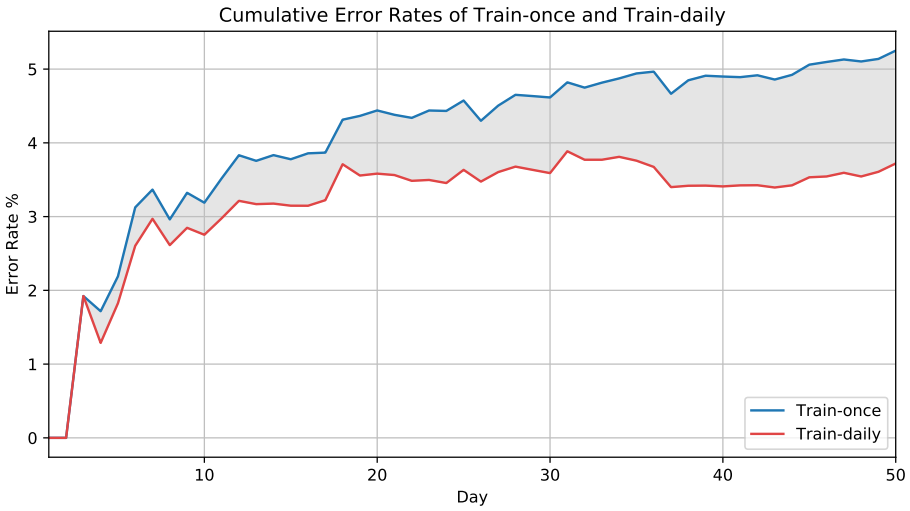

Figure 5.17: Cumulative error rates for Train-once and Train-daily over a period of 50 days.

Figure 5.17 shows the difference obtained when using Train-once and Train-daily over a period of 50 days. The mean difference in error rate between the Train-once and Train-daily was 1.48% ($\sigma = 1.50$), and the final difference in CER was 1.53% - i.e. the cost of not performing daily retraining over a period of 50 days is that 1.53% of phishing URLs are misclassified as benign.

Given the low-memory nature of working with lexical features, retraining frequently is more feasible than when using full-features, allowing the model to stay up-to-date with the evolving trends in URL features.

5.6 Summary

The experiments undertaken to find the optimal classification identified two algorithms (Random Forest and XGBoost) that had the highest performance, when evaluated using appropriate measures (Accuracy, Precision, Recall, F1-score and ROC AUC). The optimal hyperparameters which maximized the evaluation metrics were also identified, and can be found in Section 5.3. Random Forest outperforms XGBoost marginally, and a comparison against 21 existing studies identifies that this study outperforms the state-of-the-art lexical classifiers, as well as many classifiers that use full-features.

Instances where the classifier performs worse than existing studies show that the differences in performance are minimal, and could be an acceptable trade-off given the benefits that are present when using lexical classification over full-feature classification.

Finally, using a daily retraining scheme, similar to that proposed by (Ma, Saul, Savage and Voelker, 2009*b*, Whittaker et al., 2010, Xiang et al., 2011), can reduce the cumulative error rate of phishing URLs by 1.51% over a period of 50 days.

6 Proof of Concept: Poseidon

With the optimal feature vector and classification algorithm identified, a distributed system (Poseidon) was built to filter a high-volume stream of URLs in order to reduce the load on a more expensive classification process (Stage 2) operated by Netcraft (Netcraft, 2017*b*). The results of classification are subsequently fed to a large range of organisations, from companies behind major web-browsers, anti-virus products and firewalls to ISPs and domain registries.

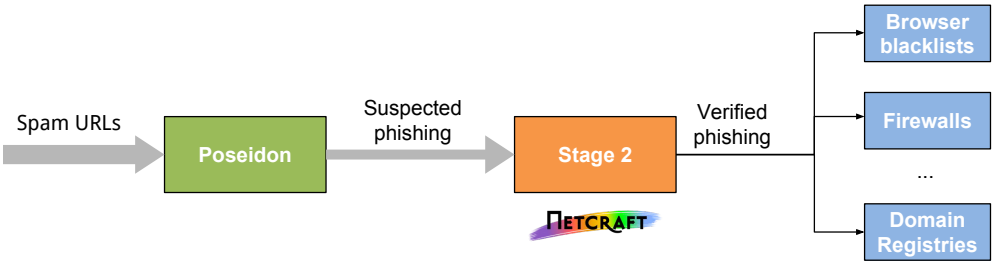

Figure 6.1: The role of Poseidon in identifying suspicious URLs.

Figure 6.1 shows the pipeline in which Poseidon sits. The incoming stream of URLs has an average volume of 26 URLs/s, though can reach up to 1000 URLs/s at peak times. Similar to the observations of Gyawali et al. (2011), a large class imbalance exists between phishing and non-phishing URLs within the feed, and therefore the volume of the feed from Poseidon to Netcraft is expected to be magnitudes smaller than the volume of the input stream. Please note that Poseidon's codebase is only included in the electronic submission of this project, and has not been included in this document due to its size (85 files, approximately 8700 lines).

6.1 Workflow

Figure 6.2 describes a workflow that is suitable for the requirements of Poseidon. The feed from Netcraft is exposed through files that are written to the file system every 5 minutes. Each rectangle depicts an independent process that works asynchronously, descriptions of which are as follows:

- **Feed Forager**: Retrieves the available batches of URLs from the file-system and performs any initial filtering, such as restricting the number of submissions in a given time-period for a particular hostname (an organisational requirement). The filtered population is distributed to all available classification nodes.
- **Classifier** i: Classifies the URL as phishing or benign, and sends the URL along with the class and likelihood to the Persistor. Multiple classifiers are required to enable high throughput.
- **Persistor**: Each classification result is stored in the database, and subsequently forwarded to any further eligible downstream processes.
- **Netcraft Reporter**: This process is considered "eligible" for receiving classification results that indicate that a URL belongs to the phishing class. Each received URL is subsequently reported to Netcraft.
- **Submission Processor**: Netcraft reports the results of submission as *Tab Separated Values* (TSV) files, every hour. This node processes the results, and amends the records created by the Persistor to include the verified class of the URL.
- **Head**: This node allows a user to perform administrative tasks, such as creating/training/testing classifiers and monitoring the status of other process.

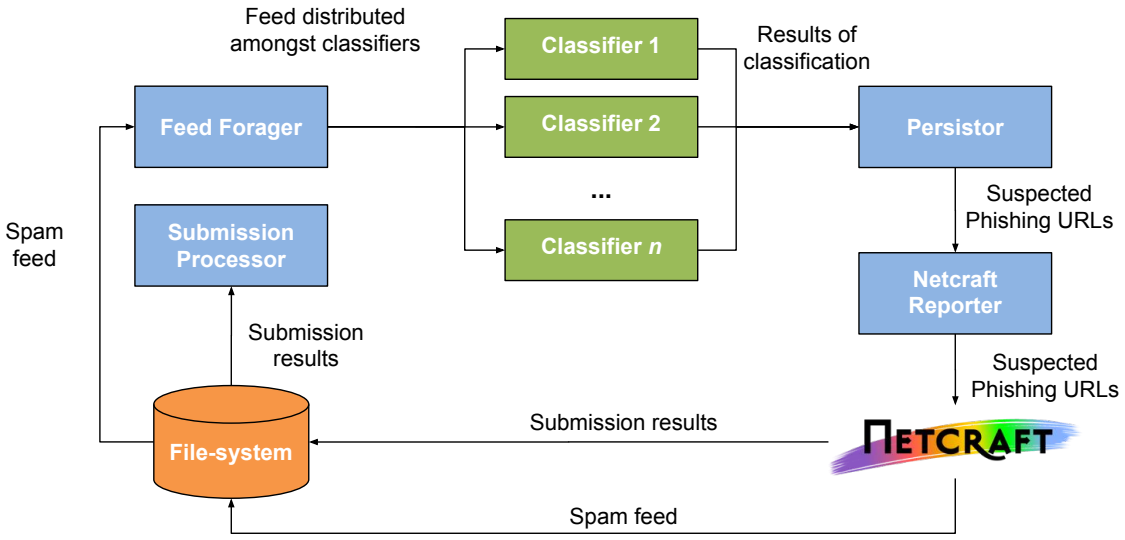

Figure 6.2: The devised workflow for processing the feed from Netcraft. The 'Head' node is not shown as it is not a part of the classification workflow.

6.2 Implementation

Python 2.7 (Python Software Foundation, 2017*a*) was used primarily to implement the solution, due to being cross-platform, its suitability to rapid prototyping (Adams et al., 2002), and a variety of extensions which enable high-performance data processing (Oliphant, 2007). These extensions include NumPy (NumPy Developers, 2017), Sci-Py (SciPy Developers, 2017) and scikit-learn (scikit-learn, 2017*a*).

6.2.1 Nodes and Inter-node Communication

Each process depicted in Figure 6.2, referred to as a 'node', was implemented as an individual process. Communication between nodes was performed using the MQTT protocol, due to being "extremely-lightweight" (mqtt.org, 2014) and event driven. MQTT uses the publish-subscribe pattern (Eugster et al., 2003) where communication is performed through a "broker" process. Each node connects to the broker and subscribes to a particular topic, denoted by a topic string, e.g. `/foo`. If a node needs to publish a message on a particular topic, it sends the message and the topic to the broker, which subsequently relays the message to all nodes subscribed to the topic. An event is raised when a message arrives, allowing the subscriber to perform any required tasks.

This pattern is advantageous as it allows publishers and subscribers to act asynchronously (Eugster et al., 2003), and is therefore suitable for the proposed architecture. A request-response layer was built on top of MQTT, to allow nodes to request data from other nodes (either synchronously or asynchronously) if required. MQTT also offers varying levels of Quality of Service that describes the delivery of data, from *'at most once'* to *'exactly once'*. Please see `src/nodes/basenode.py` for the implementation of a base node, which is inherited by all nodes in the workflow.

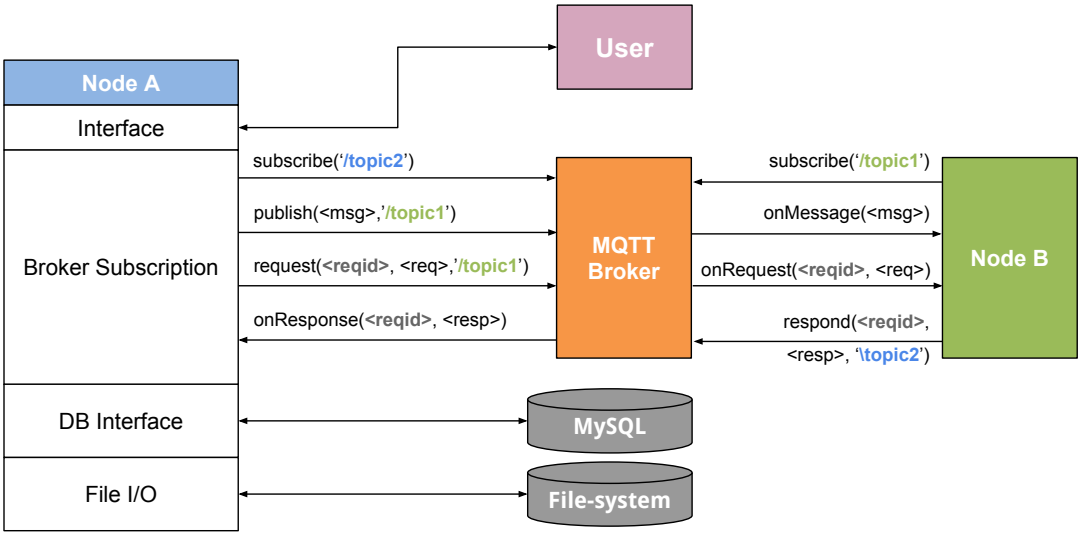

Figure 6.3: The structure and functionality of a general node.

Figure 6.3 shows the basic functionality of a node. Each node also connects to a MySQL server, and has access to the local file-system. A user can interact with each node through a command-line interface. It is important to note that each node comprises of two initial threads - a thread for the user interaction and a thread that handles communication with the broker. Extra threads are spawned as required, based on the functionality of the node.

6.2.1.1 Parallelism and Load Balancing

The problem of phishing classification lends itself well to data-parallelism. As there is no inter-dependency between URLs to be classified, the URLs can be distributed amongst n classifiers, increasing n as required, and obtaining linear speedup, while $n <= p$ where p is the number of processors. A generic method of load balancing was created (`src/nodes/nodeutils.py`) which allows invoking the same functionality across multiple nodes with different data (SIMD).

6.2.2 Database

The main purpose of the database was to store data that could be accessed/modified by multiple nodes. This includes training data, classification results, error logs and classifier configurations. As this form of data was suitable for storage in a relational database, MySQL (version 5.7.17) (Oracle, 2017) was chosen for its ease of use and deployment, as well as the ability to support concurrent access (Sun et al., 2016). However, the database back-end is not tightly-coupled with the nodes, a different back-end could be swapped into the system provided the required interface is implemented.

The following data is stored in the database:

- Training data
- Results of classification
- A list of prominent target organisations of phishing attacks (obtained from Netcraft)
- Exceptions/Errors encountered
- Classifier configurations

6.2.3 File-system

The primary use for interacting with the file-system is to access the feed of spam URLs, and analyse the results of submission to Netcraft. The file-system is also used to store the classification models, which are serialized Python objects that are loaded into memory when a classification node starts. Finally, configuration files for the nodes themselves are also stored in the file-system.

6.2.4 Detailed Implementation

The following sections describe the implementation of each nodes functions in detail.

6.2.4.1 Feed Forager

The feed forager operates using two extra threads, displayed in Figure 6.4. One of these retrieves the feed files uploaded by Netcraft, and filters the population to a list which is subsequently balanced evenly amongst the available classification nodes. The current filtering system supports hostname caching, where each next URL encountered is first checked against the hostname cache. If the hostname of the URL exists in the cache then the URL is ignored. Each entry in the cache contains a configurable expiry time, and the second thread prunes the cache periodically to remove expired entries. As explained previously, this is an organisational requirement to prevent large submissions of URLs under the same hostname within a small period of time. The implementation of this node can be found in `src/nodes/baserpnode.py`.

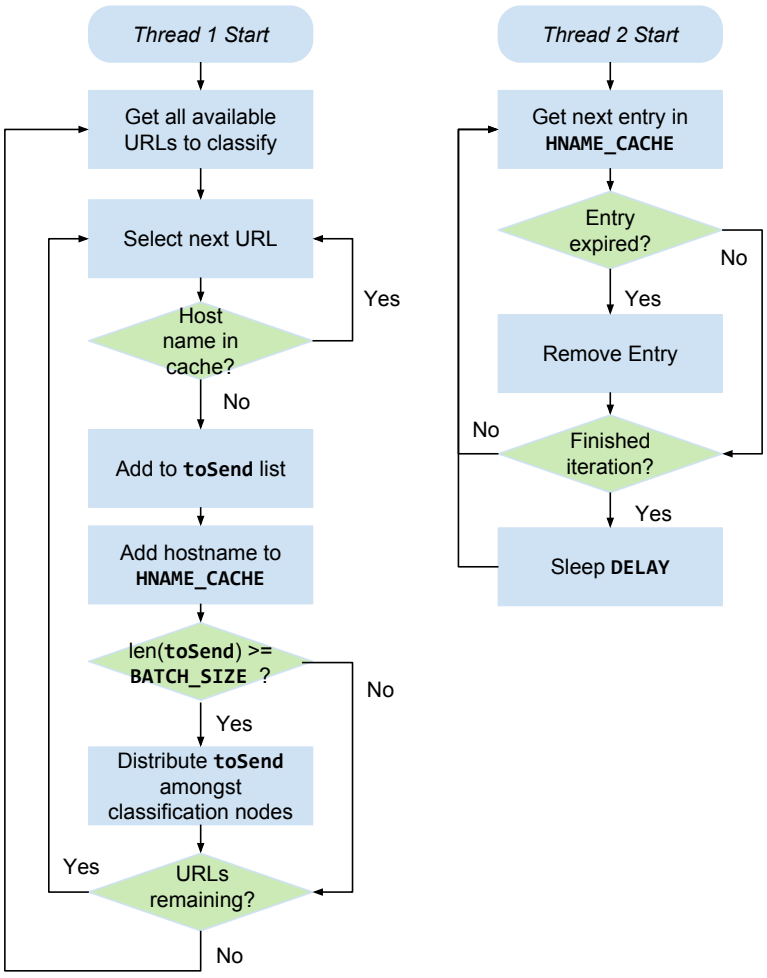

Figure 6.4: The workflow of the Feed Forager node.

6.2.5 Classifier *i*

When a classification node starts, the required *classification service* is loaded into memory, and initialized. A *classification service* consists of several parts:

- **Classification model**: The underlying model which can be used to make predictions, given a feature vector.
- **Feature Extractor**: An object used to extract features from a URL.
- **Utility Functions**: Allows management of the classifier either programmatically or through a CLI, including the ability to evaluate the classification model using various evaluation metrics, and visualize classification performance.

The node then accepts and buffers an incoming stream of URLs in a queue. Another thread retrieves a batch of URLs from the classification queue, and uses the *classification service* to obtain the most likely class (and respective probability) for each URL in the batch. The results of classification are then added to a results queue. A final thread monitors the results queue, and sends the results in batches to the Persistor node. Please see `src/nodes/lexicalclassifiernode.py` for this implementation.

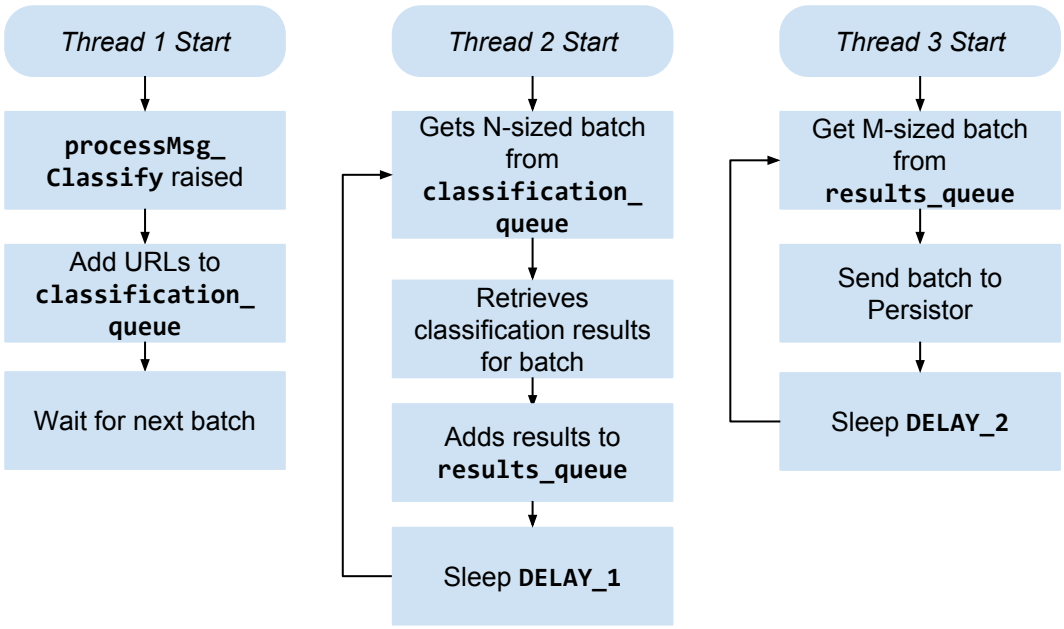

Figure 6.5: The workflow of a classification node.

6.2.5.1 Persistor

The Persistor accepts an incoming feed of classification results and stores it in a persist queue. Another thread periodically retrieves items from the queue, and stores them in the database. For each result, a check is made to see if any further downstream nodes are eligible for receiving the classification result. For example, a downstream node may only be eligible if the result indicates that a URL has been classified as phishing and the probability of this is over 60%. If a node is eligible, then the results of classification is forwarded to this node. The implementation of this can be found in `src/nodes/classificationpersistornode.py`.

6.2.5.2 Netcraft Reporter

This node operates similarly to the Persistor, however is only forwarded results of classification if the result is positive, i.e. the sample has been labelled 'phishing'. Rather than persisting the forwarded result in a database, the URL is reported to Netcraft via a HTTP request. The implementation of this can also be found in `src/nodes/classificationpersistornode.py`.

6.2.5.3 Submission Processor

Netcraft provide the results of submissions as TSV files containing the reported URL and whether it has been confirmed as phishing or not. This node parses the files, and modifies the records created by the Persistor to include the "validated" class of a URL. This allows subsequent gathering of analytics and retraining of the model with new data. Please see `src/nodes/toolbarresultsnode.py` for this implementation.

6.2.5.4 Head

The head node (`src/nodes/headnode.py`) allows the user to manage the network of nodes, as well as administrate classification services and test the performance of various features. The features are as follows:

- **Classification Service Manager**
  - Create *classification services* by specifying the classification algorithm, feature extractor and data set to use.
  - Generate confusion matrices, and various evaluation metrics (Precision, Recall, Accuracy and ROC Curves).
  - Compare classification services by evaluation metrics.
- **Feature Extractor Manager**
  - Generate features for a single sample and samples in bulk using the various feature extractors available.
  - Generate ranking of features using χ^2 (See Section 2.7.6) and generate visualizations of distributions of values for each features.
  - Perform Recursive Feature Elimination (scikit-learn, 2017*b*) to identify feature performance.
  - Profile the feature extractor using Python's cProfile module (Python Software Foundation, 2017*b*) to identify bottlenecks in this process.
- **Classification Results Manager**
  - Manually update classification results with "validated" labels.
  - Prune classification results for which validated classes have not been received in a particular time-frame.
- **Exception Log**: View the exception log, which shows when and on which node exceptions have occurred.
- **Node Status**: View the statuses (Alive, Timeout) of all nodes.

6.2.5.5 Content-Fetcher

The Content-Fetcher is a currently inactive node (and not shown on Figure 6.2), though it could be used in the future as a mechanism of tunneling external communication through a particular node. This node allows external content to be fetched both synchronously and asynchronously, and supports caching of web-requests, as well as programmatic access to specific services. For example, if a node required the search-engine rank of a particular URL, then the content-fetcher node could be requested to fetch and parse the required content, and respond with the rank.

6.2.6 Libraries

To aid with implementation, the following libraries were used:

- **paho-mqtt** (Roger Light, 2017): Python MQTT Client.
- **scikit-learn** (scikit-learn, 2017*a*): Machine learning library for Python.
- **NumPy** (NumPy Developers, 2017) and **SciPy** (SciPy Developers, 2017): Extensions for Python that support scientific computation (data analysis, linear algebra, etc.), with operations executed by native code.
- **matplotlib** (Matplotlib development team, 2017): Graphing library.
- **requests** (Kenneth Reitz, 2017) and **requests-cache** (Roman Haritonov, 2017): Performing http-requests with an optional transparent cache.
- **xgboost** (DMLC, 2017).
- **mysql-python** (Andy Dustman., 2014): Python interface to a MySQL server.
- **google-api-python-client** (Google Inc., 2017): Used to connect to Google services to use the cloud-hosted Predict API (Google, 2017) instead of scikit-learn.
- **stemming** (Matt Chaput, 2010): Optional requirement during tokenization process, though did not prove to be useful.

In addition to the above, the following software was used:

- **mosquitto** (Eclipse Turner Foundation, 2017): A MQTT Broker.
- **MySQL Server** (Oracle, 2017).
- **Apache HTTP Server** (The Apache Software Foundation, 2017) and **phpMyAdmin** (phpMyAdmin contributors, 2017): Allows management of a MySQL database through a web-interface.
- **tmux** (tmux, 2017): Allows multiple persistent sessions in a single terminal session.

6.3 Evaluation

Poseidon was deployed as per Figure 6.1, using a Random Forest model trained on the entire SBSP data set, with 150 trees. Two classification nodes were found to be sufficient for processing each batch of URLs from Netcraft without the build-up of a backlog. The machine chosen was a cloud server from DigitalOcean (DigitalOcean Inc., 2017), with the following specifications:

- CentOS 7 (The CentOS Project, 2017)
- 8-Core Processor (Intel Xeon E5-2650L v3 @ 1.80GHz)
- 16GB RAM

While the system did not need the full 8-cores to function at the required performance, this ensured that each node could use a full core, with surplus resources available for related services such as the MySQL server, and MQTT Broker.

The following data was collected hourly for a period of 14 days from 2017-03-17 to 2017-03-31:

- **num_processed**: The number of URLs from Netcraft that were classified by Poseidon. This population excludes URLs filtered out by the feed forager, and solely represents those that were sent to the classification nodes.
- **num_reported**: The number of URLs classified as phishing and reported to Netcraft.
- **num_tp**: The number of reported URLs that were confirmed as being truly phishing.
- **num_fp**: The number of reported URLs that were confirmed as being benign.

As mentioned by Canali et al. (2011), obtaining other metrics such as false negatives and true negatives are difficult, as the input data is unlabelled. Identifying the true classes manually of a data set of this size was not feasible.

6.3.1 Results

The following results were collected during the experiment:

Time interval	num_processed	num_reported	num_tp	num_fp
Overall	31,639,143	186,517	2803	183714
Per day	2,259,939	13,323	200	13,122
Per hour	94,164	555	8	547
Per minute	1569	9.25	0.14	9.11
Per second	26.16	0.154	0.002	0.152

Table 6.1: The total volume processed by Poseidon, and statistics of submissions, averaged over various time intervals.

System	Samples per day	Load Reduction %	False Positive % [1]	Classification Time[2]
Poseidon	2,259,939	99	0.58	0.004s/URL[3]
Prophiler (Canali et al., 2011)	320,000	85.7	13.7	3.58s/URL
Gyawali et al. (2011)	479,470	90%	N/A	N/A

Table 6.2: Data collected for Poseidon over the 14-day period, as well as data available for similar systems.

[1] Percentage of submissions that are benign.
[2] This includes both feature-extraction time and time taken to evaluate using the classification model.
[3] Single-core performance.

6.3.2 Performance Analysis

The primary purpose of Poseidon is to reduce the number of URLs that require classification by a more expensive classification process. Table 6.2 shows data for other similar systems - Prophiler, a full-feature classifier built by Canali et al. (2011), and a lexical classifier built by Gyawali et al. (2011). It is evident that Poseidon is able to achieve the greatest load reduction (99% compared to 85.7% and 90% respectively).

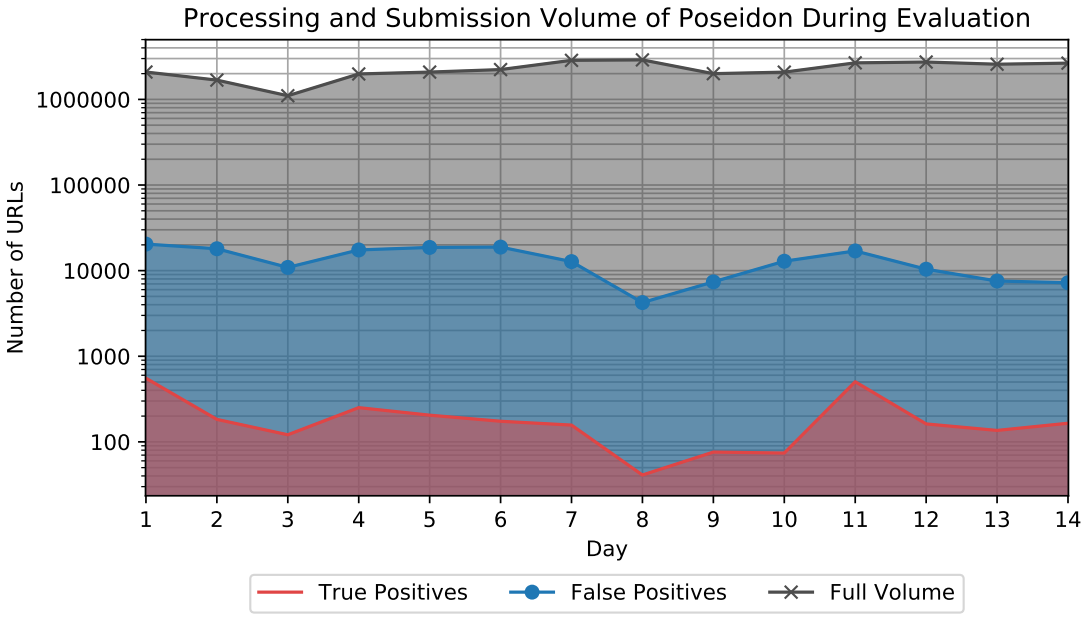

Figure 6.6: The full volume of URLs processed by Poseidon during the evaluation period are shown in gray, with the number of false positives (blue) and true positives (red) overlaid. It is important to note that the y-axis scale is log_{10}. Averaged values can be found in Table 6.1.

Posiedon also maintains a significantly lower percentage of false positives - 0.58% compared to Prophiler's 13.7%. This means that Poseidon only submitted 0.58% of the total population incorrectly as phishing, whereas Prophiler submits 13.7% (24-fold times higher).

On the other hand, when observing the true positives in Prophiler's submissions, 4%[4] of submissions were in fact phishing, while only 1.6% of Poseidon's submissions were phishing. This 2.5-fold decrease in True Positives does however come at the advantage of a 24-fold decrease in false positives.

The environment in which Gyawali et al. (2011) evaluated their classifier is interesting as it is also an imbalanced feed of spam URLs. Gyawali et al. (2011) report two metrics when evaluating their classifier, Cumulative Error Rate (CER) and Balanced Success Rate (BSR). The CER is calculated as $\frac{FN}{FN+TP}$, and indicates the percentage of the total phishing population that was misclassified as benign. While the large-scale evaluation of Poseidon does not facilitate the calculation of this metric due to processing unlabelled data, the results from Section 5.4 indicate a lower CER of 6% [5] compared to 7.5% as reported by Gyawali et al. (2011). The data for this was generated by creating a training set containing a 1:2 ratio of phishing to benign URLs, and a testing set containing a 1:652 ratio - the same conditions as the feed processed by Gyawali et al. (2011).

[4] 14.3% submitted in total, with 13.7% false positives, so 0.6% true positives. $\frac{0.6}{14.3} = 4\%$ (Canali et al., 2011).
[5] TP = 94, FN = 6

The BSR obtained during this experiment was 93.4%, which is similar to the BSR reported by Gyawali et al. (2011) (The explicit values for BSR per day are not reported by Gyawali et al. (2011), though their visualization indicates a BSR of mid-90s). However, it is important to note that this performance is maintained with a 10% greater load reduction.

6.3.3 Cost of Deployment

The machine on which Poseidon was deployed uses a 8-core processor[6], each core running at 1.8GHz. A single classification node is able to classify 250 URLs/s on a single core. If only a single-core is dedicated to classification, this allows the processing of a feed consisting of a maximum volume of 21,600,00 URLs per day, with the entire classification workflow (including required software such as the MQTT broker and MySQL server) taking up no more than 3 cores during peak times.

With this data, it is possible to estimate the monthly cost of running the classification system. DigitalOcean (DigitalOcean Inc., 2017) and Amazon EC2 (Amazon Web Services, Inc., 2017) are two providers of large-scale cloud-computing, offering hosts that have up to 32 and 64 cores respectively.

Provider	**Cores**	**Memory (GB)**	**Hourly Cost ($)**	**Daily Cost ($)**	**Monthly Cost ($)** [7]	**Max Throughput (URLs/day).**
Digital Ocean	4	8	0.119	2.856	85.68	43,200,000
Amazon EC2 (t2.xlarge)	4	16	0.188	4.512	135.36	43,200,000

Table 6.3: Running costs for a 4-core cloud-based virtual machine from DigitalOcean (DigitalOcean Inc., 2017) and Amazon EC2 (Amazon Web Services, Inc., 2017) [8].

Table 6.3 indicates that Poseidon costs $85.68 per month to process a feed of up to 43,200,000 URLs per day using the solution from DigitalOcean. Pricing was retrieved for 4-core systems as configurations for 3-core systems were not available, hence allowing an extra classification node to run on the fourth core. When applied to the feed from Netcraft (which has a smaller average volume of 2,259,939 URLs per day), Poseidon was able to identify 200.2 true positives per day, which equates to $0.0143/phishing URL. If a feed of the maximum volume was used, extrapolating the current classification performance indicates that true positives would cost $0.0007 each.

When measuring the general classification cost, Poseidon can perform 50,420 classifications per dollar using the same 4-core system. To contrast with this, Google's Machine Learning Engine (a cloud-based machine-learning service) only offers 10,000 classifications per dollar, along with a flat hourly fee of $0.40. In order to estimate the cost for producing a validated phishing feed that uses Poseidon as an initial filter, one would need to also factor in the cost of the input stream of URLs, as well as the cost of running the secondary classifier.

6.3.4 Evaluation Summary

The evaluation indicates that Poseidon is able to outperform the state-of-the-art when considering previously-reported evaluation metrics, while being able to simultaneously achieve the greatest load reduction, pruning the input feed to 1% of its size. These evaluation metrics include Balanced Success Rate (which equally weights Recall and the True Negative Rate), Cumulative Error Rate, False Positives and False Negatives.

When applied to the Netcraft feed of spam URLs, Poseidon is able to mine 200 phishing URLs/day on average, costing $0.0143 per true positive when deployed on a 4-core cloud server.

[6] Intel Xeon E5-2650L v3 @ 1.80GHz
[7] Based on a 30-day month.
[8] Prices taken from the US East Region (as shown by default) on 03-04-2017

7 Conclusion

This study investigated the application of machine learning for purposes of lexical phishing classification, through three main stages - identifying the optimal feature vector, identifying the optimal classification algorithm and subsequently building and evaluating a large-scale classification system that could be used in the industry.

7.1 Traditional and Novel Features

Existing literature in this domain mostly focuses on the classification of phishing websites by using features extracted from more than just the URL, i.e. the web-page itself (Ma, Saul, Savage and Voelker, 2009*a*, Whittaker et al., 2010, Xiang et al., 2011) as well as information requested from third-party services, such as search-engine ranking. The limitations that arise as a result of using these methods (e.g. cloaking, latency, high-memory usage) are prevalent (Garera et al., 2007, Le et al., 2011), however these can be avoided by restricting the feature collection process to features from the URL only.

By leveraging the research available for lexical classification, as well as identifying novel features, an optimal feature vector was identified, using the χ^2 test to measure feature performance. This feature vector only retrieves data from the URL itself, and therefore has no external dependencies, overcoming the problems identified in Section 2.7. The results indicated that 85% of novel features are able to distinguish between benign and phishing URLs with statistical significance ($p <= 0.001$), with 50% of novel features ranking in the top 50% of features in the optimal feature vector. This process also identified features that were not effective, despite having shown effectiveness in past studies, which could be attributed to evolution of methods used in phishing attacks in an attempt to evade detection.

7.2 Optimal Classification Algorithm

The process of finding the optimal classification algorithm identified that tree-based algorithms perform well for this domain, in particular Random Forest and XGBoost (DMLC, 2017). Greatest performance was achieved with the Random Forest algorithm with 150 trees and using $\sqrt{n}$ features[1] when splitting at nodes during tree-generation (91.5% Accuracy, 91.6% F1-score and 0.97 ROC AUC).

It is important to note that a noisy data set was used for training and testing models, where the benign and phishing URLs are structurally similar and contain various URL elements, and are also sourced from a location where phishing attacks are prominent (email) (Almomani et al., 2013). This contrasts with a large amount of existing literature, where benign URLs are sourced from directories such as Alexa (Alexa, 2017) or DMOZ (DMOZ, 2016), and do not often contain URL elements such as a path, filename and arguments. Attackers use various online communication channels to distribute phishing URLs, and therefore being able to distinguish between benign and phishing URLs within these channels is advantageous and is more representative of real-life application.

It was shown that the classification model built in this study is able to outperform the state-of-the-art when performing lexical phishing classification, in terms of the various evaluation metrics identified in Section 2.6.1. When compared with full-featured classifiers, this model is often able to achieve greater performance too, indicating its suitability as a potential replacement for particular uses. These findings are consistent with the findings of Le et al. (2011), where only a small degradation in performance was observed when using lexical features only.

[1] Where n is the total number of features.

The effects of evolution of phishing attacks were also investigated over a period of 50 days, where only an increase of 1.53% in cumulative error rate was observed when using a model that was trained once at the start of the 50 day period over a model that was trained daily.

7.3 Real-life Application

Very few studies have attempted to evaluate phishing classification methods in a real-life settings. In this study, a large-scale classification system (Poseidon) was created, and applied to a high-volume feed of URLs[2] sourced from spam email. The purpose of the system was to act as a method of reliable load-reduction, to reduce the number of URLs that are subsequently sent to a more expensive classification process. Poseidon was found to also outperform existing systems of a similar nature (Canali et al., 2011, Gyawali et al., 2011), when observing metrics such as load-reduction, false-positive rate, and recall.

An analysis of Poseidon's computational requirements indicated that a maximum throughput of 43 million URLs/day can be achieved on a 4-core system. When applied to the URL feed that was used during evaluation (which is 5% of the maximum throughput), each correctly identified phishing URL came at the cost of $0.0143[3].

The system also offers various tools that can be used to create and evaluate classification models, evaluate various forms of feature extraction and manage the classification network. The workflow is easily scalable, and nodes can be added/removed with ease. Existing libraries were leveraged where possible to aid with both the machine-learning aspect of the system, and the distributed nature of the system.

Poseidon has been deployed at Netcraft since February 2017, and is currently identifying 7500 unique phishing attacks per month.

7.4 Limitations & Future Work

Several limitations can be identified in this study, which offer opportunities for extension of this research in the future.

7.4.1 Classification Algorithms & Technology

The classification algorithms that were evaluated were mainly limited to those that were found to perform well in literature. While a recently developed algorithm, XGBoost (DMLC, 2017), was introduced and evaluated in this domain, a more extensive effort may have found more well-suited algorithms that may outperform those that are traditionally used.

Furthermore, the implementations of the algorithms used do not leverage a GPU if present, either for model-generation or making predictions. It has been shown that Neural Networks are faster and more efficient when using a GPU for computation (Luo et al., 2005). The performance gains from using a GPU may make more complex models be viable, as making predictions will take less time and so the computational cost of a using more complex model will be lower.

Some studies have used online-learning algorithms (Blum et al., 2010, Le et al., 2011, Whittaker et al., 2010) to tackle phishing classification, unlike the batch-based learning algorithms explored in this study. Online-learning is performed by updating the classification sequentially with each next sample (Ma, Saul, Savage and Voelker, 2009*b*), allowing the model to change the way various features influence the

[2] 2,259,939 URLs/day.

[3] Using DigitalOcean's 4-core cloud server, and only accounting for costs of running Poseidon.

prediction. A comparison of online-learning and batch-learning algorithms performed by Ma, Saul, Savage and Voelker (2009*b*) identified that online-learning algorithms are able to keep up to date with "changing trends in URL features", thus adapting to the evolution of phishing attacks. Ma, Saul, Savage and Voelker (2009*b*) also identified that the online-learning algorithms outperformed batch-learning algorithms.

7.4.2 Retraining Poseidon

While online-learning algorithms lend themselves to continuous retraining by design (Ma, Saul, Savage and Voelker, 2009*b*), batch-based algorithms can also be retrained at specific intervals to keep up to date with changing URL trends, as shown in Section 5.5. Poseidon offers the ability to specify "dynamic data sets", which select the most recently classified URLs that have been labelled with their true class. An automated retraining system could be created that would leverage this dynamic data set, and retrain the underlying model at user-defined intervals. Research would need to be performed to find the optimal interval - this study identified that a *daily* retraining scheme leads to a reduction in cumulative error rate of 1.5%.

Furthermore, ways to sample the new training data would need to also be investigated, so that particular trends do not overfit the model if they are significantly more prominent than other types of attacks within a short period of time. Whittaker et al. (2010) address this by limiting the maximum number of URLs from a particular domain to 150 per week, in order to reduce "concentrated phishing attacks from dominating" the training data.

7.4.3 Improved Large Scale Evaluation

A limitation of the large-scale evaluation arises due to the issue of having unlabelled data. Without an estimate of the ratio of phishing and benign URLs, it is impossible to identify the number of phishing URLs that were missed by Poseidon. While the other metrics were indeed useful, having a labelled data set would facilitate a more interesting evaluation which could also offer opportunities to understand the nature of URLs that Poseidon fails to detect. Canali et al. (2011) were able to address this problem by verifying the true classes of 1% of URLs that their classifier deemed to be benign, thus being able to obtain an estimate of false-negatives in the total population. A similar approach could be taken to do the same for Poseidon.

7.4.4 Feature Analysis

It could be interesting to analyse if joint influences exist from various subsets of features. While some features may not significantly influence the outcome of classification, some features may have a significant impact on the predicted class when they take on particular combinations of values. *Quantative Input Influence* (QII) is a set of measures developed by Datta et al. (2016), which can help identify joint influences from input features. QII can also provide another perspective on the individual influence of features, which can be useful when analysing the effectiveness of particular features. Another useful application of QII would be to identify deficiencies in the underlying training data. As QII can provide information to explain decisions regarding particular outcomes, one would be able to identify if a model is overfitted to particular trends in the training data that would not be observed in the real life scenario.

7.4.5 URL Shorteners

URL shorteners can be used by attackers to evade detection by lexical classification, since none of the true URL's lexical features are exposed. This is a crucial weakness of Poseidon, as it is unable to meaningfully classify URLs that use URL-shortening. In order to combat this, the system could be extended to use a URL-resolver to retrieve the true URL if URL-shortening is detected. While this does indeed involve fetching external content, and is subsequently susceptible to the issues identified in

Section 2.7, performing this would be better than nothing at all. Poseidon's network has a content-fetcher node (currently unused) which can act as a middle-man between all external requests. By strategically placing the node on networks that are not known to fetch phishing content for the purposes of detection, the effects of "cloaking" could be mitigated, allowing more reliable retrieval of external content.

7.4.6 Exploring Methods of Evasion

Detecting phishing is a cat-and-mouse game, with attackers constantly trying to evade detection. As pointed out by Darling et al. (2015), attackers have access to the vast amounts of literature available on phishing detection, and are therefore able to formulate their attacks in ways that can overcome some detection techniques. While attackers are indeed able to modify the structure of a URL to evade some of the features identified in this study, other features play a vital role in the process of deceiving a victim. Including a the target organisation's brand name in the URL, for example, can convince an individual that the link is legitimate - though this is also a very effective indicator that the link points to a phishing page if the domain at which it is hosted does not belong to the target organisation. It is therefore evident that a trade-off exists when deploying a phishing attack - a URL could be transformed to be as less phishing-like as possible, though this may come at a cost of its ability to mislead individuals, and may incur a significant economic cost too (Pan and Ding, 2006). An investigation could be undertaken to find features which are most resistant to evasion, as well as features which are most easily evaded with the least cost to the attacker.

Bibliography

Abdelhamid, N., Ayesh, A. and Thabtah, F. (2014), 'Phishing detection based associative classification data mining', *Expert Systems with Applications* **41**(13), 5948–5959.

Abu-Nimeh, S., Nappa, D., Wang, X. and Nair, S. (2007), A comparison of machine learning techniques for phishing detection, *in* 'Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit', ACM, pp. 60–69.

Abunadi, A., Akanbi, O. and Zainal, A. (2013), Feature extraction process: A phishing detection approach, *in* '2013 13th International Conference on Intellient Systems Design and Applications', IEEE, pp. 331–335.

Aburrous, M., Hossain, M. A., Dahal, K. and Thabtah, F. (2010), Predicting phishing websites using classification mining techniques with experimental case studies, *in* 'Information Technology: New Generations (ITNG), 2010 Seventh International Conference on', IEEE, pp. 176–181.

Adams, P. D., Grosse-Kunstleve, R. W., Hung, L.-W., Ioerger, T. R., McCoy, A. J., Moriarty, N. W., Read, R. J., Sacchettini, J. C., Sauter, N. K. and Terwilliger, T. C. (2002), 'Phenix: building new software for automated crystallographic structure determination', *Acta Crystallographica Section D: Biological Crystallography* **58**(11), 1948–1954.

Ake-Johnson, O. G., Zavarsky, P., Ruhl, R., Lindskog, D. and Umana, C. (2010), A game theoretical multi-layered defense approach against phishing attacks, *in* 'Computer Security Conference 2010, At Myrtle Beach, SC'.

Alexa (2017), 'Alexa [online]', Available from: `http://www.alexa.com/`. [Accessed: 2017-04-01].

Alkhozae, M. G. and Batarfi, O. A. (2011), 'Phishing websites detection based on phishing characteristics in the webpage source code', *International Journal of Information and Communication Technology Research* **1**(6).

Almomani, A., Gupta, B., Atawneh, S., Meulenberg, A. and Almomani, E. (2013), 'A survey of phishing email filtering techniques', *IEEE communications surveys & tutorials* **15**(4), 2070–2090.

Amazon Web Services, Inc. (2017), 'Elastic compute cloud (ec2) âĂŞ cloud server & hosting âĂŞ aws [online]', Available from: `https://aws.amazon.com/ec2/`. [Accessed: 2017-01-08].

Andy Dustman. (2014), 'Mysql-python 1.2.5 : Python package index [online]', Available from: `https://pypi.python.org/pypi/MySQL-python/1.2.5`. [Accessed: 2017-01-08].

Anti-Phishing Working Group (2016), 'Phishing activity trends report: 2nd quarter 2016 [online]', Available from: `https://docs.apwg.org/reports/apwg_trends_report_q2_2016.pdf`. [Accessed: 2016-11-03].

Aydin, M. and Baykal, N. (2015), Feature extraction and classification phishing websites based on url, *in* 'Communications and Network Security (CNS), 2015 IEEE Conference on', IEEE, pp. 769–770.

Balusamy, B., Velu, M., Nandagopal, S. and Mano, S. J. (2016), 'Achieving security to overcome attacks and vulnerabilities in mobile banking security', *Online Banking Security Measures and Data Protection* p. 237.

Batista, G. E., Prati, R. C. and Monard, M. C. (2004), 'A study of the behavior of several methods for balancing machine learning training data', *ACM Sigkdd Explorations Newsletter* **6**(1), 20–29.

Bergholz, A., Chang, J. H., Paass, G., Reichartz, F. and Strobel, S. (2008), Improved phishing detection using model-based features., *in* 'CEAS'.

Berners-Lee, T., Masinter, L. and McCahill, M. (1994), Uniform resource locators (url), Technical report.

Blum, A., Wardman, B., Solorio, T. and Warner, G. (2010), Lexical feature based phishing url detection using online learning, *in* 'Proceedings of the 3rd ACM workshop on Artificial intelligence and security', ACM, pp. 54–60.

Breiman, L. (2001), 'Random forests', *Machine learning* **45**(1), 5–32.

Breiman, L. (2003), 'Manual–setting up, using, and understanding random forests v4. 0. 2003 http://oz. berkeley. edu/users/breiman', *Using_random_forests_v4. 0. pdf* .

Brin, S. and Page, L. (2016), 'The anatomy of a large-scale hypertextual web search engine [online]', Available from: `http://infolab.stanford.edu/~backrub/google.html`. [Accessed: 2016-11-20].

Canali, D., Cova, M., Vigna, G. and Kruegel, C. (2011), Prophiler: a fast filter for the large-scale detection of malicious web pages, *in* 'Proceedings of the 20th international conference on World wide web', ACM, pp. 197–206.

Chen, T. (2014), 'Introduction to boosted trees', *University of Washing Computer Science. University of Washington* **22**.

Chen, T. and Guestrin, C. (2016), Xgboost: A scalable tree boosting system, *in* 'Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', ACM, pp. 785–794.

Chu, W., Zhu, B. B., Xue, F., Guan, X. and Cai, Z. (2013), Protect sensitive sites from phishing attacks using features extractable from inaccessible phishing urls, *in* '2013 IEEE International Conference on Communications (ICC)', IEEE, pp. 1990–1994.

Cova, M., Kruegel, C. and Vigna, G. (2008), 'There is no free phish: An analysis of" free" and live phishing kits.', *WOOT* **8**, 1–8.

Cover, T. M. (1991), 'Elements of information theory thomas m. cover, joy a. thomas copyright© 1991 john wiley & sons, inc. print isbn 0-471-06259-6 online isbn 0-471-20061-1'.

Darling, M., Heileman, G., Gressel, G., Ashok, A. and Poornachandran, P. (2015), A lexical approach for classifying malicious urls, *in* 'High Performance Computing & Simulation (HPCS), 2015 International Conference on', IEEE, pp. 195–202.

Datta, A., Sen, S. and Zick, Y. (2016), Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems, *in* 'Security and Privacy (SP), 2016 IEEE Symposium on', IEEE, pp. 598–617.

Davis, J. and Goadrich, M. (2006), The relationship between precision-recall and roc curves, *in* 'Proceedings of the 23rd international conference on Machine learning', ACM, pp. 233–240.

DigitalOcean Inc. (2017), 'Compute on digitalocean — droplets: fast, resizable cloud servers [online]', Available from: `https://www.digitalocean.com/products/compute/`. [Accessed: 2017-01-08].

DMLC (2017), 'Xgboost: Scalable, portable and distributed gradient boosting (gbdt, gbrt or gbm) library, for python, r, java, scala, c++ and more. [online]', Available from: `https://github.com/dmlc/xgboost`. [Accessed: 2017-03-22].

DMOZ (2016), 'Dmoz [online]', Available from: `https://www.dmoz.org/`. [Accessed: 2016-11-14].

Eclipse Turner Foundation (2017), 'An open source mqtt v3.1 broker [online]', Available from: `https://mosquitto.org/`. [Accessed: 2017-01-08].

Eugster, P. T., Felber, P. A., Guerraoui, R. and Kermarrec, A.-M. (2003), 'The many faces of publish/subscribe', *ACM computing surveys (CSUR)* **35**(2), 114–131.

Fawcett, T. (2006), 'An introduction to roc analysis', *Pattern recognition letters* **27**(8), 861–874.

Feroz, M. N. and Mengel, S. (2015), Phishing url detection using url ranking, *in* 'Big Data (BigData Congress), 2015 IEEE International Congress on', IEEE, pp. 635–638.

Fette, I., Sadeh, N. and Tomasic, A. (2006), Learning to detect phishing emails, Technical report, DTIC Document.

Friedman, J., Hastie, T. and Tibshirani, R. (2001), *The elements of statistical learning*, Vol. 1, Springer series in statistics Springer, Berlin.

Garera, S., Provos, N., Chew, M. and Rubin, A. D. (2007), A framework for detection and measurement of phishing attacks, *in* 'Proceedings of the 2007 ACM workshop on Recurring malcode', ACM, pp. 1–8.

Google (2016), 'Google [online]', Available from: `https://www.google.co.uk/`. [Accessed: 2016-11-14].

Google (2017), 'Google cloud prediction api documentation [online]', Available from: `https://cloud.google.com/prediction/docs/`. [Accessed: 2017-01-08].

Google Inc. (2017), 'google-api-python-client 1.6.2 : Python package index [online]', Available from: `https://pypi.python.org/pypi/google-api-python-client/`. [Accessed: 2017-01-08].

Görling, S. (2006), The myth of user education, *in* 'Virus Bulletin Conference', Vol. 11, p. 13.

Gyawali, B., Solorio, T., Montes-y Gómez, M., Wardman, B. and Warner, G. (2011), Evaluating a semisupervised approach to phishing url identification in a realistic scenario, *in* 'Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference', ACM, pp. 176–183.

Hsu, C.-W., Chang, C.-C., Lin, C.-J. et al. (2003), 'A practical guide to support vector classification'.

Huang, H., Qian, L. and Wang, Y. (2012), 'A svm-based technique to detect phishing urls', *Information Technology Journal* **11**(7), 921.

ICANN (2016), 'About whois [online]', Available from: `https://whois.icann.org/en/about-whois`. [Accessed: 2016-11-20].

James, J., Sandhya, L. and Thomas, C. (2013), Detection of phishing urls using machine learning techniques, *in* 'Control Communication and Computing (ICCC), 2013 International Conference on', IEEE, pp. 304–309.

Kaggle (2017), 'Kaggle: Your home for data science [online]', Available from: `https://www.kaggle.com/`. [Accessed: 2017-03-22].

Karthika, L. and Perumal, V. (2016), 'A study on phishing attack in internet banking', *International journal of Interdisciplinary in Engineering Science and Technology* **1**(2), 29–32.

Kenneth Reitz (2017), 'Requests: Http for humans [online]', Available from: `http://docs.python-requests.org/en/master/`. [Accessed: 2017-01-08].

Khonji, M., Iraqi, Y. and Jones, A. (2013), 'Phishing detection: a literature survey', *IEEE Communications Surveys & Tutorials* **15**(4), 2091–2121.

Khonji, M., Jones, A. and Iraqi, Y. (2011), A novel phishing classification based on url features, *in* '2011 IEEE GCC Conference and Exhibition (GCC)'.

Knickerbocker, P., Yu, D. and Li, J. (2009), Humboldt: A distributed phishing disruption system, *in* '2009 eCrime Researchers Summit', IEEE, pp. 1–12.

Kumaraguru, P., Rhee, Y., Acquisti, A., Cranor, L. F., Hong, J. and Nunge, E. (2007), Protecting people from phishing: the design and evaluation of an embedded training email system, *in* 'Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, pp. 905–914.

Kwak, N. and Choi, C.-H. (2002), 'Input feature selection by mutual information based on parzen window', *IEEE transactions on pattern analysis and machine intelligence* **24**(12), 1667–1671.

Lakshmi, V. S. and Vijaya, M. (2012), 'Efficient prediction of phishing websites using supervised learning algorithms', *Procedia Engineering* **30**, 798–805.

Le, A., Markopoulou, A. and Faloutsos, M. (2011), Phishdef: Url names say it all, *in* 'INFOCOM, 2011 Proceedings IEEE', IEEE, pp. 191–195.

Lee, J.-L., Kim, D.-H. and Chang-Hoon, L. (2015), 'Heuristic-based approach for phishing site detection using url features'.

Liaw, A. and Wiener, M. (2002), 'Classification and regression by randomforest', *R news* **2**(3), 18–22.

Likarish, P., Jung, E., Dunbar, D., Hansen, T. E. and Hourcade, J. P. (2008), B-apt: Bayesian anti-phishing toolbar, *in* '2008 IEEE International Conference on Communications', IEEE, pp. 1745–1749.

Ludl, C., McAllister, S., Kirda, E. and Kruegel, C. (2007), On the effectiveness of techniques to detect phishing sites, *in* 'International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment', Springer, pp. 20–39.

Luo, Z., Liu, H. and Wu, X. (2005), Artificial neural network computation on graphic process unit, *in* 'Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on', Vol. 1, IEEE, pp. 622–626.

Ma, J., Saul, L. K., Savage, S. and Voelker, G. M. (2009*a*), Beyond blacklists: learning to detect malicious web sites from suspicious urls, *in* 'Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, pp. 1245–1254.

Ma, J., Saul, L. K., Savage, S. and Voelker, G. M. (2009*b*), Identifying suspicious urls: an application of large-scale online learning, *in* 'Proceedings of the 26th annual international conference on machine learning', ACM, pp. 681–688.

Ma, L., Ofoghi, B., Watters, P. and Brown, S. (2009), Detecting phishing emails using hybrid features, *in* 'Ubiquitous, Autonomic and Trusted Computing, 2009. UIC-ATC'09. Symposia and Workshops on', IEEE, pp. 493–497.

Marchal, S., François, J., Engel, T. et al. (2012), Proactive discovery of phishing related domain names, *in* 'International Workshop on Recent Advances in Intrusion Detection', Springer, pp. 190–209.

Matplotlib development team (2017), 'Matplotlib: Python plotting - matplotlib 2.0.0 documentation [online]', Available from: `http://matplotlib.org/`. [Accessed: 2017-01-08].

Matt Chaput (2010), 'stemming 1.0 : Python package index [online]', Available from: `https://pypi.python.org/pypi/stemming/1.0`. [Accessed: 2017-01-08].

Matthews, B. W. (1975), 'Comparison of the predicted and observed secondary structure of t4 phage lysozyme', *Biochimica et Biophysica Acta (BBA)-Protein Structure* **405**(2), 442–451.

Miyamoto, D., Hazeyama, H. and Kadobayashi, Y. (2008), An evaluation of machine learning-based methods for detection of phishing sites, *in* 'International Conference on Neural Information Processing', Springer, pp. 539–546.

Mohammad, R. M., Thabtah, F. and McCluskey, L. (2014), 'Intelligent rule-based phishing websites classification', *IET Information Security* **8**(3), 153–160.

Mohammad, R. M., Thabtah, F. and McCluskey, L. (2015), 'Phishing websites features'.

Mohri, M., Rostamizadeh, A. and Talwalkar, A. (2012), *Foundations of machine learning*, MIT press.

Moore, T. and Clayton, R. (2007), Examining the impact of website take-down on phishing, *in* 'Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit', ACM, pp. 1–13.

mqtt.org (2014), 'Mqtt [online]', Available from: `http://mqtt.org/`. [Accessed: 2016-10-27].

net4sec (2016), 'Clean-mx [online]', Available from: `http://support.clean-mx.com/clean-mx/phishing.php/`. [Accessed: 2016-11-14].

Netcraft (2017*a*), 'Netcraft extension [online]', Available from: `http://toolbar.netcraft.com/`. [Accessed: 2017-04-01].

Netcraft (2017*b*), 'Netcraft [online]', Available from: `https://www.netcraft.com/`. [Accessed: 2017-04-01].

Ng, A. Y. and Jordan, M. I. (2002), 'On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes', *Advances in neural information processing systems* **2**, 841–848.

Nguyen, V. (2013), Attribution of spear phishing attacks: A literature survey, Technical report, DTIC Document.

Novaković, J., Strbac, P. and Bulatović, D. (2011), 'Toward optimal feature selection using ranking methods and classification algorithms', *Yugoslav Journal of Operations Research* **21**(1), 119–135.

NumPy Developers (2017), 'Numpy – numpy [online]', Available from: `http://www.numpy.org/`. [Accessed: 2017-03-22].

Nunan, A. E., Souto, E., dos Santos, E. M. and Feitosa, E. (2012), Automatic classification of cross-site scripting in web pages using document-based and url-based features, *in* 'Computers and Communications (ISCC), 2012 IEEE Symposium on', IEEE, pp. 000702–000707.

Oliphant, T. E. (2007), 'Python for scientific computing', *Computing in Science & Engineering* **9**(3).

OpenPhish (2016), 'Openphish [online]', Available from: `https://openphish.com/`. [Accessed: 2016-11-14].

Oracle (2017), 'Mysql [online]', Available from: `https://www.mysql.com/`. [Accessed: 2016-11-10].

Pan, Y. and Ding, X. (2006), Anomaly based web phishing page detection., *in* 'Acsac', Vol. 6, pp. 381–392.

PayPal (2017), 'Paypal uk: Pay, send money and accept online payments [online]', Available from: `https://www.paypal.com/gb/home`. [Accessed: 2017-05-02].

Pennsylvania State University (2017), '11.2 - chi-square test of independence [online]', Available from: `https://onlinecourses.science.psu.edu/stat200/node/73`. [Accessed: 2017-02-04].

PhishTank (2017), 'Phishtank [online]', Available from: `https://www.phishtank.com/`. [Accessed: 2017-04-01].

phpMyAdmin contributors (2017), 'phpmyadmin [online]', Available from: `https://www.phpmyadmin.net/`. [Accessed: 2017-01-08].

Python Software Foundation (2017*a*), 'Python 2.7.0 release [online]', Available from: `https://www.python.org/download/releases/2.7/`. [Accessed: 2017-03-22].

Python Software Foundation (2017*b*), 'The python profilers - python 2.7.13 documentation [online]', Available from: `https://docs.python.org/2/library/profile.html#module-cProfile`. [Accessed: 2017-01-08].

Ramanathan, V. and Wechsler, H. (2012), 'phishgillnetâĂŤphishing detection methodology using probabilistic latent semantic analysis, adaboost, and co-training', *EURASIP Journal on Information Security* **2012**(1), 1.

Ramzan, Z. (2010), 'Phishing attacks and countermeasures', *Handbook of information and communication security* pp. 433–448.

Robie, J. (1998), 'What is the document object model?', Available from: `https://www.w3.org/TR/WD-DOM/introduction.html`. [Accessed: 2016-11-21].

Robila, S. A. and Ragucci, J. W. (2006), Don't be a phish: steps in user education, *in* 'ACM SIGCSE Bulletin', Vol. 38, ACM, pp. 237–241.

Roger Light (2017), 'paho-mqtt 1.1 : Python package index [online]', Available from: `https://pypi.python.org/pypi/paho-mqtt/1.1`. [Accessed: 2017-01-08].

Roman Haritonov (2017), 'requests-cache 0.4.13 : Python package index [online]', Available from: `https://pypi.python.org/pypi/requests-cache`. [Accessed: 2017-01-08].

Sanglerdsinlapachai, N. and Rungsawang, A. (2010), Using domain top-page similarity feature in machine learning-based web phishing detection, *in* 'Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on', IEEE, pp. 187–190.

scikit-learn (2017*a*), 'scikit-learn: machine learning in python [online]', Available from: `http://scikit-learn.org/stable/`. [Accessed: 2017-03-22].

scikit-learn (2017*b*), 'sklearn.feature_selection.rfe [online]', Available from: `http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html`. [Accessed: 2017-04-22].

SciPy Developers (2017), 'Scipy – scipy[online]', Available from: `https://www.scipy.org/`. [Accessed: 2017-03-22].

Shekokar, N. M., Shah, C., Mahajan, M. and Rachh, S. (2015), 'An ideal approach for detection and prevention of phishing attacks', *Procedia Computer Science* **49**, 82–91.

Sheng, S., Holbrook, M., Kumaraguru, P., Cranor, L. F. and Downs, J. (2010), Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', ACM, pp. 373–382.

Sheng, S., Wardman, B., Warner, G., Cranor, L. F., Hong, J. and Zhang, C. (2009), An empirical analysis of phishing blacklists, *in* 'Proceedings of Sixth Conference on Email and Anti-Spam (CEAS)'.

Stanberry, L. (2013), 'Chi-squared test', *Encyclopedia of Systems Biology* pp. 399–400.

Stanford Security Lab (1998), 'Spoofguard', Available from: `https://crypto.stanford.edu/SpoofGuard/`. [Accessed: 2016-11-15].

Stringhini, G., Hohlfeld, O., Kruegel, C. and Vigna, G. (2014), The harvester, the botmaster, and the spammer: on the relations between the different actors in the spam landscape, *in* 'Proceedings of the 9th ACM symposium on Information, computer and communications security', ACM, pp. 353–364.

Sun, Q., Fu, L., Qiu, W. and Sun, J. (2016), 'An automatic anti-attack scheme for mysql database', *Advances in Intelligent Systems Research* **113**, 397–400.

Suriya, R., Saravanan, K. and Thangavelu, A. (2009), An integrated approach to detect phishing mail attacks: a case study, *in* 'Proceedings of the 2nd International Conference on Security of Information and Networks', ACM, pp. 193–199.

The Apache Software Foundation (2017), 'httpd - apache hypertext transfer protocol server [online]', Available from: `https://httpd.apache.org/docs/2.4/programs/httpd.html`. [Accessed: 2017-01-08].

The CentOS Project (2017), 'Centos [online]', Available from: `https://www.centos.org/`. [Accessed: 2017-01-08].

tmux (2017), 'tmux [online]', Available from: `https://tmux.github.io/`. [Accessed: 2017-01-08].

Tourassi, G. D., Frederick, E. D., Markey, M. K. and Floyd, C. E. (2001), 'Application of the mutual information criterion for feature selection in computer-aided diagnosis', *Medical physics* **28**(12), 2394–2402.

Vasava, V. and Mangrule, R. (2015), 'Detection and prevention of javascript vulnerability in social media', *International Journal of Advanced Research in Computer Science and Software Engineering* **5**(5), 708, 713.

Vömel, S., Holz, T. and Freiling, F. (2010), 'I'd like to pay with your visa card: an illustration of illicit online trading activity in the underground economy'.

Von Ahn, L., Blum, M., Hopper, N. J. and Langford, J. (2003), Captcha: Using hard ai problems for security, *in* 'International Conference on the Theory and Applications of Cryptographic Techniques', Springer, pp. 294–311.

Wardman, B., Shukla, G. and Warner, G. (2009), Identifying vulnerable websites by analysis of common strings in phishing urls, *in* 'eCrime Researchers Summit, 2009. eCRIME'09.', IEEE, pp. 1–13.

Weider, D. Y., Nargundkar, S. and Tiruthani, N. (2008), A phishing vulnerability analysis of web based systems, *in* 'Computers and Communications, 2008. ISCC 2008. IEEE Symposium on', IEEE, pp. 326–331.

Whalen, T. and Inkpen, K. M. (2005), Gathering evidence: use of visual security cues in web browsers, *in* 'Proceedings of Graphics Interface 2005', Canadian Human-Computer Communications Society, pp. 137–144.

Whittaker, C., Ryner, B. and Nazif, M. (2010), 'Large-scale automatic classification of phishing pages'.

Winham, S. J., Freimuth, R. R. and Biernacka, J. M. (2013), 'A weighted random forests approach to improve predictive performance', *Statistical analysis and data mining* **6**(6), 496–505.

Xiang, G., Hong, J., Rose, C. P. and Cranor, L. (2011), 'Cantina+: A feature-rich machine learning framework for detecting phishing web sites', *ACM Transactions on Information and System Security (TISSEC)* **14**(2), 21.

Yang, Y. and Liu, X. (1999), A re-examination of text categorization methods, *in* 'Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval', ACM, pp. 42–49.

Yang, Y. and Pedersen, J. O. (1997), A comparative study on feature selection in text categorization, *in* 'Icml', Vol. 97, pp. 412–420.

Yue, C. and Wang, H. (2010), 'Bogusbiter: A transparent protection against phishing attacks', *ACM Transactions on Internet Technology (TOIT)* **10**(2), 6.

Zander, S., Nguyen, T. and Armitage, G. (2005), Automated traffic classification and application identification using machine learning, *in* 'The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l', IEEE, pp. 250–257.

Zhang, Y., Hong, J. I. and Cranor, L. F. (2007), Cantina: a content-based approach to detecting phishing web sites, *in* 'Proceedings of the 16th international conference on World Wide Web', ACM, pp. 639–648.

Zhu, J., Zou, H., Rosset, S. and Hastie, T. (2009), 'Multi-class adaboost', *Statistics and its Interface* **2**(3), 349–360.

Appendix A

Features

A.1 Sample of Features in Literature

Feature	Example/Notes	Study
Lexical Features		
Target organisation's domain	`paypal-verify.com`	Khonji et al. (2011)
Target organisation in path	`myfreeweb.com/paypal.com`	Garera et al. (2007)
No. of characters from end of organisation's name to end of hostname.	`paypal.account.verify-secure.com`	Garera et al. (2007)
Use of IP address as the host	`http://93.4.3.27/verify.php`	Abdelhamid et al. (2014)
Suspicious TLD used	`.tk, .hu, .info`	Ma, Saul, Savage and Voelker (2009*a*)
Number of dots in URL		Abunadi et al. (2013)
Number of slashes in domain	$n > 5$ slashes $\rightarrow$ Phishing	Lakshmi and Vijaya (2012)
Presence of hexadecimal encoding	`...secure.php?id=%6A%4E`	Lakshmi and Vijaya (2012)
URL Length		Abunadi et al. (2013)
Number of domain tokens	A URL can be split into tokens using a set of delimiters (?, _, ., =, &)	Le et al. (2011)
Longest domain token length		
Average domain token length		
Number of path tokens	The path of a URL can be split into tokens using a set of delimiters (?, _, ., =, &)	Le et al. (2011)
Longest path token		
Average path token		
Number of subdirectories	`phisher.com/dir1/dir2/dir3`	Le et al. (2011)
Length of longest subdirectory		Le et al. (2011)
Number of special characters in file name		Le et al. (2011)
Length of argument list	`index.php?arg1=s&arg2=sda`	Le et al. (2011)
Adding a prefix/suffix to a target organisation name	`account-paypal.com`	Abdelhamid et al. (2014)
Special characters in path		James et al. (2013)
Number of subdomains	Determined by number of dots in host	Mohammad et al. (2014)
Replacing characters with visually similar characters	`www.pavpal.com`	Aburrous et al. (2010)
Presence of @ symbol	`http://www.paypal.com@abc.com`	Aburrous et al. (2010)
Use of URL shortening	tinyurl, bitly	Mohammad et al. (2015)

Feature	Example/Notes	Study
Host & Domain		
Abonormal DNS Record	Empty/Incomplete or does not match the WHOIS identity	Pan and Ding (2006)
Age of domain	$n \leq 60$ days old → Phishing	Fette et al. (2006)
Rank by traffic	Used Alexa (Alexa, 2017) Rank	Mohammad et al. (2014)
PageRank (Brin and Page, 2016) value	PageRank of $n \leq 0.2$ → Suspicious, No PageRank → Phishing	Xiang et al. (2011)
Legitimacy of WHOIS information (ICANN, 2016)	No WHOIS data → Phishing	Lakshmi and Vijaya (2012)
Suspicious geolocation		Whittaker et al. (2010)
Security		
Lack of SSL/HTTPS		Aburrous et al. (2010)
Untrusted Certificate	Untrusted authority or Certificate age < 2 years → Suspicious, No certificate → Phishing	Mohammad et al. (2014)
Cookie holds foreign domain		Lakshmi and Vijaya (2012)
Script Analysis		
Using `onMouseOver` to hide links	`onMouseOver` AND changes status bar text → Phishing, just `onMouseOver` → Suspicious	Mohammad et al. (2014)
Changing link destinations		
Form Handler	`<form action='x'>` where `x` is one of (`about:blank`, `'return:void'`, an external resource) → Phishing	Alkhozae and Batarfi (2011)
Right click disabled		Mohammad et al. (2014)
Page Content		
Non-matching URLs	`<a href='paypal-secure-account.com'>paypal.com </a>`	Ramanathan and Wechsler (2012)
External Objects/Scripts	$22 < n \leq 61\%$ of objects/scripts are foreign → Suspicious, $> 60\%$ → Phishing	Mohammad et al. (2014)
External images		Ludl et al. (2007)
External favicon		Mohammad et al. (2015)
External links	I.e. links that take the user to foreign domain	Ludl et al. (2007)
Nil anchor	`<a href='about:blank'>...`	Lakshmi and Vijaya (2012)
Using Pop-up windows		Mohammad et al. (2014)
Contains an iframe		Abunadi et al. (2013)
Contains a suspicious entry fields	HTML text fields `input,textarea` that have labels such as 'password' and 'credit card'	Zhang et al. (2007)
Redirection	Contains code that redirects the user	Abunadi et al. (2013)

Table A.1: An aggregation of features used by various studies.

A.2 All Lexical Features

Feature	Type	Notes	Study
URL			
length_url	int		Abunadi et al. (2013)
num_dots_url	int		Abunadi et al. (2013)
contains_at	bool	Contains the @ symbol	Abunadi et al. (2013)
contains_hex	bool	Contains hexadecimal codes	Abunadi et al. (2013)
contains_ip	bool	Contains an IP address	Abunadi et al. (2013)
trigram_url	bool	Sum of probabilities[1] and Bag-of-words[2]	Darling et al. (2015)
quadgram_url	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
longest_token_url	int		Darling et al. (2015)
avg_token_len_url	float	Average URL token length	Darling et al. (2015)
num_non_alpha_url	int	Number of non-alphabet characters	Darling et al. (2015)
target_brand_garera	bool	Target brand in path but not in host	Garera et al. (2007)
sensitive_term_url	bool	`webscr, secure, banking, ebayisapi, account, confirm, login, signin, paypal, free, lucky, bonus index, includes, content, images, admin, file_doc, account, update, confirm, verify, secur, notif, log, click, inconvenien, urgent, alert`	Bergholz et al. (2008), Garera et al. (2007), Le et al. (2011), Lee et al. (2015), Ma, Ofoghi, Watters and Brown (2009)
hyphen_url	bool	URL contains a hyphen	Mohammad et al. (2014)
num_slashes_url	int		Lakshmi and Vijaya (2012)
slash_redir	bool	Contains a double slash redirection method	Mohammad et al. (2015)
contains_port	bool	Contains a port number	Le et al. (2011)
out_of_pos_tld	bool	A TLD present in non-TLD position	Xiang et al. (2011)
Hostname			
subdomain_presence	bool	Contains a subdomain	Canali et al. (2011)
unigram_hostname	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
bigram_hostname	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
trigram_hostname	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
quadgram_hostname	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
vowel_cons_ratio	float		Darling et al. (2015)
longest_token_hostname	int		Darling et al. (2015)
avg_token_len_hostname	float		Darling et al. (2015)
length_hostname	int		Darling et al. (2015)

[1] As explained in paragraph 2.7.1
[2] **Bag-of-words**: A binary feature is generated for each instance of the 'word' or 'token'

Feature	Type	Notes	Study
num_chars_after_target	int	Number of characters after the position of a target's brand name	Garera et al. (2007)
embedded_domain	bool	Contains a target's domain	Xiang et al. (2011)
Domain			
token_count_domain	int		Chu et al. (2013)
avg_token_len_domain	float		Chu et al. (2013)
longest_token_domain	int		Chu et al. (2013)
domain_brand_distance	int	Levenshtein Distance	Chu et al. (2013)
bigram_domain	bool	Sum of probabilities and Bag-of-words	Blum et al. (2010)
white_domain	bool	Uses a whitelisted domain	Garera et al. (2007)
mispelled_domain	bool	Mispelled in relation to a target's domain	Le et al. (2011)
num_hyphens	int		Le et al. (2011)
tld	bool	Bag-of-words representation	Ma, Saul, Savage and Voelker (2009*a*)
similar_char_repl	bool	Visually similar characters replaced	Aburrous et al. (2010)
Path			
last_path_token	bool	Bag-of-words representation	Blum et al. (2010)
token_count_path	int		Chu et al. (2013)
avg_token_len_path	float		Chu et al. (2013)
longest_token_path	int		Chu et al. (2013)
unigram_path	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
bigram_path	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
trigram_path	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
quadgram_path	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
digit_letter_ratio	float		Darling et al. (2015)
length_filename	int		Le et al. (2011)
num_delim_filename	int		Le et al. (2011)
organisation_in_path	int		Garera et al. (2007)
Query String/Parameters			
trigram_params	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
quadgram_params	bool	Sum of probabilities and Bag-of-words	Darling et al. (2015)
length_querystr	int		Le et al. (2011)
num_params	int		Le et al. (2011)
longest_query_value	int		Le et al. (2011)
max_num_delim_value	int	Maximum number of delimiters in query values	Le et al. (2011)

Table A.2: Aggregation of lexical features

A.3 URL Decomposition

```
url = [ protocolComposite ], urlSansProtocol ;

protocolComposite = protocol,"://" ;
protocol          = "http" | "https" ;

urlSansProtocol     = hostname, [ portComposite ], [ path ], [ query ] ;
hostname            = [ hostnameSansDomain, "." ], domain;
hostnameSansDomain  = str | hostnameSansDomain, ".", str;

domain        = domainSansTLD, ".", SLDTLD ;
domainSansTLD = str ;
SLDTLD        = [SLD, "."], [TLD];

portComposite = ":", number;

path      = directory, ["/", filename] | directory, [ "/" ] ;
directory = str[ "/", directory ] ;
filename  = str, [ ".", extension ] ;
extension = strSansDot

query      = ?, [ argsList ] ;
argsList   = keyValPair, [ "&", argsList ] ;
keyValPair =  str, [ "=", str ] ;
```

Figure A.1: The decomposition of a URL used in the process of feature extraction, shown using extended Backus-Naur form. `str` refers to any allowed character in a URL, and `strSansDot` is `str` without the dot.

A.4 Sensitive Terms

```
index, includes, content, admin, paypal, notif, secure,
account, webscr, login, logon, ebayisapi, signin, banking,
confirm, update, verif, secur, click, inconvenien,
authentica, recover
```

Figure A.2: The list of sensitive terms extracted from the works of Bergholz et al. (2008), Lee et al. (2015), Ma, Saul, Savage and Voelker (2009*a*), Zhang et al. (2007)

Appendix B

Inter-algorithm Comparison Results

	Phishing	**Benign**
Phishing	92704	7296
Benign	90269	9731

Table B.1: Confusion Matrix for Random Forest Model

	Phishing	**Benign**
Phishing	80408	19592
Benign	12654	87346

Table B.2: Confusion Matrix for Logistic Regression Model

	Phishing	**Benign**
Phishing	89533	10467
Benign	10451	89549

Table B.3: Confusion Matrix for Multi-layer Perceptron Model

	Phishing	**Benign**
Phishing	92151	7849
Benign	9464	90536

Table B.4: Confusion Matrix for XGBoost Model

	Phishing	**Benign**
Phishing	88189	11811
Benign	12402	87598

Table B.5: Confusion Matrix for AdaBoost Model

Appendix C

Comparison Against Existing Studies

Study	Type	Datasets	Dataset Weight	Algorithm	Acc.	Acc.	Prec	Prec	Recall	Recall	F1	F1	AUC	AUC
Pan and Ding (2006)	Full	**Phishing**: 50/25 (Millersmiles) **Benign**: 50/25 (Millersmiles)	100	SVM	0.952	0.867								
Sanglerdsinlapachai and Rungsawang (2010)	Full	**Phishing**: 75/25 (clean_mx) **Benign**: 75/25 (google)	150	Random Forest	0.915	0.945					0.914	0.942	0.971	0.988
Sanglerdsinlapachai and Rungsawang (2010)	Full	**Phishing**: 75/25 (clean_mx) **Benign**: 75/25 (google)	150	Neural Network	0.925	0.945					0.925	0.942	0.956	0.988
Zhang et al. (2007)	Full	**Phishing**: ?/? (Phishtank) **Benign**: ?/? (Alex, Proprietry)	100	CANTINA	0.955	0.905			0.970	0.900				
Mohammad et al. (2014)	Full	**Phishing**: 450/? (Phishtank) **Benign**: 450/? (Yahoo, startingpoint)	900	CBA	0.953	0.961								
Xiang et al. (2011)	Full	**Phishing**: 222/1997 (Phishtank) **Benign**: 768/1793 (Alexa, Yahoo)	990	Bayesian Network	0.964	0.872	0.994	0.977	0.935	0.775	0.960	0.864		
Ma, Saul, Savage and Voelker (2009*a*)	Full	**Phishing**: 550/550 (Phishtank) **Benign**: 750/750 (DMOZ)	1025	Logistic Regression	0.970	0.956								

Garera et al. (2007)	Full	**Phishing**: 822/423 (Proprietry) **Benign**: 833/429 (Proprietry)	1655	Logistic Regression	0.973	0.963			0.958	0.981			0.990	0.993
Miyamoto et al. (2008)	Full	**Phishing**: 1125/375 (Phishtank) **Benign**: 1142/381 (Alexa, Yahoo, 3Sharp)	2267	Random Forest	0.856	0.972	0.861	0.959	0.855	0.985	0.855	0.972	0.930	0.995
Miyamoto et al. (2008)	Full	**Phishing**: 1125/375 (Phishtank) **Benign**: 1142/381 (Alexa, Yahoo, 3Sharp)	2267	Neural Network	0.858	0.972	0.863	0.959	0.851	0.985	0.857	0.972	0.931	0.995
Miyamoto et al. (2008)	Full	**Phishing**: 1125/375 (Phishtank) **Benign**: 1142/381 (Alexa, Yahoo, 3Sharp)	2267	AdaBoost	0.859	0.972	0.861	0.959	0.855	0.985	0.858	0.972	0.940	0.995
Huang et al. (2012)	Lexical	**Phishing**: ?/? (Phishtank) **Benign**: ?/? (DMOZ, Yahoo)	3695	SVM	0.991	0.962							0.997	0.987
Ludl et al. (2007)	Full	**Phishing**: 612/68 (Phishtank) **Benign**: 3734/415 (google)	4346	C4.5	0.972	0.976	0.969	0.946	0.831	0.881	0.895	0.912		
Le et al. (2011)	Lexical	**Phishing**: 2200/1800 (Phishtank) **Benign**: 2200/1800 (Yahoo)	4400	Confidence Weighted	0.968	0.969								
Le et al. (2011)	Lexical	**Phishing**: 2200/1800 (Phishtank) **Benign**: 2200/1800 (Yahoo)	4400	SVM	0.900	0.970								
Lee et al. (2015)	Full	**Phishing**: 2700/300 (Phishtank) **Benign**: 2700/300 (DMOZ)	5400	Random Forest	0.982	0.955	0.981	0.938	0.983	0.974	0.982	0.956		

James et al. (2013)	Lexical	**Phishing**: 5344/8016 (Phishtank) **Benign**: 2803/4205 (DMOZ)	8147	J48	0.932	0.960	0.946	0.958	0.949	0.982	0.947	0.970		
Darling et al. (2015)	Lexical	**Phishing**: ?/? (Phishtank) **Benign**: ?/? (DMOZ)	10800	Logistic Regression	0.963	0.970					0.968	0.971		
Feroz and Mengel (2015)	Full	**Phishing**: ?/? (Phishtank) **Benign**: ?/? (DMOZ)	14400	Novel	0.985	0.973								
Gyawali et al. (2011)	Lexical	**Phishing**: ?/? (Phishtank) **Benign**: ?/? (spam_benign)	40000	SVM					0.925	0.940				
Whittaker et al. (2010)	Full	**Phishing**: 82947/20737 (Proprietry) **Benign**: $7 \times 10^6/2 \times 10^6$ (Proprietry)	7.5×10^6	Proprietry	0.999	0.983	0.975	0.919	0.950	0.889	0.962	0.904		

Table C.1: Data collected from studies where the various evaluation metrics are collected. Data set weight is the number of URLs in the training set. The data set names correspond to Table 3.1. A '?' symbol indicates that this information was not disclosed. Lexical classifiers are indicated by green cells.