UNIVERSITY OF BATH

Feature Selection
in
Online Lexical Phishing URL Classification

Oliver Youle
Bachelor of Science in Computer Science with Honours
The University of Bath
May 2017

Feature Selection in Online Lexical Phishing URL Classification

Submitted by: Oliver Youle

COPYRIGHT

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.

Signed:

Abstract

Online banking fraud is a significant source of criminal revenue in the modern world, causing losses of over 133M in the UK alone during 2015. We have seen a five-fold increase in the number of monthly phishing attacks reported since 2007, with 466,065 unique attacks detected in the second quarter of 2016. Current efforts to detect phishing websites draw upon a wide variety of sources for their data, including the HTML content of the website, and the DNS records associated with the domain. Although such approaches are precise, the processing rate is heavily limited by network latency and bandwidth.

In this dissertation, we investigate the classification of phishing websites using only lexical features derived from the URL itself. We implement an online random forest classifier, and using a novel lexical feature set we achieve a precision 0.98106 and a recall of 0.93594 at a processing rate that is over 660,000 times faster than current content-based approaches. In addition, the proposed system is also more robust to feature manipulation from the attacker and free from any potential security risks posed by fetching a URL and analysing the contents.

Contents

List of Figures

List of Tables

Acknowledgements

This project would not have been possible without the support of Netcraft, who provided me not only with inspiration, but the data upon which my results are founded.

Special thanks also go to Prof. Peter Hall for his supervision and guidance throughout the course of this project.

This project also makes use of the open source H2 database engine, as well as a number of libraries from the Apache Commons.

Chapter 1

Introduction

Contents

1.1 Background

1.1.1 Phishing: A Definition

The practice of phishing is a criminal mechanism which employs both *social engineering* and *technical subterfuge* in order to collect personal data or credentials through the impersonation of a trusted party (APWG, 2016; Whittaker, Ryner and Nazif, 2010).

This is achieved by directing the customers of some target organisation to a website that has been crafted to appear as similar as possible to the website of the organisation being targeted. An example of a phishing website is shown in figure 1.1. These websites exploit the trust that exists between the customer and the target organisation in order to convince the user to part with sensitive information—typically in the form of passwords or credit card details (OECD, 2009, p.29). Such details are then recovered by the attacker, who may in turn sell the details on to other fraudsters, or use them to defraud the victim themselves.

1.1.2 Common Phishing Practices

The OECD (2009) identifies a number of common methods that are used by fraudsters to direct potential victims to their phishing sites. Automation is commonly exploited in order to distribute large numbers of phishing URL[1]s to potential victims, with relatively little expense to the attacker (OECD, 2009, p.24). The most common of these methods are identified below:

a) *Spam E-mail.* Perhaps the most common and rapidly expanding vector for sending phishing URLs is through the use of spam e-mail (OECD, 2009, p.26). Here, attackers make use of compromised mail servers to send out large numbers of emails containing links to currently active phishing sites. Often, these e-mails will be created using the logos and slogans of a legitimate company (James, L. and Thomas, 2013), and usually include ungrounded threats, such as that of account termination, in order to persuade the user to take action (Ludl et al., 2007). The work of Dhamija, Tygar and Hearst (2006) notes that attackers will often forge the headers of such e-mails in order to alter, or "spoof", the sender address to appear legitimate to the untrained eye. Combining this seemingly legitimate address with a deceptive appearance and threatening language, spam e-mails form an inexpensive and effective method of not only distributing phishing URLs, but also of convincing the victim of the legitimacy of the site—perhaps even before they have visited it.

b) *SMiShing.* SMiShing is an emerging technique that allows phishers to target a larger range of devices (OECD, 2009, p.25). This technique involves sending out SMS messages to mobile phone users, the contents of which are similar to that of the spam e-mails discussed previously. SMS messages have the inherent downside of only containing textual information, however this is enough for the attacker to send messages purportedly claiming to represent some legitimate organisation, with a link to the URL of a phishing site. First seen in August 2006, this vector has been predicted to become increasingly popular (OECD, 2009), and has been observed in use as a method of malware distribution, as well as for phishing.

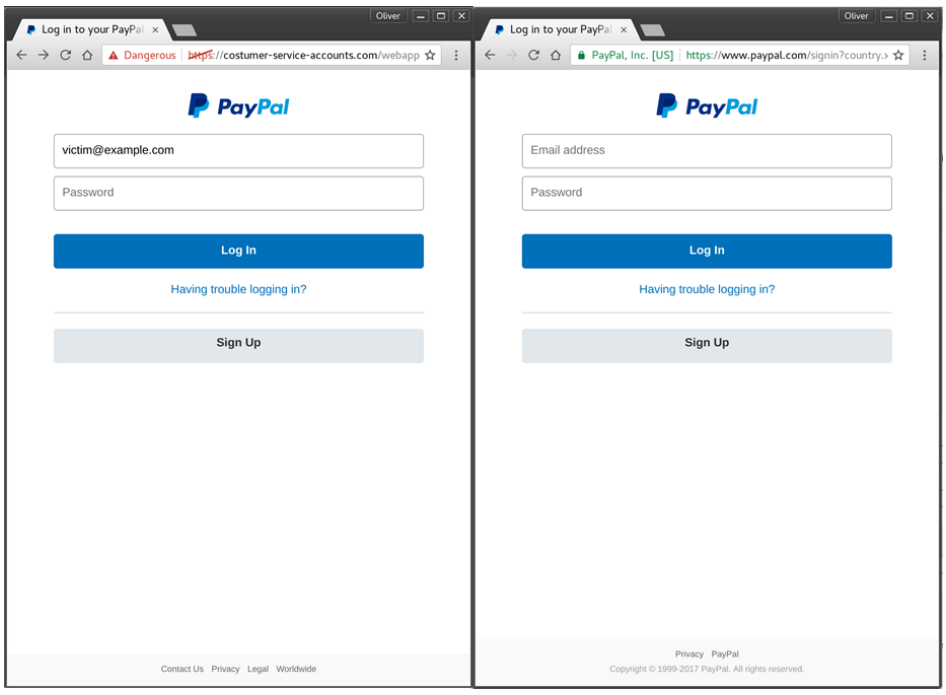
[1]Uniform Resource Locator, *RFC 3986: 1.1.3*

The OECD (2009) also defines a number of phishing practices that do not require the open distribution of phishing URLs:

a) *Vishing.* Vishing operates in a similar fashion to a traditional phishing attack, where an attacker will send out either deceptive e-mails or SMS messages, however, instead of inviting the recipient to visit a URL, the attacker provides a telephone number (OECD, 2009, p.25). A typical example of such an attack would request personal information or credentials in order to complete some bogus process, usually with the use of an automated attendant (OECD, 2009, p.25). It has been proposed that victims usually feel safer with a telephone call than a website, and are therefore more willing to share their details. The OECD (2009) also notes that in some cases the initial step is skipped altogether, and instead the attacker will adopt a more pro-active role of "cold-calling" the victims. This method of attack is interesting, as it exploits customers who are less confident with technology, and by taking a pro-active role the attacker gains an element of surprise over the victim, which may aid the deception.

b) *Pharming* Pharming utilises malware on the victim's machine in order to interfere with DNS[2] lookups in such a way as to redirect the user to an illegitimate version of the site that they were searching for (OECD, 2009, p.24). This method of attack has the advantage of using the victim's own intentions against themselves, as they are unlikely to suspect that the site is a forgery if they came to the site from a reputable source, such as a search engine.

[2]Domain Name System, *RFC 1035*



`https://costumer-service-accounts.com/webapps/3320c/index.php?u=victim@example.com`

Figure 1.1: A visual comparison between a phishing site (left) and the legitimate site being impersonated (right).

1.2 Motivation

1.2.1 A Rising Threat

A study by Dhamija, Tygar and Hearst (2006) has shown that the average user will fail to distinguish a legitimate site from a phishing site, even when they are aware that at least one of them is phishing. Their conclusion was attributed to a number of key factors:

a) A lack of understanding as to the structure and meaning of the different parts of a URL.

b) A lack of knowledge of security indicators and how to identify them, such as the presence of SSL[3].

c) The use of visually deceptive text in the URL, such as a close misspelling of a legitimate domain.

d) A deceptive look and feel to the website, and a lack of visualisation as to where their data is going.

e) A willingness to believe falsified security indicators, or a failure to recognise their absence.

This lack of understanding with regards to online safety, coupled with the rising popularity of the Internet has allowed phishing to become an increasingly popular vector of attack. Since 2007, the APWG[4] has reported an increase of nearly five-fold in the number of unique phishing sites detected in a single month, with 466,065 unique phishing sites detected in total for the second quarter of 2016 (APWG, 2007, 2016). Online banking fraud was estimated to have caused losses of at least 133M in the UK alone during 2015 (FFA, 2016), and with the number of attacks continuing to grow these losses will only increase unless further preventative measures are taken.

1.2.2 Phishing Countermeasures

All popular modern browsers implement URL blacklists as the primary method of protecting their users from phishing websites (Ludl et al., 2007; Whittaker, Ryner and Nazif, 2010). Blacklists are lists containing patterns that match on fraudulent URLs that have been discovered. If the user of a browser attempts to visit a URL that matches a pattern in the database, the browser will prevent access to the website—displaying a warning instead. This method of protection is only effective once the phishing site has been discovered, and as the life cycle of a typical attack lasts less than 58 hours (Anderson et al., 2013; Blum et al., 2010) it is critical that the blacklist is updated quickly. Current approaches attempt to address this through the application of machine learning to the process of identifying phishing websites (Whittaker, Ryner and Nazif, 2010).

Modern approaches to phishing site classification employ feature sets selected from a number of different sources. For example, Google's own phishing detection system utilises multiple

[3]Secure Sockets Layer Protocol *RFC 6101*

[4]Anti Phishing Working Group, *http://apwg.org/*

features from the site content, such as the proportion of external links, terms used in high frequency and the presence of a password field. The system also considers a number of binary features from the URL, as well as a number of proprietary statistics, producing a false positive rate of below 0.1% (Whittaker, Ryner and Nazif, 2010). This precision comes at a cost, however, as feature extraction requires a number of specialised operations, such as locally rendering the page content. Such operations cost time to implement and add an extra level of processing overhead that could adversely affect the rate of classification.

Fetching the content of a site presents a number of additional downsides. Most importantly, the inherent latency of network requests limits the rate of processing, which could impact the responsiveness of a blacklist-based system at large scales. The content-based approach described in the work of Whittaker, Ryner and Nazif (2010) reported a mean classification time of 76 seconds per URL. When scaled up to millions of potential phishing sites a day, the time spent classifying quickly becomes very large. Network load must also be considered, as well as the monetary cost involved in installing and operating an infrastructure capable of supporting large amounts of data transfer. Another, less economically focussed drawback of visiting the site is that it provides the attacker with an opportunity to evade detection—perhaps by responding with seemingly benign content, such as a 404 page. This is less than ideal, especially if the decision is weighted heavily on the content of the page, as an attack using such an approach may not be caught as a result.

1.2.3 Uncertainty in Lexical Feature Sets

Previous studies have suggested that the limitations of content-based approaches could be avoided by using a feature set derived purely from the URL (Ma et al., 2011; Le, Markopoulou and Faloutsos, 2011). These features are collectively described as *lexical* features. A wide variety of lexical features have been suggested, which we separate into two categories: *simple* lexical features and *situated* lexical features. Definitions for these are given in section 3.2.1.

Deciding on the importance of a feature, along with its relevance in the presence of other features remains an ongoing research issue. For example, the number of dots in the hostname—used in the work of Ma et al. (2011)—may be indicative of an attempt to hide an unrelated domain from the view of the victim in the following manner:

`http://login.my-bank.com.a5b32y3n86s8d{...}8j34jk35.example.com/`

This feature provides very little information, however, in the case where a domain has been registered with the intention of fraud:

`http://my-b4nk.com/login/`

This feature may even become irrelevant in the presence of a different feature—for example, in this case, when used in conjunction with the length of the hostname.

We can demonstrate this uncertainty by applying principal component analysis to the simple lexical features described in the work of Ma et al. (2011). The length of the URL, the number of dots within the hostname and the length of the hostname are calculated for a random sample of 1000 phishing sites and 1000 benign sites from the April 2016 training data (for details on this data, see section 7.1). These features are then normalised to a value between 0 and 1 and plotted using principal component analysis to project the data into

two dimensions, the results of which can be seen in fig. 1.2. This simple exercise shows that although there is some discrimination between the two classes of URL, these features alone are not enough to separate the data. It is often the case that a study will suggest particular lexical features with little empirical reasoning as to the choice. This is particularly prevalent in the case of online learning, as previous studies—such as the work of Blum et al. (2010)—often fail to address the computational cost of their suggested methods.

In this dissertation we seek to address the issue of uncertainty in the selection of lexical features for online learning by providing empirical evidence in favour of a particular method of lexical feature selection.

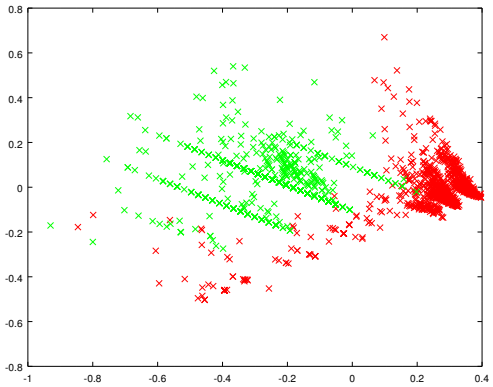


Figure 1.2: A plot of the data spread between phishing and non-phishing using URL length, number of dots within the hostname and hostname length. Green indicates benign sites, and red indicates phishing sites.

1.2.4 Online Learning

Online learning algorithms have been shown to have high accuracies and low latencies at large scales Ludl et al. (2007); V and A (2014). They are well-suited to phishing URL classification as they can train on an incoming stream of data, allowing them to stay up-to-date without having to store large numbers of URLs to support batch learning. Online learning in the context of phishing URL classification has been explored in previous work, however these studies either did not been test the system over large, real world data sets (Blum et al., 2010), or have false positive rates that are too high to be feasible in a real world application (Zhao and Hoi, 2013).

Our work expands upon this area of research by implementing an online classifier that has not yet been applied to this problem, performing an empirical evaluation of our proposed feature sets using a large, real world data set of over 7.5 million URLs.

1.2.5 Dissertation Focus

This dissertation seeks to investigate whether lexical features from the URL can be leveraged to produce a suitably accurate classifier in an online setting using real-world data. In order to achieve this, we shall consider a number of different methods of feature extraction, providing novel suggestions in order to address their shortcomings and supporting this with empirical evidence. If successful, processing latencies could be significantly reduced, which would increase the number of users protected by blacklist based systems—as well as reducing infrastructural and operational costs. Furthermore, the resulting system would be more robust to feature manipulation from the attacker and free from any potential security risks posed by fetching a URL and analysing the contents.

1.3 Hypothesis

Our hypothesis for this dissertation is as follows:

Lexical features drawn purely from the URL are sufficiently discriminative to allow for the accurate classification of phishing websites in an online setting. The use of these lexical features will decrease processing latency whilst maintaining a precision and recall similar to that of current content-based approaches.

1.4 Aim

The aim of this dissertation is to investigate the feasibility of purely lexical feature sets in the context of phishing URL classification in an online setting, and to provide novel recommendations on both a suitable classifier and feature extraction methods through empirical evaluation using a large, real-world dataset.

1.5 Objectives

I. **Identification of existing feature extraction methods**. The existing work regarding the classification of phishing URLs will examined to provide an overview of the current sources and methods of feature extraction, as well as the classification mechanisms used.

II. **Critique and extension of existing lexical feature sets**. Once the existing literature has been examined, critique of the different feature sets and experimental designs will be provided. This critique will then inform the proposal of at least one novel lexical feature set.

III. **Implementation of an online classifier**. Research into online classification methods will be conducted. At least one online classification method will be selected for implementation. The selected classification method(s) must not have been applied to the task of phishing URL classification in previous work. The selected method(s) will be implemented, the details of which shall be documented.

IV. **Experimental design**. An experiment will be designed to investigate the hypothesis. This experiment will use a large, real-world dataset and shall evaluate the performance of the selected classifier(s) over time with varying feature sets.

V. **Evaluation of the proposed feature sets**. The results from the experiment designed in objective 4 will be used to critically evaluate the proposed feature sets with respect to existing methodologies—providing empirical evidence to support or challenge my hypothesis.

1.6 Summary

In this chapter we have outlined various phishing practices and identified them as a significant threat to businesses and web users alike. The problem of phishing site classification using machine learning has been proposed, and various state-of-the-art methods have been briefly discussed. It was shown that whilst accurate, content-based methods of phishing site classification are often slow—impacting the effectiveness of blacklist-based solutions and increasing operational costs. Lexical URL analysis was proposed as a potential solution to these issues, as it does not require costly network requests to operate. Previous work has suggested a wide variety of lexical features, but often little time is spent on understanding the effectiveness of individual features, or how they interact once combined. The current uncertainty surrounding the selection of lexical features for online learning was identified as the motivation for the project, and the following hypothesis generated:

Lexical features drawn purely from the URL are sufficiently discriminative to allow for the accurate classification of phishing websites in an online setting. The use of these lexical features will decrease processing latency whilst maintaining a precision and recall similar to that of current content-based approaches.

From this hypothesis, the aim of the project was identified, along with five objectives to define and constrain the scope of the project.

1.7 Dissertation Structure

In chapter 2, we critically examine the existing literature surrounding the classification of phishing sites and provide a justification of the contributions made in our work. Chapter 3 concerns lexical feature sets in more detail, and defines two different categories of lexical features: simple lexical features and situated lexical features. Chapters 4 and 5 examine simple and situated features in more detail. Here potential limitations of existing methods are identified, and novel solutions proposed. Chapter 6 covers online learning, and details our implementation of an online random forest classifier. In chapter 7, we cover the optimisation of our classifier, and the design and execution of an empirical evaluation into the performance of the various feature sets described in our work. Chapter 8 forms our concluding chapter, and reflects upon the extent to which our aim and objectives are met, as well as reflecting upon what we have learned.

Chapter 2

Literature Survey

Contents

2.1 Existing Literature

2.1.1 Feature Sources

Phishing site URL classification has received a reasonable amount of attention from the research community. A number of different methodologies already exist, however, many are not without their limitations. In order to critically examine each of these methodologies, it is useful to first discuss the different sources of features that are available from a URL.

(a) The first, and perhaps most obvious, source of features is the URL itself. Features which have been obtained directly from the URL shall be referred to as *Lexical Features*.

(b) The second source of features used to classify a phishing URL are known as *Host Based Features*. James, L. and Thomas (2013) defines host based features to be features which "explain "where" phishing sites are hosted, "who" they are managed by, and "how" they are administered.". Such features can be obtained by performing certain specialised requests, such as forward and reverse DNS resolution requests or WHOIS lookups.

(c) A third source of features is the content of the web page that is returned by the hosting web server when the phishing URL is requested. Features derived from the web page of the URL being classified will be referred to as *Content Based Features*.

(d) The final source of features used for classifying phishing URLs involves making requests to a third party. Examples of this include the Google PageRank (Brin and Page, 1998), and the Alexa Site Rankings[1]. Such features shall be referred to as *Third Party Heuristics*.

2.1.2 Early Work

Phishing URL Categories and First Steps

In 2007, Garera et al. (2007) proposed a framework for detecting phishing sites which laid the groundwork for much of the research in this area. In order to further understand how phishing URLs might differ from those of benign sites, they identified four separate categories of phishing URLs.

(a) **Obfuscation of the hostname with an IP address**. These forms of attack use the IP address of the fraudulent site in place of the domain name, often including the name of the target organisation in the path instead.

(b) **Obfuscation of the hostname with a legitimate domain**. Here, a valid looking domain name is used, usually with the name of the target organisation in the path. This is usually done in order to imitate a URL that would contain a redirect.

(c) **Obfuscation with large hostnames**. These attacks will include the name of the target organisation, or a related phrase, in the hostname, with a large string of subdomains appended in order to push the actual domain off of the end of the URL bar in the victim's browser.

[1] Alexa site ranking: http://www.alexa.com/topsites

(d) **Domain name unknown or misspelled**. Here, a close misspelling of the target domain has been registered in order to make the URL appear legitimate at first glance. Unknown domains, or URLs which do not fit into any of the above categories, are also included in this final category.

Using the four categories, Garera et al. (2007) identified a number of lexical features within the URL with the aim of finding features that could best represent these different forms of attack. Such features included the presence of an IP address in the hostname, whether a target organisation is mentioned in the path but not the host, and the number of characters present in the hostname after a mention of an organisation name. The domain name was also checked against a white list of known benign domains, and whether it matched or not was used as a feature.

These lexical features were then combined with a number of third party heuristics. It was assumed that, as many phishing sites are short-lived, the pages themselves would not yet have been indexed by search engine web crawlers. Using this, the Google crawl database was queried to find whether or not the URL was present in the site index. It was found that 39.5% of the benign URLs in the training set were present in the site indices, as opposed to only 8.2% for the phishing URLs (Garera et al., 2007). The absence of a URL from the crawl database index, the PageRank of both the host and the URL, along with the Google indexing quality scores were all chosen as features for classification.

Along with the lexical features and third party heuristics discussed above, Garera et al. (2007) also identified a number of common "suggestive" word tokens in their training set. Using a sub-string extraction algorithm, tokens were extracted from the URLs within the phishing training set, and their frequencies recorded. After discarding any tokens with a length less than five characters, and any organisation names, the eight most frequent tokens were selected. These tokens included words such as "signin" and "secure" (Garera et al., 2007). The presence of each of these eight tokens were used as binary features for the classifier, bringing the total number of features to 18.

A logistic regression classifier was trained and tested over a labelled set of 2508 URLs, 1245 of which were phishing, and 1263 were benign with a 66% to 44% training to testing split (Garera et al., 2007). The classifier was then evaluated "over multiple runs with randomized partitioning", yielding an average accuracy of 97.31%, with a true positive rate of 95.5% and a false positive rate of 1.2% (Garera et al., 2007).

Although a reasonably high accuracy was achieved with this method, it is required that both a list of target organisations, as well a white list of legitimate domains, is maintained. This inherently limits the scalability of this solution, as generating and maintaining such lists would likely require human involvement. The use of the presence of suspect tokens allows for the exploitation of the use of language within URLs, however the selection of a limited number of tokens does not allow for the system to respond to a change in frequency of these tokens. The training and testing sets are also relatively small, at only 2508 URLs in total, which may not be enough to provide a representative sample. The use of third party heuristics also requires that network requests be made for every URL that is classified. This may further limit the scalability of the solution, especially if rate limits are applied to the use of such heuristics. There is also the concern that if the third party decides to no longer provide the information, then the feasibility of the classification method would need to be re-evaluated. Finally, the false positive rate achieved of 1.2% is too high to be feasible for a large scale solution (Whittaker, Ryner and Nazif, 2010).

CANTINA - A Content-Based Approach

Around the same time, Zhang, Hong and Cranor (2007) proposed a content based framework for the classification of phishing URLs, known as CANTINA. This approach operates on the same assumption that was made by Garera et al. (2007) that a search engine, such as Google, will index the vast majority of legitimate sites, but that many short-lived sites—in particular, phishing sites—will not appear in the search indexing. This framework uses term frequency-inverse document frequency (TF-IDF) to analyse the relative importance of the different terms within the HTML content of the phishing page. The five terms with the highest TF-IDF weightings are selected and treated as a "lexical signature". This lexical signature is then queried using the Google search engine, and the domain of the URL being classified is compared with those of the top N search results. If the domain appears in the top N results, it is treated as legitimate, otherwise it is considered to be a phishing site. The method described above was tested on a set of 200 URLs, 100 benign and 100 phishing. A true positive rate of 94% was achieved, however, a false positive rate of 30% was also observed (Zhang, Hong and Cranor, 2007).

In order to reduce this false positive rate, the domain of the URL being classified was also included in the lexical signature. This had the effect of reducing the false positive rate from 30% to 10%, increasing the accuracy to 97%. Following this, Zhang, Hong and Cranor (2007) proposed an expansion of the TF-IDF methodology by incorporating several additional "heuristics" in order to further reduce the false positive rate. These heuristics included the age of the domain, whether the URL contains an '@' symbol, or a '-', whether the URL has an IP address in place of the domain, the number of dots in the URL, and whether the page content contains a text entry field. Each heuristic was assigned a weight that was calculated from 100 phishing sites from PhishTank, and the same 100 benign URLs that were used in the earlier evaluations. A forward linear model was used to classify a new set of 200 URLs (again with a 50-50 phishing to benign split). By including these heuristics, the false positive rate was decreased from 10% to 1%, with a slight decrease in the true positive rate from 97% to 89% (Zhang, Hong and Cranor, 2007).

Zhang, Hong and Cranor (2007) acknowledged a number of drawbacks with this method. Firstly, the use of TF-IDF means that a dictionary must be maintained for the classification mechanism to work. This also limits the system to only pages which are using the same language as the dictionary. In particular, it was discovered that TF-IDF does not perform well with east Asian languages (Zhang, Hong and Cranor, 2007). It was also noted that the speed of classification was greatly limited by the "time lag involved in querying Google" (Zhang, Hong and Cranor, 2007). Similarly to the work of Garera et al. (2007), the use of a third party system also means that an implementation would rely on the continued availability of the third party service, and could potentially be affected by rate limiting measures, which is not suitable for a large scale system. The training and testing sets used in the evaluation are also relatively small, with only 200 samples in each, and so it is hard to say whether the results achieved would be reproducible over larger data sets. Finally, by fetching and analysing the content of the web page itself, not only does the system incur another lengthy time delay, but it also allows the attacker an opportunity to evade detection. For example, if an attacker was to recognise the IP address of the fetching system, they could serve a benign response, such as a 404 page, or they could design the phishing site to use images instead of text such that no lexical signature could be generated.

Evaluating CANTINA

To address the concerns over the small data samples used in the evaluation of CANTINA, Miyamoto, Hazeyama and Kadobayashi (2009) performed a more rigorous evaluation of the CANTINA framework over multiple different machine learning methods. A slightly larger data set of 3000 URLs was used, with a 50:50 phishing to benign URL split. The machine learning methods evaluated included AdaBoost, Bagging, SVM, CART, Logistic Regression, Random Forests, Neural Networks, Naive Bayes and BART. The f_1 measure of each of these methods was calculated with four-fold cross validation, repeated ten times to achieve an average result (Miyamoto, Hazeyama and Kadobayashi, 2009). The f_1 measure is defined by the following:

$$2 \cdot \frac{p \cdot r}{(p + r)}$$

Where p is the precision and r the recall. The original implementation of CANTINA was found to have an f_1 value of 0.7606 (Miyamoto, Hazeyama and Kadobayashi, 2009). The highest f_1 measure achieved was 0.8581 with AdaBoost, with Bagging, Logisitic Regression, Random Forests, Neural Networks, Naive Bayes and BART also outperforming the original CANTINA implementation (Miyamoto, Hazeyama and Kadobayashi, 2009).

A number of the content based features from CANTINA, along with a couple of lexical features first seen in the work of Garera et al. (2007) were combined with host based features in the work of Aburrous et al. (2010). The focus of this work was to investigate the effectiveness of various data mining techniques for the detection of phishing websites. A number of lexical features were identified, some of which have already been seen in the work of Garera et al. (2007), such as the use of an IP address in place of a hostname, the length of the hostname, and the use of suspicious characters such as '@' and '-'. Along with these, a number of novel lexical features were used, including the use of hexadecimal characters in the URL, and the total length of the URL as a whole. In addition to the lexical features, this approach also considered content based features such as the use of forms with submission buttons, the presence of spelling errors, and the use of pop-up windows. Interestingly, a social-human factor was also considered here. In a similar approach to Garera et al. (2007), a number of words and phrases were identified relating to an emphasis on security, as it was assumed that many phishing sites would use such phrases in an attempt to convince their victims that the site was legitimate. The presence of such words was used as a feature, along with the use of generic salutations (Aburrous et al., 2010). Host based features were also considered, including the content of the DNS records, along with the details of any SSL certificates present.

These features were then used with jRIP, PART, PRISM, C4.5, CBA and MCAR over a data set of 1006 URLs. MCAR was found to have the highest accuracy of 88.1% (Aburrous et al., 2010).

Although the use of host based features give us a greater amount of information to work with, the effect of network latency on the classification process is not addressed by this method, as both the page content and DNS records must be requested over the network. By moving away from the use of TF-IDF, the solution proposed by Aburrous et al. (2010) removes the dependency on third party systems that was seen in the work of Zhang, Hong and Cranor (2007), and also reduces the issue of language dependence, though it does not

remove it entirely. The overall accuracy achieved is slightly higher than that of the original CANTINA implementation, however it does not achieve false positive rates as low as those seen in the work of Miyamoto, Hazeyama and Kadobayashi (2009). It is clear that reducing the proportion of content based features has reduced the overall accuracy of the system, however, it was also found that there is a significant relation between the lexical features and the presence of SSL for identifying phishing sites, and that the content based features used provided little to no influence on the outcome (Aburrous et al., 2010). This suggests that the URL alone may contain enough information for accurate classification.

CANTINA Revisited

Following this, Xiang et al. (2011) revisited CANTINA, this time incorporating features described by Garera et al. (2007), as well as applying a number of different machine learning techniques. The issue of latency was addressed by the inclusion of a filtering stage before URLs are submitted for classification. Here, the content of the page is hashed and compared to the hashes of previously classified sites. This allows near-duplicate phishing sites to be classified without having to calculate the TF-IDF values and perform the associated search query to Google. In addition to the filtering stage, a number of new lexical features are proposed. The presence of the 'sensitive' terms in the URL, identified in the work of Garera et al. (2007), are proposed as eight new binary features for the classifier, along with the presence of an out of position brand name, and the presence of an embedded domain in the URL path. The content based feature set is also expanded to include the presence of forms with mentions of terms related to credentials, input tags and no SSL, as well as the presence of non-absolute action fields and whether or not the most frequent domain in the hyper-links coincides with the domain of the URL. The lexical signature generated from the TF-IDF analysis was also updated to include the name of the company specified in the copyright footer of the page content.

These new features were then applied to an SVM, Logistic Regression, Bayesian Network, J48 Decision Tree, Random Forest and AdaBoost and evaluated over a much larger sample of over 8000 URLs, collected over an extended period of time. The Bayesian Network was found to perform the best with the given feature set, achieving a 92.54% true positive rate, and a 0.407% false positive rate (Xiang et al., 2011). The Bayesian Network was then trained on each feature individually in order to evaluate the contribution of each feature to the overall performance. It was found that the most influential features were those derived from the page content, such as the presence of suspicious login forms, along with the third party heuristics; such as PageRank—or the presence of the URL domain in the Google search results. Lexical features, such as those proposed by Garera et al. (2007), or the number of dots in the URL, were shown to be of limited contribution, which appears to contradict the findings of Aburrous et al. (2010).

Whilst the introduction of the filtering stage seeks to address the issues of latency seen in the original implementation of CANTINA, the work by Xiang et al. (2011) still requires that the page content is fetched, which means that this issue of latency is only mitigated slightly. It also fails to address the previously identified concern that in fetching the content of the site, the attacker is provided with an opportunity to evade detection.

2.1.3 Third Party Heuristics

The use of third party heuristics, such as Google Pagerank, or the use of search engines (as seen in the work of Zhang, Hong and Cranor (2007) and Xiang et al. (2011)), has been explored in a variety of different ways, with the aim of reducing false positive rates.

Exploiting Search Engines

Khonji, Jones and Iraqi (2011) proposed a new method of identifying out of position domain names in URLs that does not have the disadvantage of maintaining a list of target organisations that was observed in the work of Garera et al. (2007). The approach proposed the use of rank-based heuristics that take into account the number of domain names within a URL. If an out of place domain in a URL has a higher rank than that of the actual domain, then the rank of the actual domain will fall by some amount. If the rank of the domain falls below a certain threshold, then the URL is considered to be phishing. Ranks are calculated based on the number of results obtained from a Google search query of the domain, where sites on the domain itself are excluded from the results. Using this method, an 83.31% true positive rate was achieved over a data set of 220,000 URLs, though the false positive rate was unacceptably high at 27.34% (Khonji, Jones and Iraqi, 2011). It was also noted that issues were experienced with Google rate-limiting the search requests, which limits the speed and scalability of this solution.

Another method of phishing URL classification using search engines was proposed by Huh and Kim (2012). Here, a number of popular search engines are queried for the full URL that is being classified. The assumption put forward by Huh and Kim (2012) is that phishing websites are often not indexed, and will have a low number of results. The total number of results returned by each query, along with the ranking of each result is recorded, and is then used as a feature for classification. Results from Google, Bing and Yahoo were used to form feature vectors of six features each. A data set of 200 URLs, with a 50:50 phishing to legitimate split, was used to test and train an LDA classifier, Naive Bayes, K-Nearest Neighbours, and an SVM. The highest accuracy achieved was 98% with the K-Nearest Neighbours classifier. Similar to the work of Zhang, Hong and Cranor (2007), Xiang et al. (2011) and Khonji, Jones and Iraqi (2011), the use of third party search engines greatly limits the scalability of this solution due to rate-limiting, and the use of multiple search engines only compounds this. The sample size used in the evaluation is also small, and the assumption that all phishing sites will not be indexed or have a low number of results is not necessarily true, particularly in the case where a legitimate site has been compromised by the attacker.

Another application of search engines to the problem of phishing URL classification can be found in the work of Nguyen et al. (2014). Using the top level domain, primary domain, hostname and path, it was found that the Levenshtien distance between these values and the Google search engine spelling suggestions for each one could be used to identify phishing URLs. This method makes the assumption that the majority of phishing URLs are *type d* URLs (Garera et al., 2007), where the URL contains close misspellings of the target domain name. This feature was augmented with the use of third party heuristics, namely the PageRank of the URL, as well as its Alexa rank and reputation rating. A weight based approach was used for classification, reporting an accuracy of 97.6% over a data set of 16,600 URLs. This approach is interesting, as it is the first we have seen that attempts to address the detection and classification of *type d* phishing URLs.

The work of Nguyen et al. (2014) was then later built upon by Kausar et al. (2014), who combined the features previously used by Nguyen et al. (2014) with a selection of lexical features drawn from the work of Xiang et al. (2011). These lexical features include whether or not an IP address is present instead of a hostname, the number of dots in the URL, the number of special characters, such as '@' or '-' in the URL, and the number of slashes. Using these features with a Naive Bayes classifier produced a 48% accuracy, with the heuristics on their own producing an accuracy of 87.5%. These results are surprising, however, with a sample size of only 300 URLs, it is likely that the data set is not representative, and that larger scale testing is required to determine the effect of including simple lexical features in the feature set.

The Dangers of Third Party Heuristics

More recently, Feroz and Mengel (2015) proposed a method of phishing URL classification using the Microsoft Reputation Services[2] (MRS). Using this method, each URL is submitted to the MRS system, which returns a number of categories that have been associated with the content on the site. Each type of category was placed into one of three bags: benign, moderate and severe, the latter being for the categories most likely to be malicious. As each URL may return multiple categories, the category with the highest severity was selected and used as a feature for the classifier. This feature was then combined with a number of lexical and host based features drawn from the work of Ma et al. (2011), and used with a K-Means classifier to achieve an accuracy of 98.46% (Feroz and Mengel, 2015). Unfortunately, on December 31st 2015, the Microsoft Reputation Services were shut down. This aids to demonstrate that the largest downside to incorporating third party heuristics into a classification system is the reliance on said third party to continue to provide the heuristic throughout the lifetime of the system.

2.1.4 Large Scale Classification

Real world implementations for large scale phishing URL classification classically use a combination of all of the previously discussed feature sets. For example, Google's own phishing URL classification uses content based features, such as the terms with the highest TF-IDF values, first used by Zhang, Hong and Cranor (2007), (Whittaker, Ryner and Nazif, 2010). Tokens are generated from the URL and used in a bag of words approach similar to the work of (Baykan et al., 2009). Third party heuristics, such as the PageRank of the site—first proposed in the work of Garera et al. (2007)—, along with proprietary meta-data are also used (Whittaker, Ryner and Nazif, 2010). Host based features, including those used by Aburrous et al. (2010) and Xiang et al. (2011), such as the geolocation of the host an name server are included in the feature set as well (Whittaker, Ryner and Nazif, 2010). All features are then scaled to a value between 0 and 1, and a logistic regression model used to classify incoming URLs. Using this method, a 94.97% true positive rate is achieved, with a false positive rate of only 0.03% (Whittaker, Ryner and Nazif, 2010).

This precision comes at a cost, however, as the process of extracting these features requires a number of specialised operations—such as locally rendering the page content (Whittaker, Ryner and Nazif, 2010)—that cost time to implement and add an extra level of processing overhead that could adversely affect the rate of classification. As discussed in sections 2.1.3

[2]Microsoft Reputation Services: https://www.microsoft.com/security/portal/mrs/

and 2.1.5, the inclusion of host based and content based features means that the classification system is also affected by network latency. This is reflected in the median processing time reported of 76 seconds per URL (Whittaker, Ryner and Nazif, 2010), which, when scaled to "millions of potential phishing sites a day", quickly becomes very large. Another major disadvantage of this approach is that by using a batch learning algorithm, the model must be retrained daily using the data from the past three months (Whittaker, Ryner and Nazif, 2010).

2.1.5 Purely Lexical Feature Sets

As has been discussed in sections 2.1.2 and 2.1.3, the use of host based and content based features, along with third party heuristics, allows for accurate classification, but has a number of downsides. Most notably, the inherent latency of the network requests that are required in fetching these features limits the rate of processing (Zhang, Hong and Cranor, 2007), which could impact the responsiveness of a system at large scales. This latency is especially prevalent in the use of third party heuristics, as such requests may be rate limited by the organisation providing the data (Khonji, Jones and Iraqi, 2011). Another danger of using third party heuristics is that the availability of the data is subject to change, and is often out of the control of the user (Feroz and Mengel, 2015). This is a problem for large scale systems, as a change in the availability of some of the features used for classification would require that the entire system be re-evaluated. Network load must also be considered, as well as the monetary cost involved in installing and operating an infrastructure capable of supporting large amounts of data transfer. Another, less economically focussed, drawback of visiting the site is that it provides the attacker with an opportunity to evade detection—perhaps by responding with seemingly benign content, such as a 404 page, or simply by replacing any textual data with images (Zhang, Hong and Cranor, 2007). This is less than ideal, especially if the decision is weighted heavily on the content of the page, as an attack using such an approach may not be caught as a result.

Are Lexical Features Alone Sufficient?

Le, Markopoulou and Faloutsos (2011) studied whether or not lexical features alone could be feasible for accurate phishing URL classification. Using the four different types of phishing sites defined by Garera et al. (2007), a number of hand-selected lexical features were defined for the different regions of the URL.

(a) The features defined for the full URL are the length of the URL, the number of dots in the URL, and the presence of any blacklisted words.

(b) The features defined for the hostname are the presence of an IP address, port number, the number of tokens, the number of hyphens and the length of the longest token.

(c) The features identified from the URL path are the length of the first directory, the number of sub-directory tokens, the length of the longest token, the maximum number of dots in any token and the maximum number of other delimiters in any token.

(d) The features proposed for the file name of the URL are the length of the file, and the number of dots in the file name.

(e) For the argument section of the URL, the total length of the arguments, the number of variables, the length of the longest variable and the maximum number of delimiters in any variable are all defined as features.

These hand-selected features were combined with the token based binary features used in the work of Ma et al. (2011), and compared against the same selection of features, but with the addition of the registration details from the WHOIS, along with geolocation data. A number of different machine learning techniques were evaluated, including an SVM, Online Perceptron, a Confidence-Weighted (CW) approach, and an Adaptive Regularization of Weights (AROW) method. Using lexical features alone, an accuracy of 97% was achieved using AROW over noisy data (Le, Markopoulou and Faloutsos, 2011). This shows that lexical features alone can indeed be leveraged to produce high classification accuracies, however, it is unclear as to the size of the data sets that were used for training and testing in this study.

From URL Topic Classification to Fighting Phishing

The use of pure lexical feature sets for URL topic classification has been explored in the work of Baykan et al. (2009, 2011). Taking inspiration from document classification, they investigated the application of a bag-of-words approach to examine the contents of the URL. In their approach, each URL is represented as a histogram of its comprising segments. Feature vectors are constructed by comparing this histogram against a dictionary of terms. They experimented with two different methods of producing segments from the URL. First, URLs were first split (*tokenised*) into simple tokens on any non-letter character. N-gram segments were then constructed using sequences of consecutive tokens. A second approach was also considered, whereby all punctuation and numbers are removed from the URL, and n-gram segments are generated from the resulting string using characters instead of tokens. For each approach, positional information was also explicitly encoded by duplicating each n-gram segment, or token, and appending the position in the form of a numeric character. Each of these feature sets were applied to a Naive Bayes classifier, SVM, Maximum Entropy classifier and various forms of boosting.

A large training set of over 2 million URLs was used to evaluate each classifier. The token based approach was shown to yield the lowest accuracy, whilst 6-grams were found to be the most effective of the n-gram from token approaches. A mixture of 4-, 5-, 6-, 7-, and 8-grams, known as *all-grams* was found to give a greater accuracy than any one set of n-grams, and all-grams with explicitly encoded positional data performed better than all-grams without this encoding. Overall, the highest accuracy achieved was 82% with all-grams generated from the URL (Baykan et al., 2011). Following this, the outputs from the Naive Bayes, SVM and Maximum entropy classifiers using all-grams from both the URL and tokens were collected into feature vectors of 6 features, and a number of boosting meta-classifiers were used to improve the classification accuracy. Using this approach, ModestAdaBoost was shown to increase the classification accuracy, but only by a couple of points (Baykan et al., 2011).

The techniques described in the work of Baykan et al. (2009, 2011) allow the classifier to capture the use of language within URLs, making it a promising candidate for phishing URL classification. The work of Ma et al. (2011) took the token based bag of words approach proposed by Baykan et al. (2009), combined it with a number of simple lexical and host based features, and applied it to the problem of phishing URL classification. The proposed system used the length of the hostname, the length of the URL and the number of dots in

the URL as features, along with binary features for each token derived from the URL. Tokens were generated by splitting the hostname on the "." character, and the path by any of the following: "/", "?", ".", "=", "-" or "_". In conjunction with these lexical features, details of the WHOIS properties of the domain, such as the age of the domain and the name of the registrant were also used as features. Furthermore, properties of the IP address were also considered, such as the Autonomous System that the address belongs to, and whether or not the IP addresses given in the DNS records are prefixes of one another. Geolocation services were also used to obtain the continent, country and city where the URL is being hosted, and these too were used as features. A request to the page was also made to measure the connection speed, as it was postulated that many phishing sites were likely being hosted on compromised machines on residential networks. A batch learning process was used to train an SVM to obtain a false positive rate of 2.09%, and a false negative rate of 2.55%.

Despite the expansion of the lexical feature set, the method proposed by Ma et al. (2011) still requires the use of a number of host based features. In order to obtain these host based features, a number of requests need to be made. DNS requests are made to establish the IP address of the website, as well as the details of the AS and MX records associated with the domain. WHOIS lookups must be performed on both the IP address and the domain name to identify the registrant, registrar and the hosting organisation, and a further request must be made to determine the connection speed of the host machine. Although these requests provide a greater amount of information to work with, the effect of network latency on the classification process originally noted in the work of Zhang, Hong and Cranor (2007) is not addressed. By moving to a more URL-based approach, the attacker now has less of an opportunity to interfere with the detection process, as the page content itself is no longer considered. This does not make the system immune to tampering, however, as the WHOIS and DNS records for the domain could still be under the control of the attacker.

Positional Encoding

It has been shown that by encoding positional data into an n-gram bag of words model, classification accuracy can be improved (Baykan et al., 2011). Khonji, Iraqi and Jones (2011) proposed an alternative method of including distance metrics using a token based approach. In this approach, URLs are broken into tokens in the same fashion as described in the work of (Baykan et al., 2009). Initial investigation into the frequencies of the tokens showed that benign URLs often contained a higher number of previously seen tokens, and that there was only a 12% overlap in tokens between phishing and benign URLs (Khonji, Iraqi and Jones, 2011). To further reduce this overlap, the distance from the TLD is also considered for each token. Using a supervised, batch based learning method using a statistical classifier, the "phishiness" of each token is calculated and then compared against a threshold (Khonji, Iraqi and Jones, 2011). Using this approach, an accuracy of 97.31% was achieved over a data set of 73,733 URLs (Khonji, Iraqi and Jones, 2011).

Online Learning - The State of The Art

One particular disadvantage of the method proposed by Khonji, Iraqi and Jones (2011) is that the training of the classifier must occur in a batch based fashion. It has been shown that the average phishing attack lasts, on average, for only 58 hours, and as such it is important the URL based classification systems respond quickly to the ever-changing nature

of phishing sites (Moore and Clayton, 2007; Blum et al., 2010). Batch based learning methods are inherently less responsive to a changing environment, as the model must be re-trained frequently to account for these changes (Whittaker, Ryner and Nazif, 2010; Ma et al., 2011). In response to this, Blum et al. (2010) proposed an extension to the token based feature set described by Baykan et al. (2009); using an online classification mechanism to continuously train on incoming data samples. Much like the work of Baykan et al. (2011) and Khonji, Iraqi and Jones (2011), the position of each token was taken into consideration. Rather than explicitly encoding the relative position of each token, position sensitive tokens were recorded for certain levels. This differs from the work of Khonji, Iraqi and Jones (2011) in that there is a specific dimension for tokens at specific points in the URL, such as the second token in the domain. This has the effect of increasing the dimensionality of the model considerably (Blum et al., 2010). Using a confidence-weighted algorithm, continuous training over a stream of 34,234 URLs gave an average accuracy of 97%, with a 3% error rate (Blum et al., 2010). The use of an online learning classifier has the advantage of eliminating "the delay, security risks, and overhead associated with examining content" (Blum et al., 2010). It was noted, however, that the feature set used was fairly basic, and that the false positive rate could potentially be reduced through the investigation into a more diverse lexical feature set, and training on real world data.

One limitation of online learning classifiers that was identified in the work of Blum et al. (2010) is that the updating of the model requires a reference label for each new training instance. Zhao and Hoi (2013) took the work of Blum et al. (2010) one step further, and attempted to address this issue by applying active learning to the classification method. Through using active learning, the goal is to automatically select the URLs which would be most useful for training. The confidence weighted algorithm proposed in Blum et al. (2010) was augmented to include an active component, over a set of 1 million URLs. This algorithm was then compared against a number of state of the art classification methods, namely Perceptron, Passive-Agressive, Confidence-Weighted, PAUM, CPA, LEPE and CSRND (Zhao and Hoi, 2013). A cumulative accuracy of 97.146% was achieved using this method, using only 0.5% of the 1 million URLs for training, greatly reducing the overall cost of training (Zhao and Hoi, 2013).

2.2 Contribution

A wide variety of simple lexical features have been proposed, the most popular of which include the presence of an IP address in place of a hostname (Garera et al., 2007; Zhang, Hong and Cranor, 2007; Aburrous et al., 2010; Xiang et al., 2011; Sunil and Sardana, 2012; Huang, Qian and Wang, 2012; Kausar et al., 2014; V and A, 2014; Whittaker, Ryner and Nazif, 2010; R et al., 2014), the length of the URL, hostname or path (Huang, Qian and Wang, 2012; James, L. and Thomas, 2013; R et al., 2014; Nguyen and Nguyen, 2016; Ma et al., 2011; Le, Markopoulou and Faloutsos, 2011), the number of specific characters such as dots, or slashes (Zhang, Hong and Cranor, 2007; Xiang et al., 2011; Huang, Qian and Wang, 2012; Kausar et al., 2014; Ma et al., 2011; Le, Markopoulou and Faloutsos, 2011), and the presence of brand names or suspicious terms (Garera et al., 2007; Zhang, Hong and Cranor, 2007; Xiang et al., 2011; Huang, Qian and Wang, 2012; James, L. and Thomas, 2013).

Le, Markopoulou and Faloutsos (2011) have shown that these simple lexical features can have a positive impact on the accuracy of classification, however, these features have only ever been used in conjunction with host based features (Ma et al., 2011; Garera et al., 2007), or a more complex bag of words model (Blum et al., 2010) due to high false positive rates. To this end, multiple authors have suggested that further investigation into expanding the set of effective simple features is needed (Ma et al., 2011; Blum et al., 2010; Le, Markopoulou and Faloutsos, 2011; Khonji, Iraqi and Jones, 2011).

Various sparse bag-of-words models using n-gram segments have also been considered (Blum et al., 2010; Khonji, Iraqi and Jones, 2011; Baykan et al., 2011). These approaches have successfully demonstrated that it is possible to base the feature set purely on the URL and achieve an reasonable accuracy, however nearly all of them failed to use a large sample size for their experimentation, leaving questions as to whether the results would generalise to a real world setting. Baykan et al. (2011) performed their experimentation over a larger set of two million URLs, with a maximum accuracy of 82%. This suggests that further work is still needed to improve the bag of words approach using a large, real world data set.

This dissertation takes inspiration from the future work outlined by Blum et al. (2010) to investigate the effectiveness of the simple lexical features already identified in existing literature, and to then expand this set through the suggestion of further novel features. This will then be combined with the work of Blum et al. (2010), Khonji, Iraqi and Jones (2011) and Baykan et al. (2011) to evaluate how the use of simple lexical features compares to a bag of words approach, and whether we can achieve better results by combining them. Concerns over the accuracy of the results from the work of Blum et al. (2010) and Khonji, Iraqi and Jones (2011) will be addressed in this dissertation by performing our experimentation over a large, real world data set. This data set comprises of over 7.5 million URLs provided by Netcraft, spanning 6 months worth of data over the course of 2016. We will also expand upon the most recent developments in the area of online phishing URL classification by implementing and testing two online learning approaches, neither of which have been applied in this area before.

2.3 Summary

In this chapter, we have discussed the existing literature surrounding the area of phishing site classification. Four different sources of features for classifying sites were identified: Lexical features from the URL, host-based features from sources such as DNS requests, third party heuristics including Google PageRank and content-based features from the source of the web page itself. We examined the early work in this area, identifying four different categories of phishing URL from the work of Garera et al. (2007). It was shown that these four categories heavily influenced the work that followed, with many authors choosing to select features from a combination of the four identified sources in an attempt to account for these different types of phishing URL.

The use of third party heuristics was criticised for being too reliant on the actions of outside organisations, a point which was highlighted in the work of Nguyen et al. (2014). Their proposed methodology focussed on the use of the Microsoft Reputation Service to generate categories based upon the content of the site. However, despite being published fairly recently in 2014, the Microsoft Reputation Service no longer exists—rendering much of the work that was done obsolete.

Following this, we also criticised the use of host based and content based features for introducing latency into the classification process. Obtaining the source data from which to draw such features requires sending and receiving network requests, which limits the rate of processing (Zhang, Hong and Cranor, 2007)—potentially reducing the responsiveness of the system. This is a problem in situations where the results of the classifications are used to protect internet users, as longer delays mean a greater number of unprotected users.

We then looked at how the work of Ma et al. (2011) attempted to address some of these concerns by increasing the number of lexical features being considered. They achieved this by using a bag of words approach inspired by the work of Baykan et al. (2009) on URL topic classification. In their work, Ma et al. (2011) split URLs into "token" segments which then formed the dictionary for a sparse binary feature model. Although these features were still combined with various host-based features, it was shown that this work formed an important first step towards building a purely lexical feature set.

Further studies into the use of purely lexical feature sets for phishing URL classification were then discussed. The possibility of positionally encoding tokens within the bag of words model was also discussed, though it was shown that previous attempts had little success (Khonji, Iraqi and Jones, 2011). These approaches successfully demonstrated that it could be possible to base the feature set purely on the URL, achieving an reasonable accuracy in most cases. Unfortunately, it was also shown that many of them failed to use large enough sample sizes for their experimentation, leaving questions as to whether their results would generalise to a real world setting.

Next, we reviewed the most recent developments in the area of lexical phishing URL classification, most of which focusses on the application of online learning in order to improve the rate of classification. Following this, the contribution of this dissertation was outlined. We showed how further work was needed to examine and extend the set of simple lexical features laid out in previous studies. It was also shown how our work would expand upon the work of Blum et al. (2010), Khonji, Iraqi and Jones (2011) and Baykan et al. (2011), and how our study would address a previous lack of data through the use of a large, real-world dataset obtained from Netcraft. Finally, we demonstrated that this dissertation would explore new

areas within the state of the art by implementing and testing two online classifiers that have yet to be applied in this setting.

In the next chapter, we will consider lexical features in more detail. We shall examine a number of different methodologies, before proposing novel feature sets for empirical evaluation.

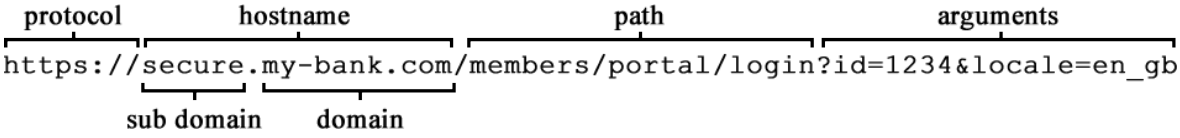
Chapter 3

Lexical Features

Contents

3.1 Making Observations From The URL

The URL of a web-hosted resource, such as a website, contains a great deal of useful information. We can often infer a number of things about the resource—as well as the configuration of the provider—simply by examining the URL. For example, consider the following fictional URL:



The *protocol* being used to access the resource is `https://`, which is the encrypted version of the HTTP protocol. This tells us two things, firstly, it tells us that we will be retrieving the resource from an HTTP server. Depending on where the URL was discovered, the presence of the HTTPS protocol may indicate that the resource will be a web page, as HTTP is the primary method for the transfer of Hyper-Text Mark-up Language—the mark-up language supporting the majority of websites in existence today. Note however, that it does not tell us for certain that the resource will be a web page, as it is possible to serve other file types over HTTP. The second thing to note from the protocol is that the connection is encrypted. This suggests that the resource in question is involved in the manipulation of sensitive information, and that the owner has taken the time to set up extra precautions. You might assume that this would therefore indicate that the site is popular and most likely legitimate, however, it is becoming more common for sites that do not handle sensitive information to adopt SSL, so—again—this observation should be taken with a pinch of salt.

In a similar sense to the protocol, the *hostname* of the URL can give us an indication of the legitimacy of the site, amongst other things. Most importantly, the *domain* allows us to identify the owner of the resource. If we know that `my-bank.com` is owned by a legitimate organisation, then it is likely that a resource located on `secure.my-bank.com` would also belong to the same organisation. It should be noted that this does not always hold true, as methods such as pharming exist to redirect the user to a different domain—but it is nonetheless a useful observation in the context of URL classification.

Furthermore, the hostname also allows us to identify attempts to include the domain of a target organisation within a sub-domain, or to use a domain with a similar appearance to that of the target organisation. For example, if it is known that `example.com` does not belong to the owner of `my-bank.com`, then a resource located on `my-bank.example.com` is likely to be suspicious. Similarly, the domain `my-b4nk.com` would also indicate that the resource is unlikely to be legitimate. The use of `secure` in the hostname of the example above might also indicate that the resource we are accessing is part of a sensitive operation, and that other subdomains of `my-bank.com` may not be using SSL. On the other hand, it could also be symptomatic of an attacker's attempt to convince a victim that the resource is safe and "secure", when in fact this may not be the case.

The *path* of the URL helps to give us more context as to the function of the resource, as well as the layout of the providing file system. Here, the use of `portal` and `login` seem to confirm our earlier suspicions that the resource is indeed involved in the manipulation of sensitive data. The location of such terms within the path can also give us an indication as to whether the URL is legitimate. If suspicious terms such as `login` are present towards the

beginning of the path, then it is likely that the URL is legitimate—or that the domain has been registered for fraud. Suspicious terms in locations where it would not normally make sense to put them, such as under `/images/`, `/js/` or `/~oy213/` may be indicative of a benign web server that has been compromised by an attacker. The path of the URL may also give us details as to the technologies being used by the provider. For example, a path ending in a `.php` extension would indicate that the web content being accessed has been produced using the PHP language, whereas a path containing `wp-content` may suggest that the owner is using the content management system, Wordpress. This presents an interesting extra piece of information, and could be useful to us as certain technologies may be more popular with fraudsters, or could be indicative of vulnerable systems commonly exploited by fraudsters.

The *arguments* of the URL are also helpful in determining the function of the request being made. In the case above, we can see that an `id`, along with a `locale`, are passed to the web server when the URL is requested. Arguments such as these are more likely to be seen when handling sensitive user data during processes such as logging in or signing up, whereas other arguments—like `article`—might be more common for sites that do not handle sensitive data. These arguments therefore have the potential to allow us to identify URLs that are likely to be involved in the handling of credentials, as well as to filter out URLs that are likely to be benign.

3.2 Recognising Phishing URLs

Following the discussion from section 3.1, it is not be unreasonable to suggest that an expert user should be able to discern a phishing URL from a legitimate URL without visiting the site itself. For example, consider the following real URLs:

1. `https://www.paypal.com/signin?country.x=GB&locale.x=en_GB` (legitimate)
2. `https://www.pansarijewels.com/PayPal-Update/6bce85405/` (phishing)
3. `http://www.paypal.com.intelligence.is/signin/` (phishing)

The first URL is clearly legitimate, as it has a domain of `paypal.com`, which we know belongs to the real PayPal Inc. Despite the use of HTTPS, the second URL is much more suspicious. The use of the target name `PayPal` in the path is particularly indicative, especially when the domain (`pansarijewels.com`) is completely unrelated. The third URL is more difficult to discern than the second, using a similar path to the legitimate URL and using `www.paypal.com` in the hostname. The only real clue we have here is the unrelated domain name, which shows this URL to be fraudulent also. This demonstrates that it is possible to decide upon the relative likelihood of each of the three URLs being a phishing site, using only the appearance of the URLs themselves.

3.2.1 Lexical Features: A Definition

Lexical features are items of data selected from the URL that allow us to capture this observable difference between the appearance of a legitimate URL and that of a phishing URL. A wide variety of lexical features have been suggested in previous work, and in this dissertation we group these into two separate categories: simple lexical features and situated lexical features.

We describe *simple* lexical features as statistics which may be obtained from the URL without the maintenance of an internal model. Examples of simple lexical features identified in previous work include the number of characters in the hostname, or the total number of dots in the URL (Ma et al., 2011). We discuss simple lexical features in greater detail within chapter 4.

Situated lexical features maintain a context outside of the URL, that is, they are "situated" within some environment. Maintaining this context requires the use of an internal model to represent some portion of the state of the world within which the URL exists. As the world is constantly changing, the model must also be updated frequently so as to remain accurate. For example, recognising the presence of a target organisation name within the URL requires a list of all of the potential targets that are likely to appear (Garera et al., 2007). It is unlikely that this list of targets will remain the same throughout the operational life of a classifier, so a process for updating the target list must also be designed and implemented. This often makes situated lexical features more computationally expensive to maintain than simple lexical features, however—as we shall discuss in chapters 4 and 5—their complex nature allows us to capture a number of details from the URL that cannot be obtained using simple lexical features.

3.3 Characteristics of Phishing URLs

A wide variety of lexical features can be found in the literature, and in order to analyse them it is useful to first reconsider the different types of phishing URL that are identified in the work of Garera et al. (2007).

1. **Obfuscation of the hostname with an IP address**. These forms of attack use the IP address of the fraudulent site in place of the domain name, often including the name of the target organisation in the path instead.

2. **Obfuscation of the hostname with a legitimate domain**. Here, a valid looking domain name is used, usually with the name of the target organisation in the path. This is usually done in order to imitate a URL that would contain a redirect.

3. **Obfuscation with large hostnames**. These attacks will include the name of the target organisation, or a related phrase, in the hostname, with a large string of subdomains appended in order to push the actual domain off of the end of the URL bar in the victim's browser.

4. **Domain name unknown or misspelled**. Here, a close misspelling of the target domain has been registered in order to make the URL appear legitimate at first glance. Unknown domains, or URLs which do not fit into any of the above categories, are also included in this final category.

These categories represent an important first step towards understanding the visual characteristics of phishing URLs, but leave much to be desired.

Since the publication of the work of Garera et al. (2007), fraudsters have had to adapt their techniques in response to advances in prevention methods. This has lead to the rise of a number of practices that were likely never anticipated in 2007. For example, the rise

in popularity of website builders—such as Squarespace and Wix—has increased the market competition, resulting in many website builders offering incentives to potential customers. These incentives often take the form of free trial periods or offers for free website hosting. Such incentives can be exploited by fraudsters, allowing them to host their phishing sites for free during trial periods. The categories suggested by Garera et al. (2007) fail to accommodate many of these new practices.

It is also important to note that phishing URLs may exhibit certain visual characteristics regardless of "category". For example, the attacker could include the email address of the victim in the arguments of the URL in order to populate the username field with said email address. To the unsuspecting user, the automatically populated field could be taken as an indication that the site is legitimate. Such an approach could be taken regardless of whether the hostname was an IP address, or a long string of random characters. The use of "categories" is, then, perhaps a little heavy-handed. Instead of identifying a number of discrete possible forms that a phishing URL may take, we instead propose a number of visual "characteristics", where one or more of these characteristics may be exhibited within a single URL. This helps to focus our discussion of simple lexical features, as we define a number of visual clues that we wish to identify.

Our proposed set of phishing URL characteristics is given below, along with a number of real world examples:

1. **Hostnames with an IP address**. These forms of attack use the IP address of the fraudulent site in place of the domain name, often including the name of the target organisation in the path instead.

   `http://192.185.73.38/~novel/service.admin.paypal.info/webapps/`

2. **Generic domains for multiple targets**. Here, the fraudster will register a suspicious domain related to the phishing process, rather than a particular target. Such a domain may include terms such as "account", "verification" or "update", allowing the fraudster to set up multiple subdomains targeting different specific organisations.

   `https://www.paypal.verification-infoupdate.com/`

3. **Long hostnames**. These attacks will include the name of the target organisation, or a related phrase, in the hostname, with a large string of subdomains appended in order to push the actual domain off of the end of the URL bar in the victim's browser.

   `http://my.hypovereinsbank.de.secure.login.77d2{...}fb9a.proluxcleaning.uk/`

4. **Compromised sites**. This is where the fraudster exploits some security vulnerability in a previously benign site in order to upload a phishing attack onto the host. This often results in the URL containing an unlikely path root, or a user directory, as the attacker is usually limited to uploading the phishing content into a directory owned by the vulnerable service.

   `http://store.webkittechnology.com/system/logs/sign/up/`

5. **Subdomains provided through free site builders**. Here, the fraudster will use a free website builder, such as Squarespace or myfreewebsite, to host their attack. These website builders will often allow customers to choose their own subdomain, which fraudsters can exploit in order to produce a more believable URL.

   `http://ufrgsbrr.my-free.website/`

6. **Shortened URLs**. Here a URL shortening service, such as tinyurl or bitly, is used to hide the suspicious nature of a phishing URL. These services generate a unique shortened URL which will redirect visitors to a URL of the fradster's choice. Users may be more inclined to follow a shortened link, as they do not appear as visually suspicious as many longer phishing URLs.

   `http://bit.ly/2pwm4n2`

7. **Deceptive domains**. These URLs are the result of the fraudster registering a close misspelling of the target domain in order to deceive visitors.

   `http://ubssecure.com/ubs/uk/en.html`

8. **Deceptive file paths**. Here the attacker manipulates the file path to appear more convincing. This is achieved by either mimicking the directory structure of the legitimate site, fabricating a seemingly relevant file path, or including the hostname of the legitimate organisation within the file path. Users who are not familiar with the structure of a URL may see this hostname and assume that the site is legitimate.

   `http://www.inmueble.web.ve/~tequetap/cgi-bin/online/banking/id/verification/`
   `http://www.csfparts1-clientsex.com/--/www.impots.gouv.fr/file/`

9. **Deceptive arguments**. Attackers will often alter the arguments of the URL to either mimic the appearance of legitimate URLs, or to add context to URLs that would otherwise appear unrelated.

   `http://ow.ly/4mIpzC?Facebook-acount-disable`

10. **Randomised directory names**. In an attempt to evade being blocked by blacklist-based systems, attackers will sometimes generate random directory names for each visitor. This is usually done by calculating the md5 hash of a randomly generated number, as seen below.

    `http://verificationzone.shannonnoel.com/update-paypal/1a0b5ee9e8371895674b810a0b/`

11. **Personalised URLs**. Here the attacker includes the email address of the victim within the URL arguments. This is usually indicative of an attempt to personalise the phishing page in some way, for example, by automatically populating a username field.

    `http://www.hotrodsdecals.com/wp-content/mailbox/post.php?email=victim@example.com`

12. **Port number manipulation**. Similarly to the randomised directory names, fraudsters may also provide the attack over a non-standard range of ports, specifying a random port within this range in the URL. This is done not only to avoid detection by blacklist-based systems, but to also fool content-based systems fetching the URL over the default port.

    `http://zeotsw.gq:1996/indexx.asp`

3.4 Summary

In this chapter, we have considered lexical features in greater detail, providing definitions along with a set of novel phishing URL characteristics. The various components of a URL were identified, and it was shown how each component could be used to make inferences regarding the content of the resource associated with the URL. Following this, we demonstrated that by considering the appearance of a URL it was possible to discern between a phishing URL and a legitimate URL. Lexical features were then defined as items of data drawn from the URL that allow us to capture elements of this difference in appearance between phishing and legitimate URLs. This concept of a lexical feature was then decomposed further into two categories: simple lexical features and situated lexical features. Simple lexical features were defined as statistics which may be obtained for the URL without requiring an internal model. Conversely, situated lexical features were identified as those which maintain a context outside of the URL itself. We then went on to consider a number of categories of phishing URL that had been identified in the work of Garera et al. (2007). These categories were criticised for being outdated and restrictive, leading us to propose a new set of twelve visual "characteristics" for recognising phishing URLs.

In chapter 4, these new categories will be applied to the analysis of the simple lexical features found in previous work. In doing so, we will provide a critique of the existing work, as well as of simple lexical features as a whole. This critique will inform the selection of a novel simple lexical feature set with the aim of improving classification accuracy.

In chapter 5, situated lexical features will be explored in further detail. The advantages of using situated features over simple lexical features will be discussed, as well as the issues that arise when using such features in an online setting at a large scale. Following this discussion, a number of solutions are proposed and our implemented solution is detailed.

Chapter 4

Simple Lexical Features

Contents

4.1 Overview

In section 3.2.1, we defined simple lexical features as statistics which may be obtained from the URL without the maintenance of an internal model. These features should be capable of capturing some observable difference between a legitimate URL and a phishing URL. In section 3.3, we identified a number of possible characteristics for identifying phishing URLs in order to give us an indication as to some of the differences that we may wish to capture.

For example, the hostname of a URL containing an IPv4 address will likely always have the same number of periods, as well as containing only numeric characters between the periods. In order to capture the difference between an alphanumeric domain name and an IP address, we could measure the number of periods, the proportion of numeric to non-numeric characters, or make use of a regular expression.

From this simple example it is already apparent that there are a number of options available to us. These options are usually the results of conjecture, and as such they cannot be guaranteed effective without empirical evidence. This forms the root of our uncertainty when it comes to selecting lexical features. Previous work has suggested a wide variety of simple features, but often little time is spent on understanding their effectiveness on an individual scale, or how they interact once combined.

In section 4.2, we identify a set of simple features drawn from existing research. Following this, in section 4.3, we use the existing set of simple features to inform the suggestion of novel additions to the simple lexical feature set.

4.2 Existing Simple Features

Table A.1 lists all of the simple lexical features that we have identified within previous research. Twenty five features are identified in total, the most common of which are the presence of an IP address within the hostname, the total length of the URL, and the number of '.' characters present in the hostname.

We consider each feature with respect to our set of phishing URL characteristics (defined in section 3.3), to see how the existing set of features covered said characteristics. This is done by consulting the study in which the feature was first identified and noting their justification for the use of said feature. For example, Whittaker, Ryner and Nazif (2010) use the number of '.' characters in the hostname as a feature for recognising URLs with long hostnames. As a result, URLs with a type 3 characteristic are marked as identified by the use of this feature. Table A.2 shows the coverage of characteristics that we identified for each simple lexical feature.

The information from table A.2 is visualised in figure 4.1. From this, we can see that most of the characteristics are targeted by at least one existing simple feature, however, some have fewer applicable features than others. URLs with deceptive file paths (type 8) have the largest number of applicable features, whilst URLs containing deceptive domains (type 7) and subdomains provided by website builders (type 5) are not covered by the existing features at all. This highlights an important limitation of simple lexical features. In order to recognise deceptive domains, or subdomains provided by website builders, some form of information regarding the world outside of the URL is required. In the case of deceptive domains, for example, a list of potential target domains would be required in order to be

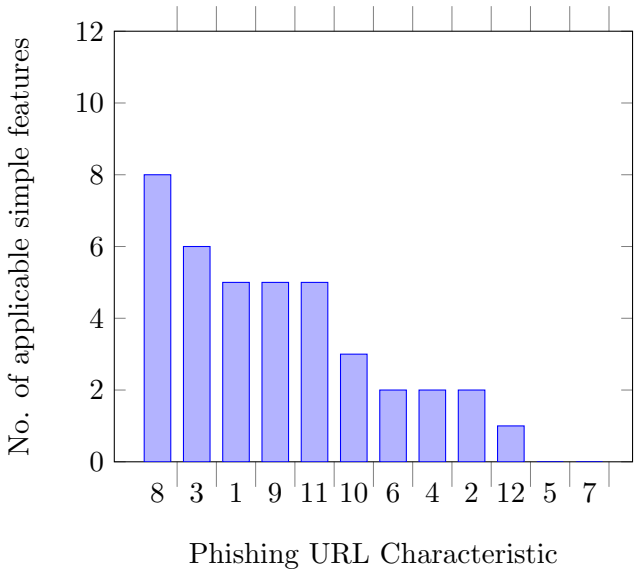


Figure 4.1: A histogram showing the coverage of phishing URL characteristics by the simple lexical features identified from existing literature

able judge whether or not a particular domain is deceptive. This further information forms a model that exists beyond the context of the URL, which is not something that a simple lexical feature can provide. In order to recognise all of the characteristics that we have identified, it is likely that we will need a more complex set of features that is capable of capturing the use of specific language within the URL.

Figure 4.2 shows the separation between phishing URLs and benign URLs produced with the set of existing simple lexical features. This data spread was produced by selecting a random sample of 1000 phishing URLs and 1000 benign URLs from our April 2016 training data (for more details on this data, see 7.1), normalising each feature to a value between 0 and 1 and performing principal component analysis to project the data into two dimensions.

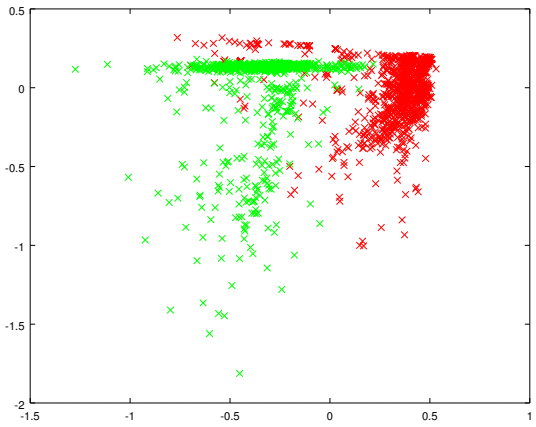


Figure 4.2: A plot of the data spread between phishing and non-phishing using the simple lexical features from existing literature. Green indicates benign sites, and red indicates phishing sites.

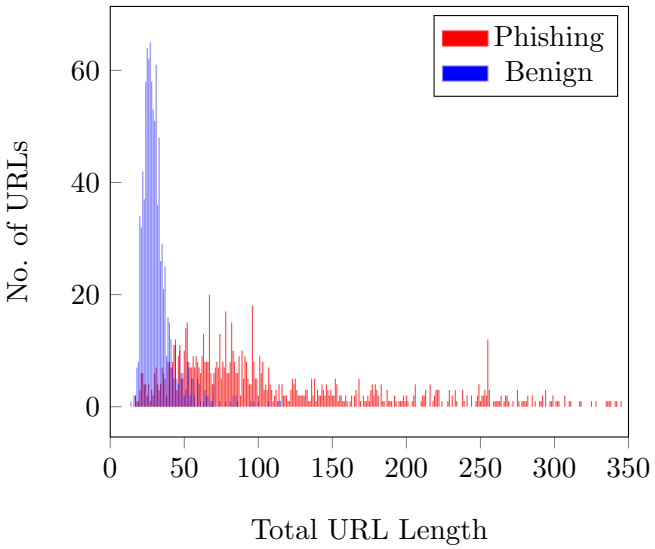


Figure 4.3: A histogram showing the distribution of URL length within a sample of 1000 phishing URLs and 1000 benign URLs.

From figure 4.2 we can see that the identified features do produce some separation between the two classes of URL, however, there is still an observable amount of overlap. This overlap indicates that the existing feature set is not discriminative enough to fully distinguish between phishing URLs and benign URLs.

In order to understand why this overlap exists, we take a closer look at three popular simple lexical features from existing literature: total URL length, the number of dots in the hostname and the number of slashes in the path. The frequencies of each value for every one of these three features are taken across a random sample of 1000 phishing URLs and 1000 benign URLs from the April 2016 training data. In doing so, we are able to visualise the distribution of values for each feature between the two classes of URL.

The results for total URL length are shown in figure 4.3. The values appear to be normally distributed for both classes of URL, and we can see that the spread of phishing URLs is wider than that of the benign URLs in this sample. This indicates that phishing URLs are more likely to be longer than benign URLs. This observation is expected, as many of the characteristics defined in section 3.3—such as types 2, 3 and 8—involve adding terms into the URL in order to make them more deceptive.

Figure 4.4 shows the number of dots within the hostname for the two classes of URL within the sample. Similarly to URL length, there appears to be a difference between the two classes, however it is less distinct. The spread of data for benign URLs is tightly centred around two, whereas the data for phishing URLs is spread more widely. We can see that within this sample, phishing URLs are more likely to have a single period within the hostname than benign URLs. Interestingly, a URL is more likely to be a phishing when the number of dots in the hostname is greater than five.

The number of slashes within the path for the two classes of URLs are shown in figure 4.5. Here again we can see a difference in the distribution of values between each class of URL. The majority of benign URLs are centred around a single slash, whilst the phishing URLs are centred around three slashes. Again, the spread of the data for phishing URLs is greater

than that of the benign URLs.

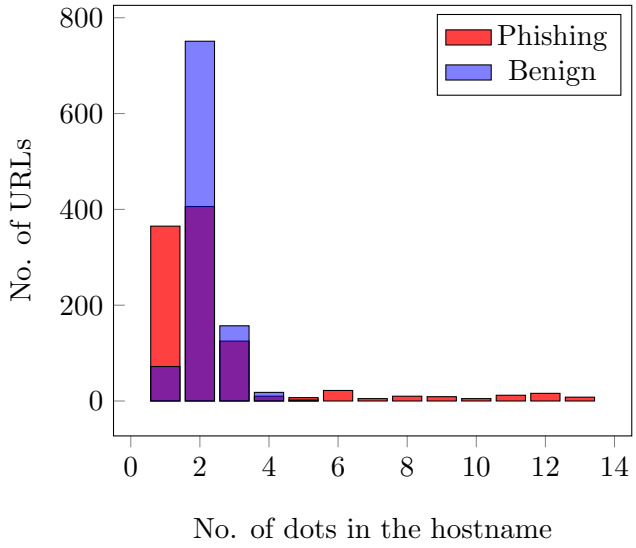


Figure 4.4: A histogram showing the distribution of dots within the hostnames of a sample of 1000 phishing URLs and 1000 benign URLs.

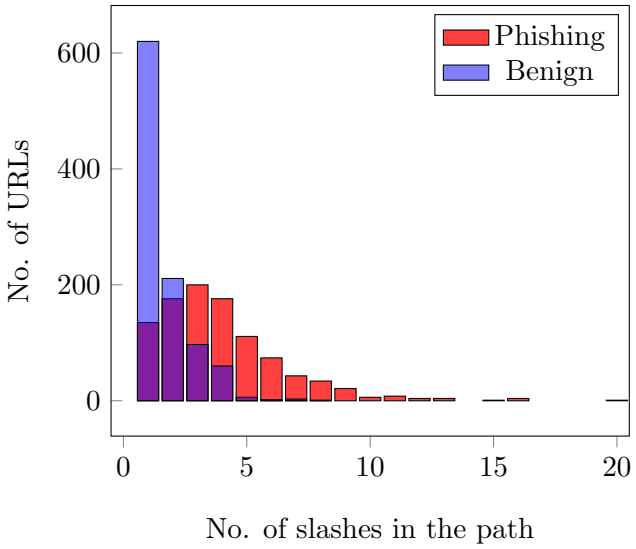


Figure 4.5: A histogram showing the distribution of slashes within the paths of a sample of 1000 phishing URLs and 1000 benign URLs.

For each of these three simple features, it has been demonstrated that there is an observable difference between the distributions of values. This shows that it is be possible to use these features to discern between a phishing URL and a benign URL, however the overlap in the data seen in figure 4.2 suggests that the existing feature set still has room for improvement. Given the uneven spread of features shown in figure 4.1, we believe that there is an opportunity to improve the existing feature set by designing further features to target the URL characteristics that are not well covered.

4.3 Feature Set Expansion

In section 4.2, twenty five simple lexical features from existing literature were identified. It was shown that these features could be used to discern a phishing URL from a benign URL, but that they were not sufficiently discriminative to be effective in every case. We explored the relationships between these features and the various characteristics of phishing URLs that we defined in section 3.3. From this, we were able to see that a number of the characteristics had fewer applicable features than others (see figure 4.1). We proposed that through identifying more features to capture the characteristics that are currently less well covered, the discriminative power of the feature set could be improved. To this end, we will consider each characteristic in reverse order of coverage with the aim of identifying novel features that can be used to expand upon the existing feature set.

4.3.1 Deceptive Domains and Subdomains From Free Site Builders

Our analysis showed that the existing simple feature set does not contain any features designed to identify URLs with deceptive domains, or URLs using subdomains provided by free site builders. As discussed in section 4.2, this is due to a fundamental limitation of simple lexical features. In order to decide whether or not a domain is deceptive, we must first identify the legitimate domain that is being impersonated. This requires the use of information beyond that which exists purely within the URL, and as such cannot be achieved using a simple lexical feature. Similarly to deceptive domains, identifying URLs using subdomains that have been provided by free site builders cannot be achieved without knowing the domain names owned by the free site builders. As a result, there is little that we can do to directly identify these characteristics using simple lexical features. In chapter 5 we consider the use of situated lexical features as a potential solution to this issue.

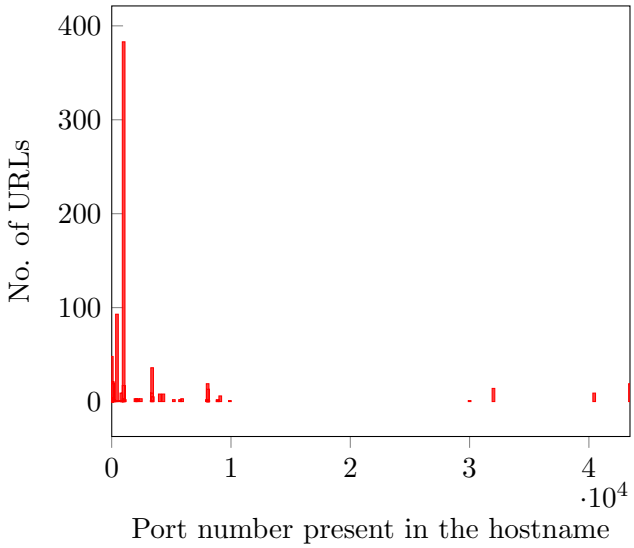


Figure 4.6: A histogram showing the distribution of port numbers within the hostnames of a sample of phishing URLs from April 2016.

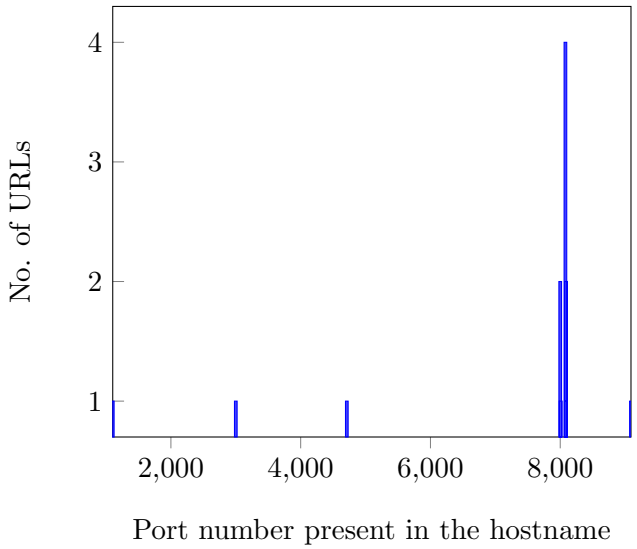


Figure 4.7: A histogram showing the distribution of port numbers within the hostnames of a sample of benign URLs from April 2016.

4.3.2 Port Number Manipulation

Figure 4.1 shows that only a single simple feature for identifying URLs containing a port number was found in the literature. This feature was identified in the work of Le, Markopoulou and Faloutsos (2011), where a regular expression was used to extract the port number from the URL. The value of this feature was defaulted to zero in the case where the URL did not contain a port number. One possible alternative to identifying the port number is to measure the number of ':' characters within the hostname. This has the advantage of being simpler to compute than isolating the port number, however it gives us less information regarding the port number itself.

For example, in our training data from April 2016, only 15 of the 325,837 benign URLs contained one or more colon characters within the hostname, compared to 805 of the 325,837 phishing URLs. This suggests that simply extracting the number of colons from the hostname would be sufficient to identify likely phishing URLs, but the fact that a number of benign URLs also contain a port number means that we could not be certain of our decision. Figures 4.6 and 4.7 show the distributions of port numbers within the samples of phishing and benign URLs respectively. We can see that the majority of port numbers found in benign URLs are centred around port 8000, probably due to the use of 8080 as an alternative to the default HTTP port, 80. For phishing URLs, however, the spread of the port numbers seen is much broader, centred around 1003, but ranging as high as 43443. By identifying the port number, we can model differences in the distributions between classes, giving us a greater amount of information upon which to base our classification. As a result, it is unlikely that the number of colon characters in the hostname could provide any new information, but it may prove useful as a faster alternative to the extraction of the port number.

4.3.3 Shortened URLs

Similarly to URLs containing deceptive domains or subdomains from free site builders, there is little that can be done to identify shortened URLs using simple lexical features alone. In order to be certain that a URL has been shortened, we would require a list of the domains of all the URL shortening services. This information goes beyond the context of the URL itself, and as such cannot be expressed with a simple lexical feature. The only simple lexical feature that could be useful for identifying possible shortened URLs is the total length of the URL itself, as first found in the work of Aburrous et al. (2010).

4.3.4 Randomised Directory Names

We identified three simple features from the existing literature for identifying URLs with randomised directory names. Those features are the presence of hexadecimal characters within the URL (Aburrous et al., 2010), the length of the longest subdirectory (Le, Markopoulou and Faloutsos, 2011) and the total length of the path (Le, Markopoulou and Faloutsos, 2011).

The presence of hexadecimal characters within the URL does not necessarily imply the presence of hexadecimal characters within the path, however. Therefore, we propose that this feature be made more specific, to identify the presence of hexadecimal characters only within the path. Regular expressions could also be leveraged to identify hexadecimal strings of particular lengths, as well as subdirectories formed entirely of such strings.

It is also possible that not all random directories are formed from a hexadecimal string. To accommodate for this, we identify three new simple features. The proportion of the number of digits to letters in the path of the URL would allow for the identification of randomisation techniques that make use of numerical strings for directory names. The total number of digits in the path may also prove useful for identifying long strings of numbers within the path. One issue with simply counting the number of digits in the path is that long paths are more likely to contain a greater number of digits. In order to account for this, the maximum number of digits in any one subdirectory could also be measured.

4.3.5 Generic Domains For Multiple Targets

The existing simple feature set contains three features targeted towards identifying generic phishing domains: the numbers of '-' and '@' characters in the hostname (Zhang, Hong and Cranor, 2007), as well as the number of tokens in the hostname (Le, Markopoulou and Faloutsos, 2011).

Because generic domains are re-used for multiple different targets, they often take the form of multiple words—such as "verification" or "account"—separated by delimiters, rather than containing the name of a company. The presence of '-' and '@' characters can therefore be indicative as these characters are often used to separate the words within the domain. The subdomains are then used to target a specific organisation, and as such, are usually shorter than the domain itself. By measuring the proportion of the length of the domain to the length of the first sub-domain, we expect to be able to capture this difference in length.

4.3.6 Compromised Sites

We identified four existing simple features for recognising compromised sites: the number of '/' characters and tokens in the path (James, L. and Thomas, 2013), as well as the total length of the path and the number of sub-directories in the path (Le, Markopoulou and Faloutsos, 2011).

The phishing content on a compromised site is often buried within the existing file system, leading to longer file paths consisting of short sub-directories. The existing features help to capture the increased length of the path, as well as it's dense nature. The "tokens" mentioned in the work of (James, L. and Thomas, 2013) are only mentioned in passing, however, with little detail given as to their actual nature. In the interests of precision, we propose that this feature be replaced by counting the number of special characters in the path, namely ' ', '-' and '@'. Similarly to how the work of Le, Markopoulou and Faloutsos (2011) treats URL arguments, it may also be beneficial to measure the maximum number of special characters in any one sub-directory, as it is less dependent on the total length of the path.

By measuring the proportion of the path length to the total URL length, we hope to be able to separate long URLs with short hostnames from long URLs with long hostnames. This is useful as compromised sites are more likely to have a short hostname, due to the attacker usually only having control over the file path. Along with this, we also consider the average sub-directory length, the length of the shortest sub-directory and the length of the longest sub-directory

4.3.7 Personalised URLs

The existing literature provides five simple lexical features for identifying personalised URLs: The presence of hexadecimal characters in the URL (Aburrous et al., 2010), the number of arguments (Le, Markopoulou and Faloutsos, 2011), the total length of the argument string (Le, Markopoulou and Faloutsos, 2011), the length of the longest argument (Le, Markopoulou and Faloutsos, 2011) and the presence of an '@' character in the URL (Aburrous et al., 2010).

We make these features more specific to personalised URLs by measuring the presence of hexadecimal and '@' characters within the arguments, rather than the full URL.

4.3.8 Deceptive Arguments

The identification of deceptive arguments has been covered previously in the work of Le, Markopoulou and Faloutsos (2011). They suggested a number of simple features including the number of arguments, the length of the longest argument and the maximum number of special characters in a particular argument.

In this dissertation, we specify these special characters as '-', '@' and '.'. This allows us to identify the use of email addresses, hostnames and hyphen-separated terms within the arguments. Fraudsters are also more likely to include arguments that have no associated values in an attempt to add context to the URL. For example, consider the following shortened URL:

```
http://ow.ly/4mIpzC?Facebook-acount-disable
```

By including an argument with the key "Facebook-acount-disable", the shortened URL can be made to appear related to the target organisation. In normal operation, URL arguments exist as key-value pairs. This is not true in the case above, as the fraudster is simply using the argument for it's visual impact—rather than for passing information to the web server. We can identify this characteristic by measuring the number of arguments in the URL that do not exist as key-value pairs. In order to extract further information from the arguments, we also measure the length of the longest and shortest keys and values, as well as the average length of the keys and values.

4.3.9 Hostnames With An IP Address

Previous work has identified IP addresses within hostnames using regular expressions (Garera et al., 2007), or through measuring the number of '.' characters within the hostname (Whittaker, Ryner and Nazif, 2010).

Hostnames containing IP addresses are likely to contain a greater number of digits than hostnames that do not. If IP addresses are more common in phishing URLs than benign URLs, it stands to reason that the distribution of digits in the hostname varies between phishing URLs and benign URLs. Figure 4.8 shows the distribution of digits within the hostnames of a sample of 1000 phishing URLs and 1000 benign URLs from our April 2016 training data. We can see that the distributions do indeed differ, with phishing URLs having a larger spread in the number of digits. In order to capture this, we measure the proportion of digits within the host. Another interesting property of IPv4 addresses is that they contain maximum values of 255. To distinguish URLs containing high amounts of digits from URLs containing IP addresses, we also measure the highest numerical value observed within the hostname.

4.3.10 Long Hostnames

URLs with long hostnames are fairly well covered by the existing literature, with a total of seven simple lexical features identified. These features include the length of the hostname (Ma et al., 2011), the number of '@', '-' and '.' characters within the hostname (Zhang, Hong and Cranor, 2007) and the length of the longest token in the hostname (Le, Markopoulou and Faloutsos, 2011).

These features focus on detecting long hostnames, but greater focus could be placed on identifying particular methods commonly used by fraudsters to extend the length of their hostnames. From our sample phishing sites, we can observe that—in a similar fashion to the randomisation of directory names—frausters will commonly extend the length of their hostnames by using long hexadecimal strings generated by hashing random numbers. We aim to identify this by measuring the number of digits and hexadecimal characters present within the hostname.

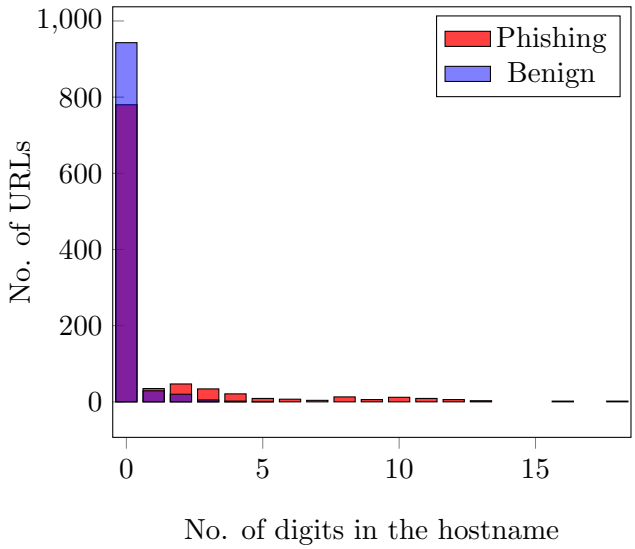


Figure 4.8: A histogram showing the distribution of digits within the hostnames of a sample of 1000 phishing URLs and 1000 benign URLs.

Feature	Targeted characteristic
Number of ':' characters in the hostname	Port number manipulation
Proportion of digits within the path	Randomised directory names
Number of hex characters in the path	Randomised directory names
Max. number of digits in any one directory	Randomised directory names
Number of digits in the path	Randomised directory names
Hash present in path (\/[a-fA-F0-9]+[\/\.])	Randomised directory names
Length of the longest sub-domain in the hostname	Generic domains for multiple targets
Average directory length	Compromised sites
Path length / total URL length	Compromised sites
Length of the shortest directory	Compromised sites
Number of hex characters in the arguments	Personalised URLs
Number of '@' characters in the arguments	Personalised URLs
Average argument value length	Deceptive arguments
Average argument key length	Deceptive arguments
Length of the shortest argument value	Deceptive arguments
Length of the longest argument key	Deceptive arguments
Presence of a non key-value argument	Deceptive arguments
Length of the shortest argument key	Deceptive arguments
Max. numerical value in the hostname	Hostnames with an IP address
Proportion of digits in the hostname	Hostnames with an IP address
Number of hex characters in the hostname	Long hostnames
Max. number of digits in any one sub-domain	Long hostnames
Number of digits in the hostname	Long hostnames

Table 4.1: Our novel simple lexical features and their targeted characteristics.

4.4 Summary

In section 4.2, we identified twenty five simple lexical features from existing literature (see table A.1). Using principal component analysis, we observed a significant overlap in the distribution of phishing and benign URLs when using these existing features (see figure 4.2). We explored the relationships between these features and the various characteristics of phishing URLs that we defined in section 3.3. From this, we were able to see that a number of the characteristics had fewer applicable features than others (see figure 4.1). We proposed that through identifying more features to capture the characteristics that are less well covered, the discriminative power of the feature set could be improved.

In section 4.3, we considered each of our phishing characteristics, proposing novel features to identify each one. A total of 27 novel simple lexical features were proposed. Table 4.1 details each of these novel features and the particular characteristic that we believe they can identify.

In chapter 7, we evaluate the effectiveness of both the existing simple lexical features and our novel set empirically—with the goal of providing a single effective simple lexical feature set.

Our work in section 4.2 also demonstrated that number of phishing URL characteristics cannot be identified using simple lexical features alone. In the following chapter, we discuss situated lexical features as a possible method of detecting such characteristics.

Chapter 5

Situated Lexical Features

Contents

5.1 Overview

In chapter 4, we explored the application of simple lexical features in the context of phishing URL classification. It was suggested that a number of phishing practices may remain undetected when using simple features due to a fundamental lack of context. For example, we cannot assess the similarity between a deceptive domain and the domain of the target organisation without knowing the potential target organisations. Simple lexical features operate solely within the context of the URL itself, and as such, it is difficult—if not impossible—to exploit the use of language within URLs due to the lack of a dictionary.

In this chapter, we consider the use of situated lexical features as a method of overcoming the limitations of simple features. Section 3.2.1 defines situated lexical features as statistics obtained from the URL which maintain a context outside of the URL itself, that is, they are "situated" within some environment. Through the use of an internal model, situated features allow us to identify the presence of particular words and phrases within URLs. This is significant, as it enables us to evaluate the URLs on a more semantic level—bringing us a step closer to perceiving the content of the URLs as a human would.

In section 5.2, we briefly review a number of situated features identified from existing literature. We discuss a number of potential limitations to these approaches, which informs the development of our approach; detailed in section 5.3.

5.2 Existing Situated Features

Table 5.1 lists the situated features that we identified during our literature survey. We identified a total of nine individual features and four more complex feature models.

The majority of the individual features require the maintenance of a list of potential phishing targets. Doing so presents the further problem of keeping the list updated, and due to time constraints, such features are beyond the scope of this project. Instead, we focus our attention on the use of token and URL-based n-grams to produce sparse feature vectors in high dimensions.

Perhaps the most ubiquitous situated feature model for textual classification is the *bag of words model* (Harris, 1954; Le and Mikolov, 2014). The bag of words approach originates from the field of document classification and natural language processing (Harris, 1954). In the simplest case, documents are represented as histograms of the frequencies of the words that they contain. These histograms can then be transformed into feature vectors by mapping the term frequencies to a dictionary. Each term in the dictionary is assigned a particular index within the feature vector, resulting in vector dimensions equal to the number of terms in the dictionary.

The work of Baykan et al. (2009) applies the bag of words approach in the context of URL topic classification. URLs are converted to lower case and separated into *tokens* by splitting them on any non-letter characters. Tokens containing fewer than two characters are removed, with the remainder forming the terms that represent the content of the URL.

One limitation of the simple bag of words approach is that the relative position of each of the terms is ignored (Le and Mikolov, 2014). This information could potentially be useful, as it provides context regarding the terms that surround a particular token. To this end, Baykan

et al. (2009) also investigated the application of n-grams (Jurafsky, 2000) as a method of positional encoding. Here, terms are formed from multiple tokens in a contiguous fashion, where the number of tokens in a term is defined by n. For example, given a bi-gram approach ($n = 2$) and the tokens "www", "secure", "example", "com", "login", "html", the following terms are generated:

1. "www secure"
2. "secure example"
3. "example com"
4. "com login"
5. "login html"

This approach retains some information regarding the terms surrounding a particular token, therefore encoding the positions of each particular term. A number of alternative methods for encoding positional information within the bag of words model have also been explored in the context of URL classification.

Baykan et al. (2011) revisit their work on URL topic classification, introducing the notion of extracting n-grams directly from the URL. Instead of parsing URLs into tokens, each URL is converted to lowercase and all non-letter characters removed. N-gram segments of the URL string are then taken using individual characters, as opposed to entire tokens. This approach improves classification accuracy, however, the increase in the dimensionality produced by this method would likely prove unmanageable at large scales (Blum et al., 2010).

Lexical Feature	Earliest Study
Hostname contains target organisation domain	Khonji, Iraqi and Jones (2011)
Path contains target organisation domain	Garera et al. (2007)
Number of characters between the end of the target organisation's name and the end of the hostname	Garera et al. (2007)
Suspicious TLD present	Ma et al. (2011)
Prefixed/Suffixed target organisation name present	Abdelhamid, Ayesh and Thabtah (2014)
Number of characters replaced with visually similar characters	Aburrous et al. (2010)
Use of a URL shortening service	Nguyen and Nguyen (2016)
Presence of suggestive tokens	Garera et al. (2007)
TLD in sub-domain or path	Xiang et al. (2011)
Presence of an unknown noun	V and A (2014)
Token-based n-gram (1, 4-8) frequencies	Baykan et al. (2009)
URL-based n-gram (4-8) frequencies	Baykan et al. (2011)
Token-based n-gram (1, 4-8) frequencies with explicit positional encoding	Baykan et al. (2011)
Regional token-based n-gram frequencies	Blum et al. (2010)
Token-based n-gram tuples	Khonji, Jones and Iraqi (2011)

Table 5.1: Situated features identified from previous research.

Along with URL-based n-grams, Baykan et al. (2011) also experiment with explicitly encoding the relative position of each token into the token string itself. This is done by duplicating each token and appending its position in the URL using an underscore and then the numerical value of the position. This approach doubles the dimensionality of the feature space, and has no significant impact on the classification accuracy (Baykan et al., 2011).

(Blum et al., 2010) take a different approach, defining a number of feature groups, each with separate dictionaries. Each feature group represents a token from a particular location in the URL. For example, the token from feature group "`domain{2}`" of the URL "`https://mail.example.com/login/`" is "`mail`". Feature vectors are formed from a number of feature groups, each targeting a specific position within the URL. This has the effect of greatly reducing the dimensionality of the feature vectors. However,by targeting specific locations within the URL, this approach discards potentially valuable information in the case where the URL does not conform to an expected pattern.

In the work of (Khonji, Jones and Iraqi, 2011), tokens are encoded as tuples, rather than simple terms. The position of each token is calculated relative to the position of the top level domain, such that tokens within the hostname are assigned a negative value, and tokens within the path a positive value. These tuples then form the terms of the dictionary.

5.3 Our Approach

5.3.1 Feature Extraction

N-Grams

Our situated lexical feature set uses token-based n-grams drawn from the URL in a manner similar to the work of Baykan et al. (2011) and Blum et al. (2010). First, URLs are split into hostname, path and arguments. Each of these strings are stripped of all non-letter characters not contained within the set of delimiters defined for that particular section of the URL:

Hostname	':', '/', '-', '.', '@'
Path	'/', '-', '.', ',', '@', '_', ';'
Arguments	'=', '&', '@', '-', '_', '/', '#', '.', ',', ';'

These characters were chosen as they are usually found separating words within each section of the URL. This places a focus on the semantic content of the URL, rather than the syntactic use of symbols and delimiters—as was the case for simple lexical features. Tokens are formed from the processed URL string by splitting on the non-letter characters identified above. The tokens are then converted to lower case, and any tokens longer than 32 characters, or shorter than 2 characters in length are removed. This is done to ensure that only tokens that are likely to contain words or common abbreviations are considered. The tokens from each section are then assembled in the order in which they appear in the URL, and n-gram segments taken from this list of tokens. A histogram is then produced representing the frequency of each segment within the URL.

Positional Encoding

Along with the frequency of each segment, we also record the position within the URL. Similarly to the work of James, L. and Thomas (2013), the position of each n-gram segment is calculated relative to the position of the top level domain. We then calculate the absolute distance from the number of tokens between the central token in the segment and the token of the top level domain. Rather than forming our feature vectors from the frequencies of each token, we instead populate the feature vector with the position of the token. This was done to avoid the increase in dimensionality of the dictionary that would occur by directly encoding the position into the segment itself.

5.3.2 Feature Selection

IDF

Our method of feature extraction described in section 5.3.1 does little to combat the issue of high dimensionality resulting from the bag of words approach. Using only uni-gram segments, a sample of 1000 phishing URLs and 1000 benign URLs from the April 2016 training data produces a total of 60,466 terms. If all of these terms were to be included in each feature vector, the resources required to support the system in a real world deployment would soon become infeasible. It is therefore clear that some form of dimensionality reduction is required.

Many forms of dimensionality reduction exist for applications within a batch learning context, such as principal component analysis (Fodor, 2002), Laplacian Eigenmaps (Van Der Maaten, Postma and Van den Herik, 2009) and Sammon mapping (Van Der Maaten, Postma and Van den Herik, 2009). Dimensionality reduction in the online context is alltogether more challenging, as we do not have a complete picture of our data before we begin.

In the work of Zhang, Hong and Cranor (2007), a statistic known as term frequency-inverse document frequency—or TF-IDF—was leveraged to select the terms from a document that best represent it within a corpus. As the name suggests, TF-IDF is calculated from the multiplication of two statistics: term frequency, `tf`, and inverse document frequency, `idf`.

Term frequency measures the frequency of the occurrence of some term, t, in a document, d (Salton and Buckley, 1988):

$$\texttt{tf}(t, d) = f_t : t \in d$$

Where $f_{t,d}$ denotes the frequency of term t.

Inverse document frequency represents the amount of information provided by a particular term (Salton and Buckley, 1988). It is calculated by taking the logarithmically-scaled inverse of the proportion of documents in the corpus that contain the term. Given a term, t, and some corpus, D, the inverse document frequency can be defined as follows:

$$\texttt{idf}(t, D) = \log \frac{||D||}{1 + |\{d \in D : t \in d\}|}$$

In our work, we leverage inverse document frequency to select the most representative terms from both phishing and benign URLs. During training, we store the frequencies of terms

within each URL, treating URLs as individual documents. These documents are split into two corpora, one containing phishing URLs and one containing benign URLs. When the number of documents observed reaches some threshold, we calculate the `idf` for all terms in each corpus, and select the top 5% of terms from each corpus for inclusion within the dictionary. The stored data is then cleared, and the process repeated. In this way, we hope to identify the most representative terms from each class of URL concurrently to the training process.

Performing this process over the 651,674 training URLs from April 2016, we identify a total of 3,473 terms—which is far more manageable than the 60,466 previously seen with only 2000 training URLs. Table B.1 shows a sample of the terms identified through this process. As we can see, a number of suspicious terms such as "verify", "signin" and "secure" are identified, along with the names of potential targets including "google", "dropbox" and "usaa". It is also observed that the dictionary produced contains a number of nonsensical terms such as "dcabccfaadd". Ideally, we wish to exclude such terms from the dictionary as they are unlikely to generalise beyond the URLs observed in the training set.

Metric Entropy

Previously, we observed that inverse document frequency could be leveraged to perform dimensionality reduction in an online setting. We also noted, however, that the resulting dictionary contained a number of nonsensical terms which appear to have been randomly generated. Such terms generally consist of long strings comprised of a few different characters.

Given a string of symbols, the Shannon Entropy provides an estimate of the minimum number of bits required to encode said string (Shannon, 2001). This is useful to us because the nonsensical strings often contain little variation in the number of characters, and thus require fewer bits to encode than most other strings of the same length. Consider some string of symbols, $X = \{x_1, x_2, ..., x_n\}$, and some vocabulary $Y = \{y_1, y_2, ..., y_z\}$. The Shannon entropy, H, of X can be found as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

Where $p(x_i)$ is the probability of observing x_i given the vocabulary Y:

$$p(x_i) = \frac{Y(x_i)}{n}$$

In our case, the vocabulary is a histogram of the frequency of x_i within X. Using the Shannon entropy, we then calculate the metric entropy (Batenkov, Friedland and Yomdin, 2015) of the term by dividing the Shannon entropy by the number of characters in the term. This is done in order to obtain a normalised value representing the randomness of the string. By only considering terms with a high enough metric entropy, we ensure that the nonsensical terms are excluded from our dictionary. Using the same set of 651,674 training URLs from April 2016, we sort the nonsensical terms according to their metric entropy and identify the maximum. We obtain a value of 0.25, and—using this as a minimum bound for selection into the dictionary—the number of identified terms is reduced from 3,473 to a more focussed 2,567.

5.4 Summary

In this chapter, we have considered a number of existing methods for the extraction of situated lexical features from phishing URLs. We discussed how the use of an internal model enables the exploitation of language within URLs—something which cannot be achieved using simple lexical features. In section 5.2, our attention was directed towards the use of a bag of words approach to represent the terms present within URLs. It was described how URLs could be tokenised and applied to a bag of words model, and we identified that this approach results in a loss of positional information. We then reviewed a number of existing methods for encoding positional information into a bag of words model, including the use of n-grams from tokens, n-grams from the URL and explicit encoding within the term itself. Limitations to each of these approaches were identified, informing a discussion of our approach to situated feature extraction and selection.

In section 5.3.1, we described our process for extracting token-based n-gram segments from the URL, and our novel method of positional encoding. We discovered that this approach produced infeasibly large feature spaces, and so, in section 5.3.2, we investigated the use of inverse document frequency and metric entropy for dimensionality reduction in an online context.

In the next chapter, we move on to discuss online learning in greater depth, as well as detailing the implementation of our classifier.

Chapter 6

Online Learning

Contents

6.1 Overview

In this chapter, we discuss online learning in greater detail. In particular, we describe the implementation of an ensemble learning method as an extension to the Hoeffding tree.

In section 6.2, we define the classification problem and examine online learning in contrast to batch-based approaches.

Section 6.3 focusses on decision trees. Here, we describe the fundamental principles behind decision trees, and identify the Hoeffding tree as a particular method for online learning.

We then move on to discuss an existing implementation of the Hoeffding tree, known as the "Very Fast Decision Tree"—or VFDT (section 6.4). In section 6.5, we develop a number of improvements upon the VFDT for use within our work.

Finally, in section 6.6, we consider random forests as a potential method of improving the Hoeffding tree, and detail the implementation of an online random forest classifier using our modified VFDT.

6.2 Background

6.2.1 The Classification Problem

Given two or more distinct classes of data, classification may be broadly described as taking some input data and assigning it to a particular class (Alpaydin, 2010). Ideally, we wish to assign the input data to the class that it actually belongs to. This is achieved through the use of *discriminants* and a *predictor*. Alpaydin (2010) defines a discriminant as a function that separates the examples of different classes. A predictor makes use of one or more discriminants to select the most likely class given the discriminants of previous examples.

More concretely: We define a *discriminant* as some function f mapping raw observational data, D, to some value, $x = f(D) : x \in \mathbb{R}$.

Given some set of discriminants $F = \{f_1, f_2, ..., f_n\}$, we can define a *feature vector* X in $d \leq n$ dimensions as $X = \langle x_1, x_2, ..., x_d \rangle$, where $x_i = f_i(D) : f_i \in F$

Given a set of N distinct classes Y, and an example set of feature vectors T of the form $t_i = (X, y) : y \in Y$, we can construct a predictor g through some training process $t : T \mapsto g$.

The predictor function g maps some feature vector X to a class $y \in Y$ such that:

$$p(y) = \max_{1 \leq i \leq N} p(y_i|T)$$

Thus, our classification problem becomes:

Produce some predictor, $y = g(X)$ such that the class y for some unseen feature vector X can be accurately predicted.

6.2.2 Binary Classification

Our classification problem is binary, that is, our set of possible classes consists of only two elements: phishing and benign. We can refine our earlier definition of the classification problem for the binary case. For binary classification, we need only consider the probability of belonging to one of our two classes: the *positive* class. Given some feature vector X, and our classes $Y = y_1, y_2$ our predictor function g can be redefined as a function mapping X to some class $y \in Y$ such that:

$$g(X) = \begin{cases} y_2, & \text{if } p(y_2) > \rho \\ y_1, & \text{otherwise} \end{cases}$$

Where y_2 is the positive class, y_1 is the negative class and ρ is some confidence bound. In our work, we define "phishing" to be the positive class, and "benign" to be the *negative* class.

The use of a confidence bound allows us to find a trade-off between the number of false positives and the recall of the classifier. This is useful to us because marking benign sites as phishing could potentially cause more harm than marking a phishing site as benign, and using a confidence bound enables tuning the classifier towards minimising this risk. This is discussed in further detail in section 7.2.2, where we consider how to select an optimum value for ρ

6.2.3 Batch vs Online learning

Traditional approaches to solving the classification problem use batch-based methodologies (Blum et al., 2010). In batch learning, all of the training data is available simultaneously, and an optimal predictor is generated using the training data in a single 'batch'. Once the classifier has been trained, the predictor is never updated again. If the distribution being modelled changes, then the classifier must be re-built using a new batch of training examples (Le, Markopoulou and Faloutsos, 2011). Because the entire sample of data is available, batch-based approaches can make more informed statistical decisions, and as such are known to generalise well (Blum et al., 2010). Batch-based approaches have a number of disadvantages, however. The training process requires the entire data set to be loaded into memory, which may not be feasible for massive amounts of data (Domingos and Hulten, 2000; Le, Markopoulou and Faloutsos, 2011). In addition, as touched upon previously, models produced with a batch-based approach cannot react to changing distributions, and as a result require frequent re-training in order to stay up to date (Blum et al., 2010; Whittaker, Ryner and Nazif, 2010). These limitations are particularly disabling in the case of phishing URL classification, due to the massive number of URLs and their dynamic nature (Le, Markopoulou and Faloutsos, 2011; Blum et al., 2010).

In contrast to batch learning, online learning methods update the predictor continuously, with training instances arriving in a sequential fashion (Le, Markopoulou and Faloutsos, 2011; Saffari et al., 2009). As the data is handled one item at a time, the memory required by online learning methods is often far less than that of a batch-based approach (Blum et al., 2010), and the process of training is often much faster (Le, Markopoulou and Faloutsos, 2011). It has been shown that online algorithms can incrementally adapt to changing data (Ma et al., 2011), making them more applicable than batch learning when dealing with non-stationary data (Laskov et al., 2006).

Due to the rapidly changing nature of phishing URLs, it is important that our chosen method

responds quickly in order to remain effective (Blum et al., 2010). This makes online learning the better choice for our application, due to the low training latency, minimal resource requirements and the ability to update the predictor continuously.

6.3 Decision Trees

6.3.1 Outline

Decision trees are a popular choice of classifier for the problem of phishing site classification (Ludl et al., 2007; James, L. and Thomas, 2013; V and A, 2014). Much of the current literature explores the use of various batch learning decision trees such as C4.5 and CART to achieve high precision with impressive processing rates (James, L. and Thomas, 2013; V and A, 2014). Because high processing rates are critical in ensuring that our system provides the most effective protection possible (Blum et al., 2010), we have selected a tree-based prediction model for use this project.

In general, decision trees represent the classification problem space in the form of a tree structure, where nodes of the tree represent a test on some particular feature, and each branch of the tree represents a possible outcome of said test. Leaves of the tree correspond to a particular class prediction, and as such the predicted class of a particular feature vector can be calculated by traversing the tree, starting at the root node and following the branches based on the tests until a leaf node is reached. Learning is achieved by replacing leaves with nodes, where the "best" attribute to test on is given by some heuristic measure. Batch-based decision trees are able to iteratively expand the model using the entire data set, until some desired precision or model depth is met—or the maximum number of iterations is exceeded. Due to the structure of decision trees, predictions can be made in $\mathcal{O}(\log n)$ time, leading to fast processing rates with high prediction accuracies (Saffari et al., 2009).

6.3.2 Hoeffding Trees

Hoeffding Trees provide a way of building decision trees in an online fashion (Domingos and Hulten, 2000). The main problem with the construction of an online decision tree is deciding the minimum number of examples that are required at a node before a test can be selected. Hoeffding Trees use a statistical measure, known as the Hoeffding bound, to decide whether a sufficient number of examples have accumulated at a node to be able to make a decision with a confidence greater than some specified minimum.

Hoeffding's Inequality

Hoeffding's Inequality provides a bound on the probability that the mean of n real valued random variables $x_i \in X$ deviates from it's expected value (Domingos and Hulten, 2000). Let $x_0, ..., x_n$ be independent random variables strictly bounded by the interval $[a_i, b_i]$. We can define the empirical mean of these variables by

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Hoeffding's inequality states that

$$\mathbb{P}(\overline{X} - E(\overline{X}) \geq \epsilon) \leq exp\left(- \frac{2n\epsilon^2}{R^2} \right)$$

Where $R = \sum_{i=1}^{n}(b_i - a_i)$, the range of the random variable x.

That is, that the probability of the observed mean deviating from the expected mean by at least ϵ is bounded by some value relative to the number of observations. Therefore, given some probability δ, we can say with confidence $1-\delta$ that the true mean of some feature given n observations is at least $\overline{X} - \epsilon$. We can derive a value for ϵ in the following fashion:

$$\begin{aligned}
\delta &\leq exp\left(- \frac{2n\epsilon^2}{R^2} \right) \\
ln(1) - ln(\delta) &\geq \frac{2n\epsilon^2}{R^2} \\
R^2 ln\left(\frac{1}{\delta}\right) &\geq 2n\epsilon^2 \\
\epsilon^2 &\leq \frac{R^2 ln\left(\frac{1}{\delta}\right)}{2n} \\
\epsilon &\leq \sqrt{\frac{R^2 ln\left(\frac{1}{\delta}\right)}{2n}}
\end{aligned}$$

Thus, we can form a conservative estimate of the deviation from the mean using the maximum bound of the value for ϵ above. This bound is known as the Hoeffding bound.

Using the Hoeffding Bound

As mentioned previously, in order to select a feature to test on at a particular node, some heuristic measure $\overline{G}(x_i)$ is needed. In the case of our implementation, this measure is information gain, however, alternatives—such as the Gini index—do exist (Domingos and Hulten, 2000).

At each leaf l of the tree, we store the frequencies of each of the n observed values of every feature x_i in every training vector X, for each class $y \in Y$. From these values, we can calculate the entropy of the leaf $E(l)$ as a measure of the balance between the classes of the accumulated examples. The entropy $E(l)$ of the leaf l can be expressed as follows:

$$E(l) = - \sum_{\forall y \in Y} p(y|l) \log_2 p(y|l)$$

Where $p(y)$ is the probability of observing the class y given the examples that we have accumulated at node l.

Given some value v for a feature x_i, we can define the entropy $E(x_i, v, l)$ at leaf l as follows:

$$E(x_i, v, l) = -\sum_{\forall y \in Y} \frac{p(y|v, x_i, l)}{p(v|x_i, l)} \log_2 \frac{p(y|v, x_i, l)}{p(v|x_i, l)}$$

Where $p(y|v, x_i, l)$ is the probability of observing class y given a value of v and feature x_i at node l, and $p(v|x_i, l)$ the probability of observing value v given feature x_i.

Following this, we can calculate the weighted sum $\overline{E}(x_i, l)$ of the entropies of all observed values $V = \{v_1, v_2, ..., v_d\}$ for the feature x_i at leaf l:

$$\overline{E}(x_i, l) = \sum_{n=1}^{n=d} \frac{l(v_n|x_i)}{N} E(x_i, v_n, l)$$

Where $l(v_n|x_i)$ is the number of examples observed with value v_n for feature x_i at node l, and N the total number of examples seen at l.

Using this, we can compute the information gain $\overline{G}(x_i, l)$ provided by splitting on some feature x_i at leaf l by subtracting the weighted entropy sum of the feature from the entropy of the leaf:

$$\overline{G}(x_i, l) = E(l) - \overline{E}(x_i, l)$$

Using the Hoeffding bound, we can provide a probabalistic confidence that the feature chosen to split on using n observations (where n is as small as possible) is the same as would be chosen given infinite observations (Domingos and Hulten, 2000). In the case of information gain, we wish to maximise $\overline{G}$, where x_a is the feature with the largest observed $\overline{G}$ after n observations, and x_b is the feature with the second largest. Let $\Delta\overline{G} = \overline{G}(x_a, l) - \overline{G}(x_b, l) \geq 0$ be the difference in their information gains. Given a desired δ, the Hoeffding bound guarantees that x_a is the correct choice with probability $1 - \delta$ after n observations if $\Delta\overline{G} > \epsilon$ (Domingos and Hulten, 2000).

Upon selecting some feature x_a to split the leaf l, the value to split on is calculated using the weighted mean $\overline{X}_a$ of the observed values $V = \{v_1, v_2, ..., v_n\}$ for x_a:

$$\overline{X}_a = \sum_{i=1}^{i=n} v_i \frac{l(v_i|x_a)}{N}$$

Where $l(v_i|x_i)$ is the frequency of examples at l given value v_i and feature x_a, and N is the total number examples observed at l.

When classifying future examples, the value of the feature x_a is considered, and the example passed to the left child unless the value of x_a is greater than $\overline{X}_a$, in which case it is sent to the right child. This process is repeated along the branches of the tree until a leaf node is reached. Here, a prediction $g(X)$ is given using our binary classification rule defined in section 6.2.2:

$$g(X) = \begin{cases} y_2, & \text{if } p(y_2) > \rho \\ y_1, & \text{otherwise} \end{cases}$$

Where y_2 is the positive class (phishing), and y_1 the negative class (benign).

The Hoeffding Tree Algorithm

We summarise the Hoeffding tree training algorithm used in our work in algorithm 1. The Hoeffding tree classification algorithm is detailed in algorithm 2.

for *training example* ($X = \langle x_1, x_2, ..., x_n \rangle$,$y$) *in data stream* S **do**
  sort (X,y) into a leaf node l using the existing tree
  for *each value* $x_i \in X$ **do**
    update the feature-value count $l(x_i, v, c)$ in leaf l
  end
  if *the examples at leaf* l *are not all of the same class* **then**
    compute the entropy of l, $E_0 = E(l)$
    for *each feature index* $1 \leq i \leq n$ **do**
      compute the weighted sum $\overline{E}_i = \overline{E}(x_i, l)$
      compute the information gain $\overline{G}_i = \overline{G}(x_i, l) = E_0 - \overline{E}_i$
    end
    $x_a \leftarrow \max_{1 \leq i \leq n} (\overline{G}_i)$
    $x_b \leftarrow \max_{1 \leq i \leq n | i \neq a} (\overline{G}_i)$
    compute the hoeffding bound, ϵ
    if $\overline{G}(x_a) - \overline{G}(x_b) > \epsilon$ **then**
      split l on x_a
      set the split value as the weighted mean $\overline{X}_a$
      remove the feature-value counts for leaf l
    end
  end
end

Algorithm 1: A summary of the Hoeffding tree training algorithm, based on the work of Domingos and Hulten (2000).

for *instance* ($X = \langle x_1, x_2, ..., x_n \rangle$,$y$) *in data stream* S **do**
  $n \leftarrow$ root note
  while n *is not a leaf node* **do**
    $a \leftarrow$ split attribute for node n
    $\overline{X}_a \leftarrow$ split value for node n
    if $x_a > \overline{X}_a$ **then**
      $n \leftarrow$ right child node
    else
      $n \leftarrow$ left child node
    end
  end
  $y_1 \leftarrow$ negative class
  $y_2 \leftarrow$ positive class
  if $p(y_2|n) > \rho$ **then**
    return y_2
  else
    return y_1
  end
end

Algorithm 2: A summary of the Hoeffding tree classification algorithm, based on the work of Domingos and Hulten (2000).

6.4 Very Fast Decision Trees

The "Very Fast Decision Tree", or VFDT, is an implementation of the Hoeffding tree—described in the work of Domingos and Hulten (2000). The VFDT model extends the Hoeffding tree algorithm, proposing a number of optimisations to make the model more applicable to the real-world. We carry a number of these optimisations over into our own work, which we discuss in the following sections.

6.4.1 Tie Confidence

In sections 6.3.2 and 6.3.2, we discussed our use of the hoeffding bound in identifying the best attribute to split a node on given some minimum confidence $1 - \delta$. The 'best' attribute x_a for some leaf l was defined as the attribute with the highest information gain $\overline{G}(x_a, l)$. If the difference between $\overline{G}(x_a, l)$ and the information gain of the second best attribute $\overline{G}(x_b, l)$ is greater than the Hoeffding bound ϵ given n observations, we select x_a as the attribute to split on. In the case where there is little difference between the information gain of x_a and x_b, a large number of examples would be required to reach a decision, which is wasteful (Domingos and Hulten, 2000). To overcome this, we define a tie confidence, τ, such that we always split on x_a if $\Delta\overline{G} < \epsilon < \tau$ (Domingos and Hulten, 2000). Whilst this means that we sometimes have less confidence in the decision to split, we are able to move past potentially expensive decisions—allowing the model to continue to grow.

6.4.2 Split Threshold

According to Domingos and Hulten (2000), the majority of the time cost during training occurs when computing the information gain for each feature. It is unlikely that the information gain of any particular feature will change dramatically between any two consecutive samples, and so it is therefore inefficient to recompute information gain for every example (Domingos and Hulten, 2000). To avoid this, we define a split threshold n_{min} to represent the minimum number of new examples required at a node before splitting the node may be re-considered. This effectively reduces the time cost of computing the information gain by a factor of n_{min} (Domingos and Hulten, 2000), improving the rate of training.

6.4.3 Leaf Deactivation

As the number of leaves within the tree increases, so does the memory required to store the frequencies of the values observed (Domingos and Hulten, 2000). In order to conserve memory and therefore construct a larger model, we deactivate nodes once the confidence in a positive classification at that node exceeds the specified confidence bound, ρ. Once a node is deactivated, the majority of the frequency information is discarded and the node is no longer considered for splitting. The only data maintained by a deactivated node are the frequencies of the classes of each example, and the total number of examples observed. These values must be maintained and updated to ensure that we can re-activate the node in the event that the confidence drops below the confidence bound. Nodes are considered for re-activation periodically, after every 1000 training examples.

This is safe to do so as we only require a particular confidence in the predictions, ρ. Refining

a node beyond the confidence bound does not benefit us as much as reclaiming some of the resources held by said node.

6.5 Building Upon VFDT

The original implementation of VFDT by Domingos and Hulten (2000) is written in C, and—as a proof-of-concept—omits a number of features that could potentially be useful in a real world deployment. Our implementation, written in Java, builds upon VFDT by providing a number of extra features, discussed in the following sections.

6.5.1 Stabilising Prediction Confidence

In the original VFDT training algorithm, nodes are split in a binary fashion into two child nodes. Upon splitting, each of these child nodes becomes a leaf node, and the frequency data of the parent node is discarded. Due to the structure of the frequency information, it is impossible to reconstruct the examples seen by the parent node, and as a result, this information cannot be passed down to the child nodes. This means that the child nodes are initialised with no previous examples, causing the positive prediction confidence to drop to zero. This could result in incorrect classifications, as the online nature of the classifier means that prediction can occur at any point during the training process. For example, given the confidence bound $\rho = 0.7$, a node with a positive prediction confidence of 0.76 would yield the class "phishing". If the decision was then made to split this node, the positive prediction confidence would drop to 0.0, despite the fact that any examples reaching the node would very likely be phishing.

This fluctuation in the positive prediction confidence not only presents an opportunity for both false positives and false negatives to arise, but could also cause unwanted side-effects in systems utilising the numerical value of the prediction confidence. If a customer facing toolbar, for example, was to display the positive prediction confidence as a "likelihood" of a URL being suspicious, the large fluctuations in values—even given the same URL—would make the toolbar appear less trustworthy.

To address this, we introduce the concept of a *learning* node alongside leaf nodes. When splitting a leaf node into two child nodes, the child nodes become learning nodes, and the parent remains a leaf node. Training examples are filtered all the way down the tree to the learning nodes, where the training process occurs as normal. Learning nodes become leaf nodes once the difference in entropy between the parent node and the current node is less than the Hoeffding bound calculated from the current node. This ensures that the learning node is not used for classification until the fluctuation has settled down, and an appropriate number of examples have been observed. As a result, the positive prediction confidence changes less dramatically over time. Once both children become leaf nodes, the resources used by the parent for maining frequency data may be freed for use elsewhere.

One limitation of this practice is that it may take some time before newly created nodes contribute to the prediction, resulting in a slightly less accurate model than the original VFDT given the same data.

6.5.2 Variable-Length Vectors

The original VFDT implementation assumed that the length of the feature vectors would remain constant throughout the operation of the classifier. This limitation makes it impossible to use a bag-of-words feature extraction methods such as those detailed in chapter 5. Our implementation uses data structures that are capable of expansion, to ensure that the classifier can cope with changing dimensionality.

The addition of this feature also provided us with the opportunity to exploit hashing to reduce the amount of resources required to store frequency data at each node. The original VFDT implementation stores frequency data in three-dimensional arrays indexed by feature index, feature value and class. Inevitably, this approach wastes memory by having to store zero values, and restricts the range of the values allowed for each feature. Our approach uses the Java `ArrayList` to store frequency entries for each feature in a variable-sized vector. Each entry in the `ArrayList` contains a `HashMap` mapping the observed values of that particular feature to a `Tuple` of integers. This tuple represents the number of examples observed for each class, positive and negative. By using hash-mapping to store the frequencies of values for each feature, we avoid the need to allocate memory for values that are never observed, whilst also allowing for a greater variation in feature values.

6.5.3 Database Utilisation

Due to the large number of training instances required to build the VFDT model (Domingos and Hulten, 2000), it is beneficial to have the ability to save the state of the model at regular intervals in order to ensure that the model can be reconstructed in the event of a system failure. Our implementation utilises the H2 [1] database for this purpose. Database updates are enqueued whenever a node is split, activated or deactivated. These updates are then periodically committed to the database in an asynchronous fashion so as not to interrupt training and classification. In the event that the system must be restarted, the predictor model can be rebuilt from the state contained within the database.

For our bag of words approach, the use of a database also allows us to offload the dictionary onto the disk. This is particularly beneficial to us as the dimensionality of the dictionary increases over time. Using disk storage as opposed to RAM storage for modelling the dictionary allows the system to run over a much longer period of time—ensuring that we are fully exploiting the benefits of online learning.

6.5.4 Web API

The processing rate benchmarks for content-based classification, provided in the work of Whittaker, Ryner and Nazif (2010), were obtained from a large-scale system. In order to make our results more comparable to such a system, our implemented system utilises both a database and a JSON web API. URLs may be enqueued for training using the `train` endpoint, as demonstrated below:

Endpoint request:	`/train?url=<encoded_url>&label=PHISHING`
Response:	`{"submitted":true}`

[1] http://www.h2database.com/html/main.html

Similarly, up two four simultaneous classifications can be requested via the `classify` endpoint:

Endpoint request:	`/classify?url=<encoded_url>`
Response:	`{"result":"PHISHING","confidence":0.9238371441686885}`

6.6 Random Forests

6.6.1 Outline

Random forests are an example of an ensemble prediction method, which means that they exploit a collection of simpler predictor models in order to produce a classification. More specifically, random forests are a collection of tree predictors where each tree depends on a random subset of both the training data and the features within the vectors themselves—sampled independently and with the same distribution for all trees in the forest (Breiman, 2001). During prediction, each tree casts a 'vote' and the most popular class is selected (Breiman, 2001).

The rationale behind the random forest methodology is that by training each tree on a subset of the available features, different trees will become more or less effective at recognising particular features than others. By combining the outputs of these trees into a majority, the resulting system becomes more robust to noisy data (Breiman, 2001). Because random forests still utilise tree-based models, they have been shown to be faster than other boosting alternatives, such as AdaBoost, whilst still maintaining a comparable prediction accuracy (Saffari et al., 2009; Breiman, 2001). This is attractive to us, as it is unlikely that the training data provided to our classifier is a complete ground truth (see section 7.1). Random forests provide us with a straightforward method of improving the prediction accuracy of our VFDT-based system, with little sacrifice in terms of processing speed.

6.6.2 The Online Random Hoeffding Forest

In a batch learning context, bagging is often used to train each decision tree on a subset of the training examples (Breiman, 2001). This is where a random selection of the training data is chosen without replacement, such that the distribution of training examples is the same across all of the decision tress in the forest (Breiman, 2001). In order to construct an online random forest from our VFDT-based system, we use an online bagging method identified in the work of Saffari et al. (2009). Because the training samples arrive one at a time in an online context, we cannot select a subset of the training data in the same way as batch-based bagging. Instead, for each new training sample, each tree is trained k times in a row using the sample, where k is a random number generated from a Poisson distribution centred around 1 (Saffari et al., 2009). It has been proven that, given enough samples, this method converges to same result as traditional batch-based bagging approaches (Oza and Russell, 2001).

In addition to the sampling described above, each decision tree considers a random subset of the available features when deciding the node to split upon. The proportion of features considered is controlled by the inclusion bound, μ.

We refer to our online random forest as the "Online Random Hoeffding Forest". To the best of our knowledge, a VFDT-based online random forest has never been developed using the

methods described in this chapter—and certainly hasn't been applied before in the context of online phishing URL classification.

6.6.3 Adapting To Changing Distributions

Despite their speed and respectable prediction accuracy, tree-based prediction models suffer an inherent limitation due to the nature of their structure. Once the decision to split at a particular node has been made, there is no way to re-evaluate the decision without discarding the rest of the tree below said node (Saffari et al., 2009). This means that if the distribution being modelled is dynamic, changes in the distribution may not be reflected in the prediction model (Saffari et al., 2009).

This is particularly important to us, as the nature of phishing URLs is likely to change over time as different phishing targets become popular, or as fraudsters develop new practices (Blum et al., 2010). We hope to mitigate this through the selection of an online classifier, as the model can always generate new branches in response to changes in the training data. This is not a perfect solution, however, and inevitably the classifier will have to be re-trained from the ground up if the modelled distribution changes significantly. We believe that this is only likely to occur over a period of months or years, so the benefit over batch-based methods of not having to re-train the classifier every week is not lost.

One potential method of ensuring that the model can adapt to changing data is provided in the work of (Saffari et al., 2009). Here, they suggest the continual removal of the most erronious tree from the random forest, replacing it with a new tree. Unfortunately, due to time constraints, we could not explore this possibility; however, we believe this to be an interesting area for future development.

6.7 Summary

In this chapter, we have discussed online learning, and given details of the online random forest classifier that has been developed as part of this project.

In section 6.2, we defined the classification problem, and described online learning approaches in contrast with traditional batch learning. Here we discussed how online learning approaches are faster and less resource intensive than batch learning methods, making online learning the better choice for the classification of phishing URLs.

Section 6.3 reviews decision trees as a form of predictor modelling. We showed that decision trees are a popular choice amongst previous research, and that due to their structure they can perform classification in $\mathcal{O}(\log n)$ time. Given the fast processing rates and impressive prediction accuracies of decision trees from previous literature, we selected the decision tree as our prediction model for this dissertation. We then expanded upon the use of decision trees in an online context, detailing the particular algorithm used in our work, the Hoeffding tree. The Hoeffding tree exploits a statistic known as the Hoeffding bound to decide whether a node in the tree should be split given the number of examples observed.

Following this, in section 6.4, we described an existing implementation of the Hoeffding tree—written in C—the Very Fast Decision Tree. We identified a number of optimisations from VFDT—namely the use of a tie confidence, split threshold and leaf deactivation—and described how each of these optimisations was included within our own work.

In section 6.5, we described a number of innovations made within our implemented system to improve upon the limitations of the previous VFDT implementation. The notion of a learning node was introduced to smooth out the fluctuations in prediction confidence that had been observed with the previous VFDT methodology. Memory efficiency was also improved through the use of hashing, and the flexibility of the model was improved to cope with variable-length feature vectors. The use of a database to reduce memory usage and to rebuild the predictor model in the case of system failure was described. A JSON web API was also developed to ensure that our results are comparable to those drawn from large-scale systems.

Finally, in section 6.6, we explored random forests as a method of improving tree-based classifiers. Here, we covered our implementation of the online random Hoeffding tree, which—to the best of our knowledge—is unique in the field of phishing URL classification.

In the following chapter, we turn our attention to experimental design, detailing our methods of data collection and the empirical evaluation of our classifier and various lexical feature sets.

Chapter 7

Empirical Evaluation

Contents

7.1 Data Sets

7.1.1 Data Collection

One of the biggest challenges faced in previous work is the collection of a representative sample of URLs (Khonji, Iraqi and Jones, 2011; Baykan et al., 2009). The sheer volume of potential phishing targets means that collecting a sufficient number of example URLs is often a time consuming process. To make matters worse, anti-phishing organisations rarely provide phishing URL data for free, as such data forms a profitable part of their business. As a result, the majority of previous work draws phishing samples from PhishTank [1], which is a free service operated by OpenDNS (Huh and Kim, 2012; Huang, Qian and Wang, 2012; V and A, 2014). PhishTank relies on a community of users to submit and verify active phishing attacks. Due to the manual nature of the verification process, the volume of URLs identified forms only a subset of phishing attacks on the whole. In June of 2016, the PhishTank community verified 16,360 phishing sites (PhishTank, 2016)—a mere 10% of the 158,782 unique attacks identified by the APWG during the same period (APWG, 2016). This means that it could take months to acquire enough data to simulate even a single month of operation in the real world. Furthermore, because the sites on PhishTank are verified by users it is possible that only the more obvious phishing URLs will be identified. This casts some doubt over whether PhishTank URLs can form a truly representative sample of the real-world distribution of phishing URLs.

This dissertation overcomes these limitations by using real-world phishing data provided by the anti-phishing organisation, Netcraft [2]. Netcraft's anti-phishing systems have been operating since 2005, with over 31.6 million unique phishing sites detected as of April 2017 (Netcraft, 2017). Our phishing data set consists of the unique phishing URLs that were detected and blocked by Netcraft during the months of April, May, June, July, August and September of 2016. This comes to a total of 3,772,994 phishing URLs, which—to the best of our knowledge—is the largest collection of phishing URLs ever applied in the context of online lexical phishing URL detection.

Collecting a sample of benign URLs presented us with a challenge. This is because URLs that have been submitted to anti-phishing services, such as Netcraft, are inherently suspicious in nature, whether they are blocked or not. We therefore decided that collecting benign URLs from Netcraft would not be representative of the real world. Instead, we used an open source web crawler (Ganjisaffar, 2017) to collect benign URLs. One potential problem with this approach is that there is no way to guarantee that the sites collected will be benign. In order to minimise the likelihood of crawling a phishing site, we began by crawling pages from DMOZ [3]; a peer-reviewed directory of websites. We chose to use DMOZ as it's peer-reviewed nature makes it less likely to contain phishing sites—though we still cannot discount the possibility of a compromised site making it into our dataset. Our decision was also influenced by a number of related studies that also utilised DMOZ for the collection of benign URLs (V and A, 2014; Feroz and Mengel, 2015; Le, Markopoulou and Faloutsos, 2011). The crawling process fetches the HTML content of a URL, extracts the URL links within the content and then repeats this process for the newly discovered URLs. To ensure that our sample did not contain large numbers of URLs from any one site, the maximum number of URLs crawled

[1] https://phishtank.com/
[2] https://netcraft.com/
[3] http://www.dmoz.org/

for a particular hostname was set to five. By setting a maximum value to five, it ensures that we collect URLs from pages other than the front page—giving us greater variety in URL structure—whilst ensuring that no one site is unfairly represented. The web crawler ran for seven days from the 5th of January 2017 to the 12th, collecting 3,772,994 URLs—giving us an equal proportion of phishing and benign URLs, and bringing the total number of URLs in the data set to 7,545,988. An important thing to note here is that the benign URLs were collected over a matter of days, whilst our phishing dataset was collected over the course of six months. The difference in time-scales during data collection presents a possible limitation of our study, however, we believe it to be a safe assumption that benign URLs change less over time than phishing URLs. This is because phishing URLs are released in campaigns targeting specific organisations, making it highly likely that the URLs seen will change from day to day as different targets become popular. The appearance of benign URLs is only likely to change as new development practices and tools are adopted, which could take months or even years.

7.1.2 Training and Testing Sets

Figure 7.1 shows the number of phishing URLs collected from Netcraft during the months from April to September of 2016. Our set of benign URLs was distributed between each of these months such that the number of phishing and benign URLs for each month were equal. The data for each month was then split into training and testing sets using an 80-20 training-testing split. Table 7.1 shows the size of each resulting set of data.

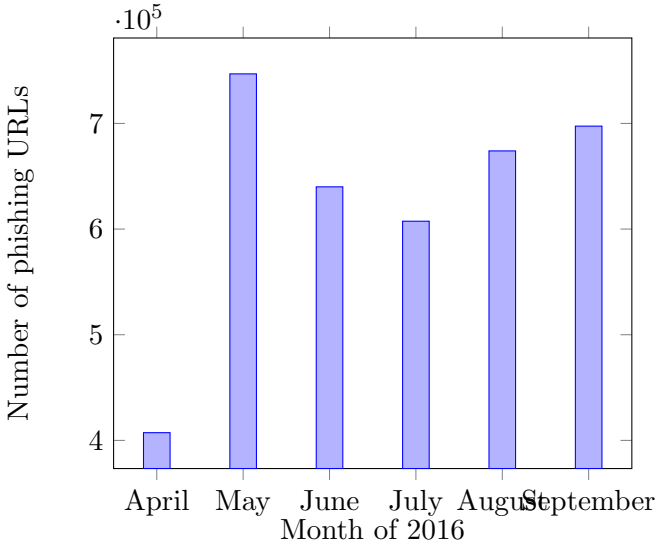


Figure 7.1: A histogram of the number of phishing URLs in our data set, categorised by the month in which they were collected.

Month (2016)	Number of URLs		
	Training	Testing	Total
April	651,674	162,920	814,594
May	1,195,028	298,756	1,493,784
June	1,023,946	255,986	1,279,932
July	971,894	242,974	1,214,868
August	1,078,364	269,592	1,347,956
September	1,115,886	278,971	1,394,857
Total	6,036,792	1,509,199	7,545,991

Table 7.1: A monthly breakdown of the number of URLs in our training and testing sets.

7.2 Optimisation

7.2.1 Performance Measures For Binary Classification

In order to discuss optimisation, we must first define a number of performance measures for use during our empirical investigation. Given some binary classification problem, the outcome will be one of four distinct possibilities (Powers, 2011). In the case where the real class of an item is the positive class, the classifier can either correctly label the item with the positive class (a *true positive*, `tp`), or incorrectly with the negative class (a *false negative*, `fn`). Conversely, in the case where the real class of an item is the negative class, the classifier can either correctly label the item with the negative class (a *true negative*, `tn`), or incorrectly with the positive class (a *false positive*, `fp`). This is demonstrated visually in table 7.2. We can derive further performance measures from these four simple statistics:

1. **Precision**. The precision of a binary classifier represents the confidence in a positive prediction, that is, the probability that the result is correct given a positive classification (Powers, 2011). Precision is expressed as follows:

$$\texttt{precision} = \frac{\sum \texttt{tp}}{\sum \texttt{tp} + \sum \texttt{fp}}$$

2. **Recall/True Positive Rate**. The recall, or true positive rate (`tpr`), of a binary classifier represents the likelihood of correctly classifying an item belonging to the positive class (Powers, 2011). Recall can be expressed as follows:

$$\texttt{recall} = \frac{\sum \texttt{tp}}{\sum \texttt{tp} + \sum \texttt{fn}}$$

3. **Fall-Out/False Positive Rate**. The fall-out, or false positive rate (`fpr`) of a binary classifier represents the probability of incorrectly classifying an item as belonging to the positive class (Powers, 2011). Fall-Out is defined by the following expression:

$$\texttt{fall-out} = \frac{\sum \texttt{fp}}{\sum \texttt{fp} + \sum \texttt{tn}}$$

	Real Positive	Real Negative
Predicted Positive	true positive	false positive
Predicted Negative	false negative	true negative

Table 7.2: A visual representation of true positive, false positive, false negative and true negative in the context of binary classification.

Whilst further performance measures exist, precision, recall and fall-out are the most applicable to our problem. When using a binary classifier to identify phishing URLs, the positive class represents a classification of "phishing", and the negative class a classification of "benign". Classifying a benign site as phishing, i.e. generating a false positive, could result in unnecessary service interruptions for the owner of the site, making it critical that we avoid false positives as much as possible. The fall-out is therefore an important statistic, as it allows us to assess the likelihood of this occurring. Precision is also a useful statistic here, as it provides a measure of our confidence in classifying a site as phishing. In addition, if we wish for our classifier to be effective in mitigating the threat of phishing, we must ensure that the majority of phishing sites are being caught. Recall allows us to measure this, however, by maximising recall we will also introduce a greater number of false positives (Cortes and Mohri, 2004).

We can visualise this trade-off using the *Receiver Operating Characteristic*, or ROC. The ROC plots recall as a function of fall-out (Cortes and Mohri, 2004). An example of this is shown in figure 7.2. Ideally, we wish to find the optimum balance between recall and fall-out such that we are catching as many phishing URLs as possible whilst minimising false positives. By measuring the area under the ROC curve (AUC), we can place a value on this trade-off, as a larger AUC means that the recall is closer to 1.0 and the false positive rate closer to 0.0. The AUC can be estimated using the following expression (Powers, 2011; Cortes and Mohri, 2004):

$$\texttt{AUC} = \frac{\texttt{tpr} - \texttt{fpr} + 1}{2}$$

The performance measures discussed above will be used extensively in the following sections, as we discuss parameter optimisation and the empirical evaluation of our feature sets. All of the experiments detailed within this chapter were performed on an Intel i7-6560U @ 2.20GHz using a single thread.

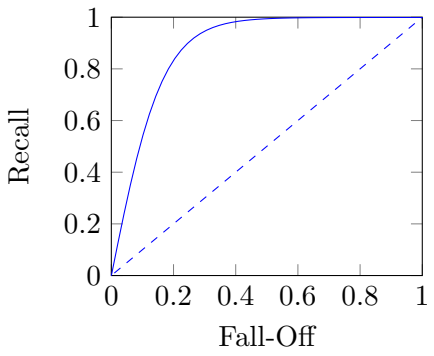


Figure 7.2: An example ROC curve. The dotted line represents the expected curve of a random classifier.

7.2.2 Optimising Model Parameters

Before we can use our classifier in our empirical experimentation, we need to select sensible values for each of the classifier's parameters. The online random forest classifier that we have implemented has six different parameters, shown below.

δ	Split confidence	The maximum probability that our decision to split on a particular feature is incorrect. Lower values are more likely to produce an accurate model, at the risk of increased computational expense.
τ	Tie confidence	The minimum difference in information gain considered to be significant. Lower values will lead to an increase in the accumulation of examples at inseparable nodes, potentially wasting computational time.
n_{min}	Split threshold	The minimum number of examples required at a node before it will be considered for splitting. Lower values will result in a deeper, potentially more accurate, model but will also increase computational complexity.
μ	Inclusion bound	The proportion of features to be considered when splitting a node. Higher values will increase the similarity between trees within the random forest.
ρ	Confidence bound	The minimum confidence required to classify a URL as phishing. Lower values will decrease precision in favour of increased recall, and vice-versa.
N	Ensemble size	The number of trees within the random forest. Higher values are likely to lead to a more accurate model, but will also increase resource usage.

The problem of parameter optimisation is combinatorially complex, and it is often unclear as to which combination of values will provide the best performance (Domingos, 2012). This is confounded further by the possible existence of multiple sets of optimum values (Domingos, 2012). A wide variety of optimisation methods exist, each of varying levels of complexity. Popular combinatorial optimisation techniques include greedy search (Feo and Resende, 1995), beam search (Ponte, Paquete and Figueira, 2012) and branch-and-bound (Domingos, 2012; Feo and Resende, 1995). Continuous methods of optimisation have also been explored, with examples including gradient descent (Domingos, 2012) and Quasi-Newton methods (Gill and Murray, 1972).

Due to time constraints, we employ a somewhat naive grid-search technique to find suitable values for five out of our six parameters. First, we choose initial values for each parameter—informed by recommendations from the surrounding literature (see "Default Value" in table 7.3). Once our initial values are chosen, the value of each parameter is varied and the AUC and rates of training and classification recorded as our performance measures. Enumerating all of the possible combinations of values for all five parameters simultaneously proves to be too time consuming for our project time-scale, so instead we vary the value of a single parameter at a time, fixing the value once an optimum configuration is reached. Values for

Parameter	Default Value	Starting Value	Maximum Value	Increment Step
Split threshold	100	1	20,000	$+5,000$
Split confidence	0.01	0.000001	0.1	$\times 10$
Tie confidence	0.1	0.0	0.5	$+0.1$
Inclusion bound	0.5	0.1	1.0	$+0.1$
Ensemble size	12	1	200	$+50$

Table 7.3: The default values and ranges for each parameter in our broad grid-search. The search is performed row by row, from top to bottom

.

each parameter are initially varied on a broad scale so as to provide a general indication of where the optimal value is located. Table 7.3 shows the ranges of values used for the broad grid search on each parameter, and the order in which we perform the search.

In order to perform the grid search optimisation, we train a random Hoeffding forest (see section 6.6.2) using the existing simple lexical features identified in table A.1. As we wish to later assess how well our model generalises to unseen data, the optimisation step uses only the training data, split into a training set and a validation set. This training set consists of 70% of the training data from April 2016 (456,172 URLs), and the validation set formed from the remaining 30% of the training data from the same month (195,502 URLs). In dividing the training data into training and validation sets, we ensure that the testing data remains completely unseen—preventing over-fitting to the testing data. This is important in ensuring that the results obtained from the use of the testing set are truly representative of the generalised case.

Split Threshold

To optimise the split threshold, we first perform a broad search ranging from a value of 1 to a value of 20,000. The results (see table 7.4) indicate that as the value of the split threshold increases, the training rate increase from 7,191 to 25,771 but the AUC experiences a slight decrease from 0.92716 to 0.88682. This aligns with our expectations, as increasing the split threshold increases the number of examples required in order to consider splitting a node. An increased number of examples means that splitting is considered less frequently, leading to an increased training rate, however, the resulting tree will have fewer nodes and is therefore less likely to reflect the data accurately.

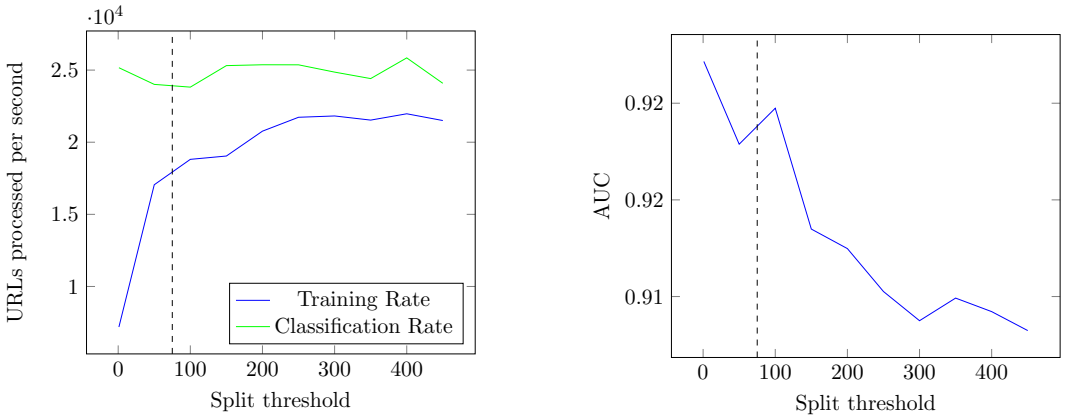
In order to find a suitable trade-off between processing rate and accuracy, we perform a more focussed search on values for the split threshold in the range of 1 to 450 (see table 7.5). Figure 7.3a confirms the positive correlation between training rate and the split threshold that was observed in the broad search, increasing sharply between a split threshold of 1 and 50, and then levelling off between 200 and 450. We also confirm a negative correlation between AUC and the split threshold in figure 7.3b, decreasing from 0.92716 to 0.91324. Given these results, we select 75 as our optimum value for split threshold as it is high enough that the increase in training rate is beginning to level off, whilst the AUC suffers only a minor penalty.

Split Threshold	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
1	**7,191**	**25,164**	**0.92716**
5,000	**25,295**	**26,316**	**0.89968**
10,000	25,444	24,444	0.88834
15,000	26,622	27,107	0.88575
20,000	25,771	27,350	0.88682

Table 7.4: Broad grid search optimisation results for varying values of n_{min}

Split Threshold	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
1	7,191	25,164	0.92716
50	**17,051**	**24,009**	**0.92288**
100	**18,812**	**23,819**	**0.92474**
150	19,043	25,310	0.91849
200	20,767	25,369	0.91748
250	21,729	25,365	0.91526
300	21,820	24,860	0.91375
350	21,533	24,411	0.91492
400	21,971	25,846	0.91422
450	21,503	24,084	0.91324

Table 7.5: Focussed grid search optimisation results for varying values of n_{min}



(a) Processing rates over varying values of n_{min}

(b) AUC over varying values of n_{min}.

Figure 7.3: Processing rates and AUC over varying values of n_{min}.

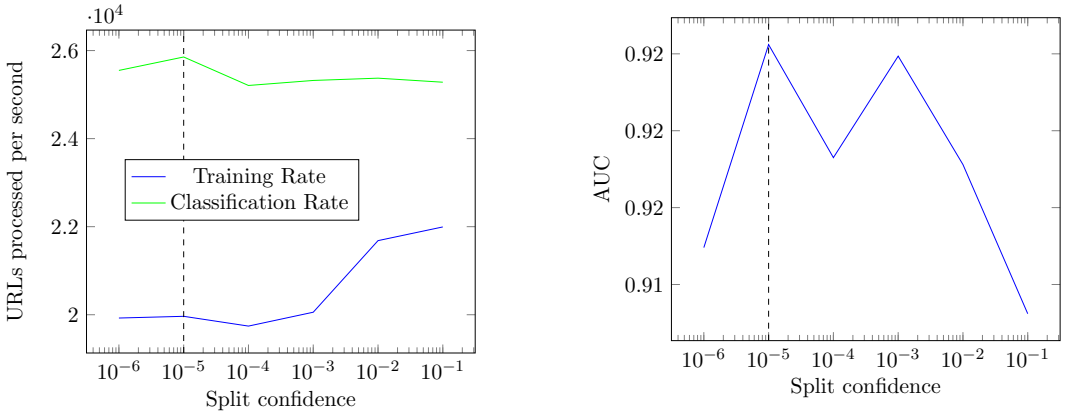
Split Confidence

To find a value for the split confidence, δ, we perform a grid search ranging from 0.000001 to 0.1. Table 7.6 shows the results of the search. We observe a minor increase in training rate from 19,925 as the value of δ increases, with little apparent change in the AUC. Larger values of δ require a lesser degree of confidence in the decision to split a node, and as such, decrease the computational cost of calculating the information gain of each feature because fewer examples need be considered. This leads to the minor increase in training rate observed in Figure 7.4a—though it is far less significant than the increase observed when changing the

split threshold. Due to the lack of any significant correlations, we selected a value of 0.00001 for δ as it provided the highest AUC of 0.91712 and the fastest classification rate of 25,854.

Split Confidence	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
0.000001	19,925	25,549	0.91448
0.00001	**19,964**	**25,854**	**0.91712**
0.0001	19,742	25,207	0.91565
0.001	20,057	25,321	0.91697
0.01	21,683	25,373	0.91556
0.1	21,994	25,281	0.91362

Table 7.6: Grid search optimisation results for varying values of δ



(a) Processing rates over varying values of δ

(b) AUC over varying values of δ.

Figure 7.4: Processing rates and AUC over varying values of δ.

Tie Confidence

To optimise the tie confidence, we first perform a broad grid search over values from 0.0 to 0.5 (see table 7.7). The results show an increase in training rate and AUC between 0.0 and 0.1. Beyond 0.1, we observe a decrease in the training rate from 20,473 to 14,918, whilst the AUC continues to increase slightly. As the tie confidence increases from 0.0 to 0.1, less time is spent deciding between attributes with similar information gains—therefore leading to an increase in the training rate. Between 0.1 to 0.2, the cost of splitting nodes more frequently begins to outweigh the benefit provided by the tie confidence, leading to the observed decrease in training rate. Splitting nodes more frequently results in a deeper tree, however, which is likely responsible for the observed increase in AUC.

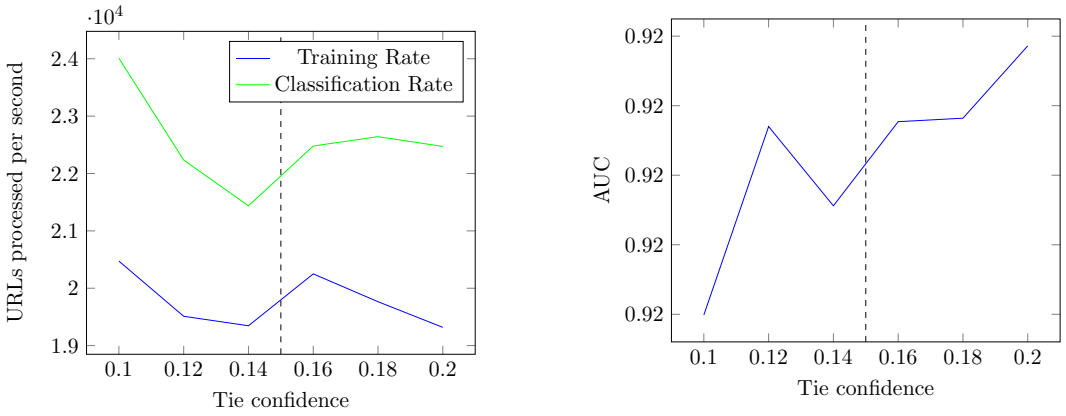
We then perform a more focussed search between 0.1 and 0.2, in an attempt to identify the point at which the cost of frequently splitting nodes outweighs the benefit of the tie confidence (see table 7.8). Figure 7.5a shows a decrease in both classification and training rates, with figure 7.5b confirming the increase in AUC observed earlier. Due to the inverse nature of the correlations observed for processing rates and AUC, we select the midpoint, 0.15, as our value for τ.

Tie Confidence	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
0.0	18,389	25,768	0.91592
0.1	**20,473**	**24,009**	**0.91999**
0.2	**19,318**	**22,472**	**0.92386**
0.3	15,751	20,798	0.92656
0.4	14,803	20,795	0.92657
0.5	14,918	21,674	0.92630

Table 7.7: Broad grid search optimisation results for varying values of τ

Tie Confidence	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
0.1	20,473	24,009	0.91999
0.12	19,511	22,235	0.92270
0.14	**19,346**	**21,437**	**0.92156**
0.16	**20,250**	**22,479**	**0.92277**
0.18	19,765	22,642	0.92282
0.2	19,318	22,472	0.92386

Table 7.8: Focussed grid search optimisation results for varying values of τ



(a) Processing rates over varying values of τ

(b) AUC over varying values of τ.

Figure 7.5: Processing rates and AUC over varying values of τ.

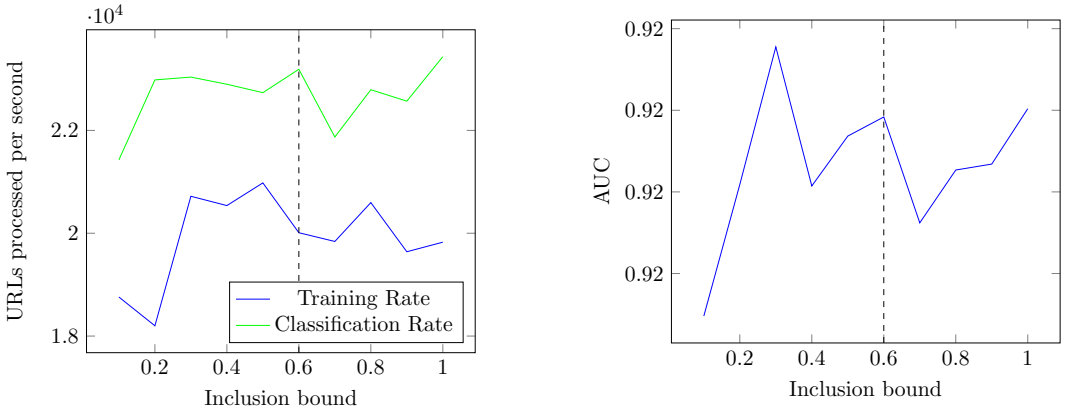
Inclusion Bound

To optimise the value of the inclusion bound, μ, we perform a grid search over values from 0.1 to 1.0 (see 7.9). Figure 7.6a shows that the processing rates appear to increase slightly with an increasing inclusion bound; which is at odds with our expectations. Given a larger inclusion bound, a greater number of features are considered when splitting a node. Our immediate assumption is that considering a larger subset of the features will increase the computational expense—leading to a decrease in the rate of processing. One possible explanation for the observed increase in the rate of processing is that by considering a larger subset of the features, highly discriminative features are selected more often, leading to fewer examples accumulating at each node. Fewer examples at each node means that calculating the information gain for each feature is less expensive, leading to an increase in processing speed. Due to the unexpected nature of these results, we select 0.6 as our value for μ, as it is one step above

from the midpoint of 0.5.

Inclusion Bound	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
0.1	18,759	21,427	0.920479
0.2	18,200	22,982	0.922091
0.3	20,719	23,037	0.923776
0.4	20,538	22,898	0.922073
0.5	20,979	22,734	0.922684
0.6	**20,010**	**23,189**	**0.922918**
0.7	19,840	21,872	0.921621
0.8	20,596	22,791	0.922268
0.9	19,640	22,569	0.922341
1.0	19,826	23,434	0.923019

Table 7.9: Grid search optimisation results for varying values of μ



(a) Processing rates over varying values of μ

(b) AUC over varying values of μ.

Figure 7.6: Processing rates and AUC over varying values of μ.

Ensemble Size

To select an optimum value for the ensemble size, we first conduct a broad grid search on values ranging from 1 to 200. The results (see table 7.10) show a strong negative correlation between the ensemble size and the training rate, with a decrease from 35,614 to 1,321 between 1 and 200. This negative correlation is also observed in the classification rate, decreasing from 29,424 to 2,137. We do see an increase in AUC with increasing ensemble size, however the effect is minimal.

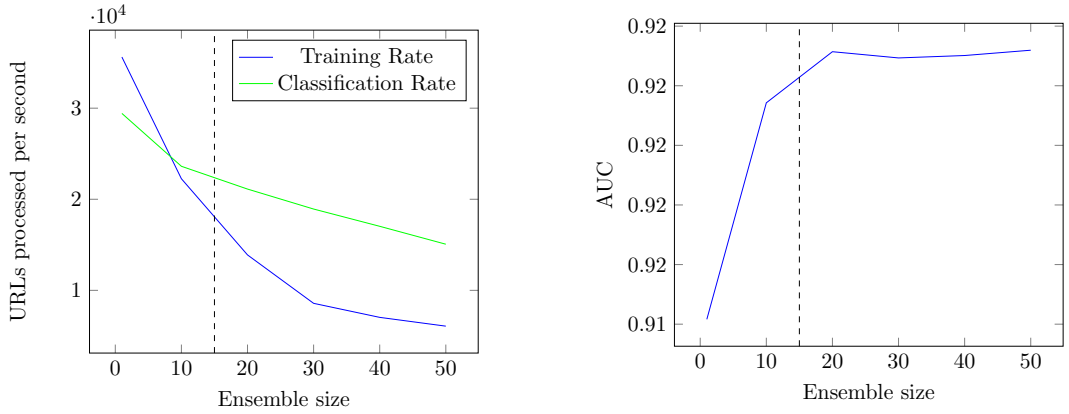
A more focussed grid search is then conducted between an ensemble size of 1 and 50 (see table 7.11). These results reflect the earlier observed decrease in training and classification rates (see figure 7.7a). From figure 7.7b we can see that AUC increases sharply between 0 and 20, before beginning to level off. We select 15 as our value for N because it is high enough to ensure that we gain the majority of the AUC increase, whilst minimising the cost to the rate of processing.

Ensemble Size	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
1	**35,614**	**29,424**	**0.914163**
50	**6,070**	**15,066**	**0.923192**
100	3,043	14,900	0.924001
150	1,835	3,118	0.923190
200	1,321	2,137	0.923759

Table 7.10: Broad grid search optimisation results for varying values of N

Ensemble Size	Training Rate (URL/s)	Classification Rate (URL/s)	AUC
1	35,614	29,424	0.914163
10	**22,253**	**23,621**	**0.921427**
20	**13,891**	**21,120**	**0.923142**
30	8,575	18,927	0.922933
40	7,034	17,032	0.923014
50	6,070	15,066	0.923192

Table 7.11: Focussed grid search optimisation results for varying values of N



(a) Processing rates over varying values of N

(b) AUC over varying values of N.

Figure 7.7: Processing rates and AUC over varying values of N.

Final Values

Table 7.12 summarises the optimum values selected through our optimisation process for each of the five parameters. These values are used throughout the remainder of our empirical evaluation.

Parameter	Value
Split threshold	75
Split confidence	0.00001
Tie confidence	0.15
Inclusion bound	0.6
Ensemble size	15

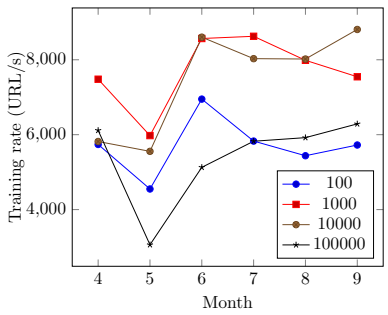
Table 7.12: Final values for each parameter after optimisation

7.2.3 Optimising Feature Extraction Parameters

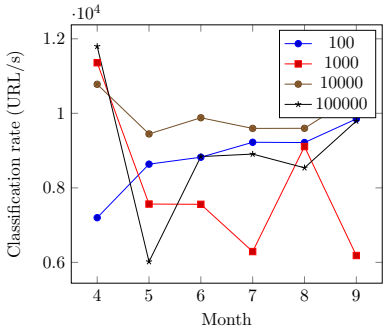
In section 5.3.2, we detailed our use of IDF and metric entropy to reduce the dimensionality of the dictionary for the situated feature sets. Because our classifier operates in an online fashion, our approach to dimensionality reduction operates over a buffer of terms—the size of which, we must decide upon. A larger term buffer gives us a greater sample of URLs, allowing us to identify the most representative terms, as well as wasting fewer resources on terms that are not interesting. The downside to using a larger buffer is that it is more difficult for terms to enter the dictionary, which could impede our ability to react to a change in phishing URL composition. We therefore need to find a value for the size of the term buffer that allows the dictionary to grow in response to changes in URL composition, without making the dictionary so large that it is unsustainable. Similarly to our model parameter optimisation in section 7.2.2, we employ a simple grid-search technique in order to achieve this.

Because we are interested in the rate of the dictionary growth over time—and its effect on performance—each term buffer size is tested on a monthly basis over the six months of data that we have available, rather than just a single month. Our training data for each month is split into 70% training and 30% validation, with training and validation occurring in a cumulative fashion so as to simulate a real-world deployment. During each simulated month, the average rate of training is recorded. Once the training data for the month has been processed, the validation set is used to calculate the AUC and the rate of classification. By simulating the operation of the classifier over an extended period of time, we can select an appropriate trade-off between the expected processing rates and the classification accuracy.

The size of the term buffer is varied from 100 to 100,000, with an incremental step of $\times 10$. Tables C.1, C.2, C.3 and C.4 show the results of the grid search. These results are summarised in figures 7.8a and 7.8b. Figure 7.8a indicates that the optimum rate of processing is achieved with a term buffer of between 1,000 and 10,000 items—with the rate of processing decreasing either side of this margin. It is likely that a buffer smaller than 1,000 items introduces too many useless terms into the dictionary—wasting time and resources—whilst a buffer larger than 10,000 becomes so large that the time spent calculating the IDF values outweighs the benefit of the dimensionality reduction. Figure 7.8b shows that a buffer size of 10,000 consistently produces the best classification rate for each month, and as a result, we selected 10,000 as our value for the term buffer size.



(a) Training rates over varying term buffer sizes



(b) Classification rates over varying term buffer sizes

Figure 7.8: Processing rates over varying term buffer sizes

7.3 Evaluating Feature Set Performance

7.3.1 Simple Feature Importance

In this section, we evaluate the discriminative power of each of the 25 existing simple lexical features identified in section 4.2, as well as each of the 23 novel simple lexical features that we proposed in section 4.3. In order to achieve this, we exploit the natural feature selection that occurs within our Hoeffding tree model. At each node in the tree, the feature providing the greatest information gain is selected to test upon (see section 6.3.2). By counting the number of nodes that select each particular feature, we can compare their discriminative power.

To compare the 25 existing simple lexical features, we train a single Hoeffding tree using the full 6 months of data within our training set in a cumulative fashion. Once the classifier has been trained, we take a histogram of the features selected to test on at each node in the model. These frequencies are then normalised to produce a proportional frequency for each of the simple features. Table C.5 shows the results of this process, with table 7.13 showing the top five most frequently selected features. The most discriminative features include the length of the hostname, the length of the path and the number of dots within the hostname. Other features, such as the total number of '.' characters in the URL, were not selected selected at all.

To evaluate the contribution of our novel simple features, we performed the same experiment described above, but this time training the model using both the 25 existing simple features and the 23 novel features that we have proposed. Table 7.14 shows the proportional frequencies of each feature, with our novel features highlighted in bold. 17 out of the 23 novel simple features were selected by the classifier, with 6 out of the top 10 selected features consisting of our features. In the presence of our novel features, the length of the hostname rose to become the most frequently selected feature, with the maximum numerical value in the hostname coming second. Table C.6 shows the features that were not selected by the classifier. We can see that a number of novel suggestions were not selected, including the number of '@' characters in the arguments, and the presence of a non key-value argument.

Whilst the discriminatory power of each feature will vary depending on the dataset, we believe that these results successfully demonstrate that the majority of the novel simple features proposed within our work improve upon the existing simple features suggested in previous literature. The ranking of each feature based on proportional frequency is something which, to the best of our knowledge, has never been attempted in the context of online phishing URL classification, and we believe that these results provide new evidence in favour of the selection of particular simple features—something which we believe to be useful to future research.

Simple Feature	Proportional Frequency
Longest token in hostname	0.16364
Hostname length	0.15303
Path length	0.14849
Longest directory in path	0.11666
Number of dots in hostname	0.07424

Table 7.13: The top 5 most frequently selected existing simple lexical features over 6 months

Simple Feature	Proportional Frequency
Length of the hostname	0.14404
Max. numerical value in the hostname	**0.11219**
Average directory length	**0.08311**
Length of the longest token in the hostname	0.06787
Length of the longest sub-domain in the hostname	**0.06510**
Path length / total URL length	**0.06233**
Proportion of digits within the path	**0.06094**
Length of the file name	0.05678
Number of hex characters in the hostname	**0.05401**
Length of the longest directory in the path	0.05401
Total length of the arguments	0.03601
Number of '.' characters in the hostname	0.03324
Number of tokens in the hostname (split on '.', '-', '_', '@')	0.01939
Length of the longest argument value	0.01523
Proportion of digits in the hostname	**0.01246**
Number of hex characters in the path	**0.01246**
Average argument value length	**0.01246**
Max. number of special characters in any one directory ('@', '-', '_', '~', ',')	0.01246
Max. number of '.' characters in any one directory	0.01108
Max. number of digits in any one directory	**0.00969**
Average argument key length	**0.00969**
Number of hex characters in the URL	0.00831
Number of '/' characters in the path	0.00831
Length of the path	0.00692
Number of digits in the path	**0.00554**
HTTPS present	0.00415
Number of special characters in the hostname ('@', '-', '_')	0.00415
Max. number of digits in any one sub-domain	**0.00277**
Number of special characters in the path ('@', '-', '_', '~', ',')	0.00277
Number of special characters in the file name ('-', '_')	0.00277
Total number of arguments	0.00277
Length of the shortest argument value	**0.00138**
Length of the longest argument key	**0.00138**
Number of hex characters in the arguments	**0.00138**
Number of digits in the hostname	**0.00138**
Total URL length	0.00138

Table 7.14: A table showing the normalised proportion of nodes testing on each simple feature within a Hoeffding tree trained over 6 months of data using both the existing simple lexical features and our novel simple lexical features. Features proposed within our work are emboldened. See table C.6 for the features not present within the tree.

7.3.2 Cumulative Error

In section 6.3, it was suggested that tree-based models fail to respond well to changes in the distribution being modelled. This poses a potential problem in our case, as the lexical contents of phishing URLs are likely to change over time as fraudsters develop new techniques, or as new target organisations appear. We seek to mitigate this problem through the use of an online classifier, as it allows the model to update in real-time. It was acknowledged, however, that this is still not a perfect solution. As a result, it is important that we evaluate the performance of our model in a way that reflects the possibility of change within the modelled distribution.

In order to account for change in the modelled distribution, we attempt to simulate a real-world deployment using the data described in section 7.1.2. This is done by constructing the classification model in a continuous fashion, using each of the six monthly training sets in chronological order. At the end of each simulated month of training, the cumulative error rate is calculated as a measure of the performance over time. The cumulative error rate, e, for some time, t, is defined by Ma et al. (2009) as the proportion of misclassified examples for all URLs encountered up until that time. This can be expressed as follows:

$$e_t = \frac{\sum \mathtt{fp}_t + \sum \mathtt{fn}_t}{\sum \mathtt{tp}_t + \sum \mathtt{tn}_t + \sum \mathtt{fp}_t + \sum \mathtt{fn}_t}$$

The cumulative error rate is a useful statistic for us as an increasing error rate will indicate that our model is failing to adapt to changes in the modelled distribution.

7.3.3 Feature Set Performance

Following our performance evaluation of individual simple features in section 7.3.1, we move on to consider the performance of various lexical feature sets. In the following sections, we empirically evaluate the performance of six different lexical feature sets from our work.

First we consider two simple lexical feature sets (see chapter 4). The first feature set we consider contains only the 25 simple features identified from existing research (see table A.1). The second feature set combines the 25 existing simple features with our 23 novel features defined in section 4.3 (see table 4.1).

Following this, we consider two situated lexical feature sets (see chapter 5). Here, we examine the performance of a uni-gram bag of words feature set, as well as a bi-gram bag of words feature set.

Finally, we look at combining simple and situated lexical features into two composite feature models. Here, we select the top 24 simple features identified during our evaluation of individual features in section 7.3.1 (see table 7.14) and combine these with both a uni-gram and a bi-gram feature set.

Each feature set is evaluated using a random Hoeffding forest trained in a cumulative fashion over the full 6 months of training data and using the parameter values identified from our previous optimisation steps (see sections 7.2.2 and 7.2.3). During training for each month, the average training rate is measured, and at the end of each month the AUC, average classification rate and cumulative error are calculated using the corresponding testing data.

Simple Lexical Features

In section 4.2, we identified 25 simple lexical features from existing literature. Following this, in section 4.3, we proposed a further 23 novel simple lexical features, bringing the total number of simple features to 54. In order to support our hypothesis, we must demonstrate that lexical features are sufficiently discriminative and that their use improves the rate of processing compared to content-based feature sets. In section 7.3.1, we provided evidence to suggest that a number of our novel features are more discriminative than the majority of the simple features proposed in previous work. Now, we investigate the effects of our novel simple features on the performance of our classifier.

Table 7.15 shows the performance statistics for the existing simple lexical feature set over 6 months. An average training rate of 20,160 URL/s is observed, along with an average classification rate of 22,541 URL/s. This is over 1.7 million times faster than the content based approach documented in the work of Whittaker, Ryner and Nazif (2010). A maximum precision of 0.95259 was observed, with a recall of 0.97244. Whilst this is not as high as previous content based approaches, we are not aware of any previous research that uses a feature set derived solely from simple lexical features. Furthermore, the observed precision and recall is comparable to much of the existing literature, despite the use of only simple lexical features. For example, Le, Markopoulou and Faloutsos (2011) were able to achieve a 98% accuracy using a combination of both simple and situated lexical features and a confidence weighted approach—whereas our random Hoeffding tree achieves around a 95% accuracy using simple features alone.

Table 7.16 shows the performance statistics of our classifier using a combined feature set containing both the existing simple features and our novel suggestions. Precision and recall appear unaffected, however the increased number of features reduces the average training and classification rates from 20,160 and 22,541 to 14,773 and 15,622 respectively. The lack of a change in predictor performance suggests that the existing simple features are suitably discriminative given our data set. It is likely that a more ambiguous data set would be able to separate the performance of these feature sets, however, given our current results it appears as though the novel features provide little benefit to the overall performance of the classifier.

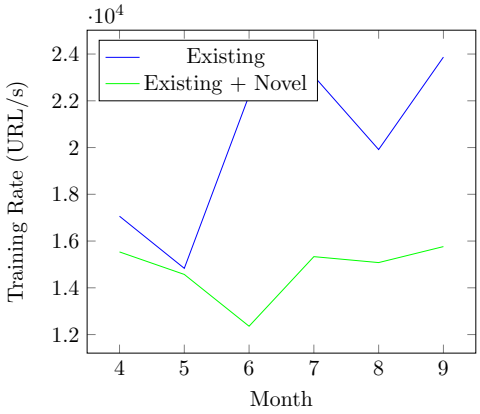
Figure 7.9d shows the cumulative error of the classifier for each feature set over the course of the 6 months. We can see that the cumulative error decreases each month for both feature sets—suggesting that the feature sets are resistant to changes in the modelled distribution.

Month	Tr. URL/s	Class. URL/s	AUC	Precision	Recall	Cum. Error
April	17,062	22,856	0.91311	0.95583	0.86593	0.08669
May	14,833	21,009	0.93129	0.96465	0.89505	0.07493
June	22,229	21,494	0.93205	0.96376	0.89734	0.07235
July	23,055	21,581	0.93011	0.96413	0.89289	0.07165
August	19,915	23,233	0.93879	0.96723	0.90813	0.06933
September	23,869	25,077	0.95259	0.97244	0.93142	0.06526

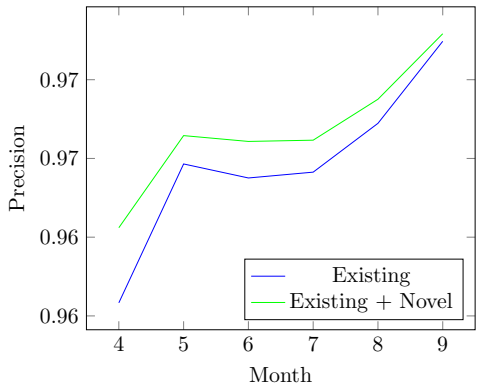
Table 7.15: Performance statistics over time with existing simple features

Month	Tr. URL/s	Class. URL/s	AUC	Precision	Recall	Cum. Error
April	15,535	15,803	0.91612	0.96060	0.86754	0.08369
May	14,572	14,493	0.93422	0.96645	0.89934	0.07198
June	12,357	15,495	0.93336	0.96608	0.89776	0.06998
July	15,333	15,419	0.92851	0.96616	0.88760	0.07028
August	15,079	15,782	0.93978	0.96876	0.90865	0.06804
September	15,763	16,741	0.95420	0.97292	0.93425	0.06391

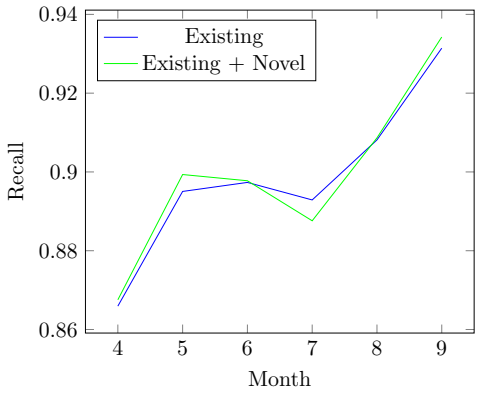
Table 7.16: Performance statistics over time with existing + novel simple features



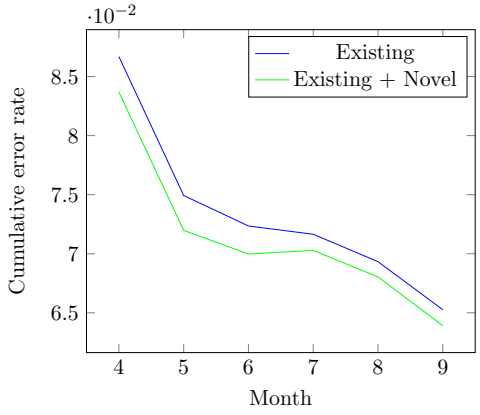
(a) Training rate over time with simple lexical features



(b) Predictor precision over time with simple lexical features



(c) Predictor recall over time with simple lexical features



(d) Cumulative error over time with simple lexical features

Figure 7.9: Performance statistics over time with simple lexical features

Month	Tr. URL/s	Class. URL/s	AUC	Precision	Recall	Cum. Error
April	5,495	10,545	0.89946	0.97736	0.81773	0.10021
May	5,240	9,043	0.91828	0.98498	0.84939	0.08801
June	7,878	9,527	0.90012	0.98240	0.81463	0.09201
July	7,290	8,698	0.88820	0.98275	0.79004	0.09723
August	6,873	8,693	0.87602	0.98206	0.76593	0.10302
September	7,456	8,955	0.86544	0.98156	0.74480	0.10880

Table 7.17: Performance statistics over time with uni-gram features

Situated Lexical Features

In chapter 5, we defined a situated bag of words approach using n-gram tokens from the URL. We have previously suggested that the use of situated features is likely to construct a more accurate model than simple features, but that it comes with an increased computational cost. In order to validate this claim, we evaluate the performance of the classifier when using both uni-gram and bi-gram situated feature sets.

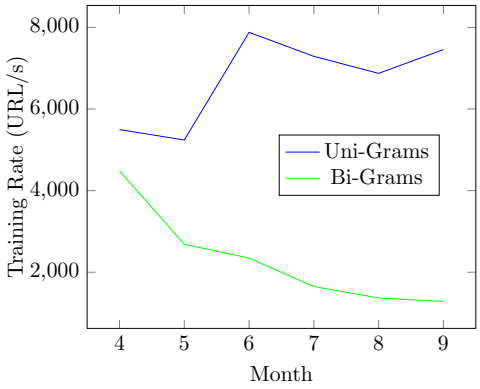
Table 7.17 shows the performance statistics of our classifier when using a uni-gram situated feature set. An average training rate of 6,705 URL/s and an average classification rate of 9,243 URL/s are observed, around three times slower than the rates observed with simple lexical features. A maximum precision of 0.98156 is achieved, along with a recall of 0.74480. This precision is higher than that observed with the simple lexical features, suggesting that the use of uni-gram features provides fewer false positives. These results are akin the work of Khonji, Iraqi and Jones (2011), where a precision of 0.97 and a recall of 0.64 observed using a similar token-based approach but without our positional encoding.

Table 7.18 shows the performance statistics for our bi-gram situated feature set. The average training and classification rates of 2,302 URL/s and 2,438 URL/s observed with the bi-gram feature set are lower than those observed with the uni-gram features. This is expected, as the increased number of n-grams increases the dimensionality of the dictionary—increasing the database query times and slowing down processing. The disparity in processing speeds between uni-gram and bi-gram features is illustrated in figure 7.10a This increase in dimensionality does result in an increase in precision and recall, however, from 0.98156 to 0.99157 and 0.74480 to 0.84879 respectively. Even at a classification rate of 927 URL/s, the bi-gram feature set is still 70,452 times faster than the content-based approach detailed in the work of Whittaker, Ryner and Nazif (2010)—a significant improvement.

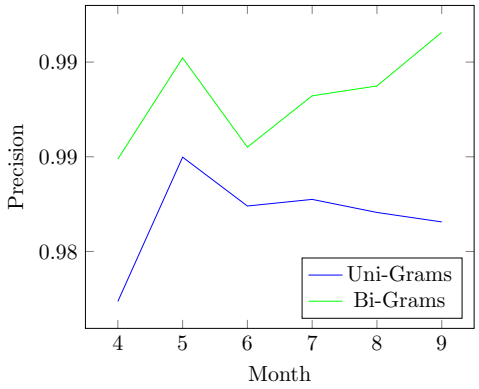
Figure 7.10c shows the difference in recall over time between the uni-gram and bi-gram feature sets. Interestingly, the recall of the uni-gram feature set appears to decrease over time, whilst the recall of the bi-gram feature set increases. This suggests that bi-gram features are more robust to changes in the modelled distribution. We also see this reflected in the cumulative error for each feature set (see figure 7.10d). The cumulative error of the classifier using the uni-gram feature set decreases from 0.10021 to 0.09201 from April to June, but then begins to rise back to 0.10880 from June to September. On the other hand, the cumulative error observed with the bi-gram feature set consistently decreases over time from 0.15351 to 0.10648. Whilst this is higher than the cumulative error observed with the uni-gram features, the continual decline in cumulative error supports the theory that the bi-gram feature set is more resistant to change than uni-grams—supporting the work of (Baykan et al., 2009).

Month	Tr. URL/s	Class. URL/s	AUC	Precision	Recall	Cum. Error
April	4,483	6,333	0.84592	0.98488	0.70255	0.15351
May	2,684	3,200	0.88831	0.99022	0.78428	0.12611
June	2,347	1,835	0.89016	0.98551	0.79179	0.12005
July	1,649	1,325	0.89059	0.98822	0.79045	0.11715
August	1,368	1,008	0.90267	0.98874	0.81455	0.11273
September	1,283	927	0.92080	0.99157	0.84879	0.10648

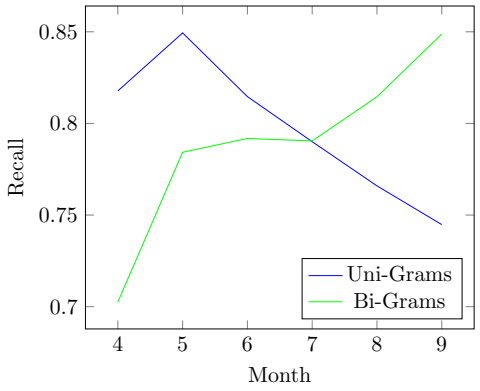
Table 7.18: Performance statistics over time with bi-gram features



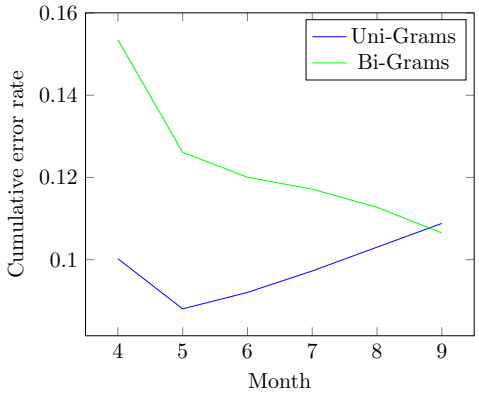
(a) Training rate over time with situated lexical features



(b) Predictor precision over time with situated lexical features



(c) Predictor recall over time with situated lexical features



(d) Cumulative error over time with situated lexical features

Figure 7.10: Performance statistics over time with situated lexical features

Composite Lexical Features

We have now evaluated the performance of both simple and situated lexical features. We found simple lexical features to be faster than situated lexical features, with a higher recall. Situated features were shown to have a higher precision than simple features, but were found to be lacking when it comes to recall. Now, we consider combining these feature sets in order to investigate whether we can maintain the high precision of the situated feature sets, whilst improving recall. In order to achieve this, we select the top 24 simple features identified during our evaluation of individual features in section 7.3.1 (see table 7.14) and combine

these with both a uni-gram and a bi-gram feature set. From hereon in, we refer to these feature sets as uni-gram and bi-gram composite lexical feature sets.

Table 7.19 shows the performance statistics for the uni-gram composite feature set. Average rates of training and classification of 6,051 URLs/ and 8,661 URL/s are observed—which appears faster than the rates observed with uni-grams alone. This is counter-intuitive, as the addition of further features to the training set would increase the processing time for each URL. It is likely that the inclusion of the simple features allows fewer examples to accumulate at each node, reducing the time taken to identify the feature to test on. This suggests that our method of positional encoding is perhaps not as discriminative as the simple features in the general case. A maximum precision of 0.98106 is observed, which appears unaffected when compared to the 0.98156 observed with uni-gram features alone. However, with the addition of simple features, the composite uni-gram feature set achieves a recall of 0.93594, which is a significant improvement over the recall of 0.74480 observed with uni-gram features alone.

Table 7.20 shows the performance statistics of the bi-gram composite feature set. An average rate of training of 4,625 URL/s is observed, along with an average classification rate of 8,144 URL/s. These processing rates are slower than the rates observed with uni-gram composite features, however, once more we see an improvement over the rates observed with bi-gram features alone. The observed precision of 0.97464 is slightly lower than the 0.99157 seen with bi-gram features alone, however, we see a significant increase in precision from 0.84879 to 0.93375.

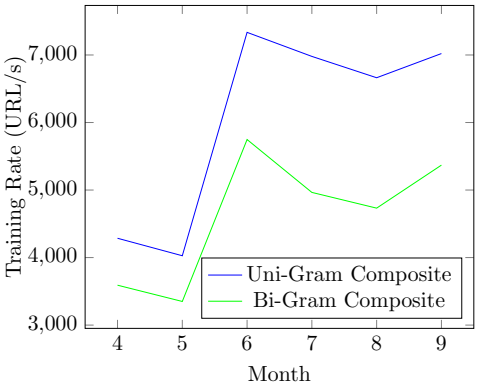
In figure 7.11d, we see the cumulative error rates over time for both the uni-gram and bi-gram composite feature sets. Interestingly, we see a consistent month by month decrease in the cumulative error for both feature sets. With situated features alone, we observed an increase in cumulative error for the uni-gram feature set. This suggests that the addition of the simple features makes the uni-gram feature set more resilient to a changing distribution. Figure 7.11d also shows us that the cumulative error is consistently higher for the bi-gram composite features, implying that the uni-gram composite features are now more expressive than the bi-gram composite features.

Month	Tr. URL/s	Class. URL/s	AUC	Precision	Recall	Cum. Error
April	4,286	8,856	0.92951	0.96847	0.88769	0.07032
May	4,027	8,577	0.94408	0.97659	0.90975	0.06089
June	7,335	8,805	0.94310	0.97622	0.90798	0.05937
July	6,979	8,467	0.93798	0.97560	0.89805	0.05995
August	6,663	8,572	0.94684	0.97810	0.91400	0.05843
September	7,021	8,691	0.95898	0.98106	0.93594	0.05519

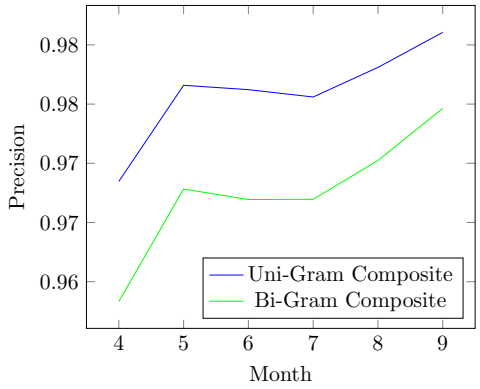
Table 7.19: Performance statistics over time with uni-gram composite features

Month	Tr. URL/s	Class. URL/s	AUC	Precision	Recall	Cum. Error
April	3,591	8,601	0.91702	0.95833	0.87164	0.08279
May	3,350	8,004	0.93612	0.96783	0.90191	0.07044
June	5,748	5,234	0.93566	0.96694	0.90168	0.06818
July	4,965	8,319	0.93361	0.96696	0.89737	0.06765
August	4,732	9,242	0.94225	0.97024	0.91229	0.06545
September	5,369	9,465	0.95480	0.97464	0.93375	0.06169

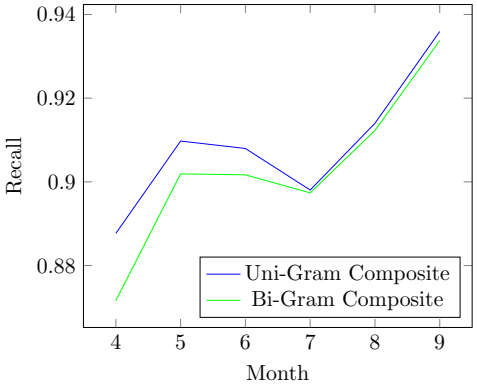
Table 7.20: Performance statistics over time with bi-gram composite features



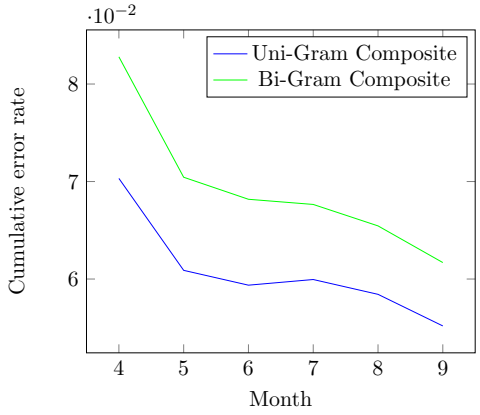
(a) Training rate over time with composite feature sets



(b) Predictor precision over time with composite feature sets



(c) Predictor recall over time with composite feature sets



(d) Cumulative error over time with situated lexical features

Figure 7.11: Performance statistics over time with composite lexical features

Comparing Feature Sets

Our hypothesis can be decomposed into two separate statements:

1. The use of lexical features alone will increase the rate of processing in comparison to the use of content-based features.

2. Lexical features are sufficiently discriminative that they may be leveraged to produce a predictor performance equal to that of a content-based approach.

Our results show that for every lexical feature set considered, a significant increase in processing rates is observed. In the work of (Whittaker, Ryner and Nazif, 2010), the mean classification rate for their content-based approach was reported as 1 URL every 76 seconds. Even with the slowest of our lexical feature sets—the situated bi-gram feature set—we observe an average processing rate that is over 185,000 times faster than the content-based approach. We can therefore discard the null hypothesis that the use of lexical features has no significant effect on the rate of processing.

Table 7.21 provides a summary of the performance statistics for each of the lexical feature sets considered within our work. We can see that simple lexical features provide the fastest processing rates by a margin of at least 8,300 URL/s, obtaining respectable predictor performances. Situated feature sets are the slowest of the feature sets that we consider, however, they achieve a higher precision than simple lexical features at the expense of recall. Uni-grams are significantly faster than bi-grams, however, we have shown that bi-grams are more robust to changing data—though the observed cumulative error suggests that they are not as robust as the simple features. Combining simple lexical features with situated features is shown to improve the processing rates of situated features, as well as improving recall. Using a composite feature set of uni-gram segments and 24 simple lexical features, we achieved a maximum precision of 0.98106 and a recall of 0.93594. This rivals the use of host-based features in the work of Garera et al. (2007), who obtained a precision of 0.958 and a recall of 0.988. Our precision of 0.98106 is also higher than the content-based approach proposed in the work of Xiang et al. (2011), however, our recall of 0.93594 is lower than the 0.99593 reported in their work. We therefore fail to demonstrate that simple lexical features alone can be used to produce a prediction performance equal to that of a content-based approach—though our results provide evidence to suggest that this may yet be possible.

Despite our failure to reproduce the recall observed in previous content-based approaches, we believe that our results still provide a number of useful innovations.

Our investigation into the importance of the individual simple lexical features in 7.3.1 provides a novel insight into the relative power of all of the simple features described in previous work. We believe this to be a useful contribution as it would help to inform the selection of simple lexical features in future work.

We have proposed a total of 23 novel simple features, the majority of which are demonstrably more discriminative than the simple features detailed in previous work. Given a more ambiguous data set, we believe that these novel features could be exploited to improve the predictor performance in future studies.

Our work on situated features in chapter 5 describes a novel positional encoding system, as well as detailing a number of simple methods for performing dimensionality reduction in an online context. Whilst the contribution of these is fairly minor, we believe that our methodology forms a sufficiently solid foundation for future study in these areas.

In chapter 6, we detailed the implementation of a novel online random forest classifier, known as the random Hoeffding tree. We described a number of optimisations which may prove useful in future work, such as the handling of variable-length feature vectors. It has been demonstrated that our classifier can operate at high speeds with accurate predictor performance, whilst coping well with noisy data and a changing distribution. We therefore believe that the described classifier has the potential for application anywhere within the broad field of online learning.

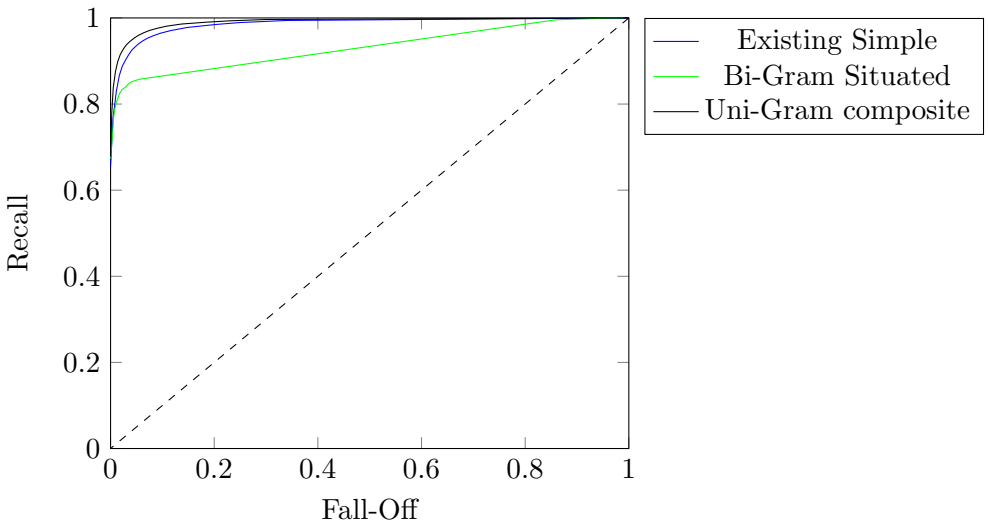


Figure 7.12: ROC curves for the existing simple features, the situated bi-gram features and the composite uni-gram features. The dotted line represents the expected curve of a random classifier.

Concerning phishing URL classification more specifically, every single feature set described in our work performs orders of magnitude faster than current content-based approaches, whilst maintaining a high precision and recall. We believe that our work could be used to significantly reduce the amount of URLs that have to be examined using current content-based approaches. By using our system to filter the stream of URLs being submitted to a content-based evaluation system, the majority of phishing and benign sites could be identified and removed from the stream without having to pay the cost of fetching the content of the URL. This would significantly increase the rate of processing in the general case, particularly for large-scale systems. By increasing the rate at which the majority of URLs are processed, blacklist-based systems could be updated faster and as a result, a greater number of users could be protected from the threat of phishing.

Feature Set	Tr. URL/s	Cl. URL/s	AUC	Precision	Recall	Cum. Error
Existing Simple	23,869	25,077	0.95259	0.97244	0.93142	0.06526
Existing + Novel Simple	15,763	16,741	0.95420	0.97292	0.93425	0.06391
Situated Uni-Grams	7,456	8,955	0.86544	0.98156	0.74480	0.10880
Situated Bi-Grams	1,283	927	0.92080	0.99157	0.84879	0.10648
Uni-Gram Composite	7,021	8,691	0.95898	0.98106	0.93594	0.05519
Bi-Gram composite	5,369	9,465	0.95480	0.97464	0.93375	0.06169

Table 7.21: A summary of the performance statistics for all of the lexical feature sets considered

Chapter 8

Conclusions

Contents

8.1 Aim Revisited

In section 1.4, we defined the aim of our project:

> The aim of this dissertation is to investigate the feasibility of purely lexical feature sets in the context of phishing URL classification in an online setting, and to provide novel recommendations on both suitable classifiers and feature extraction methods through empirical evaluation using a large, real-world dataset.

Chapter 3 described lexical features, and identified two classes of lexical features, simple features and situated features. In chapters 4 and 5, we considered simple and situated lexical feature sets in greater detail. We provided 23 novel simple features, and described a novel method of positional encoding in situated feature sets. In addition, we also discussed the problem of dimensionality reduction, and presented a number of simple solutions. In chapter 6, we detailed the implementation of a novel online random forest classifier, the online random Hoeffding tree. In chapter 7, we evaluated the performance of our various lexical feature sets using the proposed classifier over a data set of 7.5 million URLs spanning 6 months. We then gave a critique and comparison of each of the proposed feature sets, culminating in the recommendation of a particular methodology. During this critique, we successfully demonstrated the feasibility of lexical feature sets in comparison to content-based approaches—concluding that they are much faster at the expense of recall, but still a viable option for stream pre-processing. In this way, we believe that we have successfully met the aim that we defined for this project.

8.2 Objectives Revisited

In section 1.5, we identified the following five project objectives:

I. **Identification of existing feature extraction methods**. The existing work regarding the classification of phishing URLs will examined to provide an overview of the current sources and methods of feature extraction, as well as the classification mechanisms used.

II. **Critique and extension of existing lexical feature sets**. Once the existing literature has been examined, critique of the different feature sets and experimental designs will be provided. This critique will then inform the proposal of at least one novel lexical feature set.

III. **Implementation of an online classifier**. Research into online classification methods will be conducted. At least one online classification method will be selected for implementation. The selected classification method(s) must not have been applied to the task of phishing URL classification in previous work. The selected method(s) will be implemented, the details of which shall be documented.

IV. **Experimental design**. An experiment will be designed to investigate the hypothesis. This experiment will use a large, real-world dataset and shall evaluate the performance of the selected classifier(s) over time with varying feature sets.

V. **Evaluation of the proposed feature sets**. The results from the experiment designed in objective IV will be used to critically evaluate the proposed feature sets with respect to existing methodologies—providing empirical evidence to support or challenge my hypothesis.

Throughout our literature survey—in chapter 2—we discussed the different methods of feature extraction that have been applied to the classification of phishing websites. Such methods included the use of host-based features (Garera et al., 2007), content-based features (Xiang et al., 2011), third-party heuristics (Nguyen et al., 2014) and a variety of lexical features (Baykan et al., 2009; Khonji, Iraqi and Jones, 2011). The cost of performing network requests in the extraction of host-based, content-based and third party heuristics was criticised for potentially being unnecessary, with lexical features being offered as a potential solution. In doing so, we successfully achieve our first objective: the identification of existing feature extraction methods.

In chapter 3, we discussed lexical features in greater detail, and defined twelve common characteristics of phishing URLs. We divided lexical features into two categories: simple lexical features, and situated lexical features, each discussed in further detail in chapters 4 and 5 respectively. We identified twenty five simple lexical features from existing literature, comparing the motivation behind the selection of each feature with our twelve phishing URL characteristics. From this, we were able to identify a number of characteristics with few or no applicable simple features. Using this information, we proposed a further twenty three simple features to identify the characteristics that were less well-covered. We identified a subset of these characteristics, namely suspicious domains and domains provided by site builders, which could not be recognised with simple features alone due to the lack of an internal model. In order to identify these characteristics, we turned our focus towards situated lexical features. We considered n-gram models in particular detail, describing a novel method of positional encoding for tokens extracted from the URL. At this point, we also identified high dimensionality as a limitation of this approach, and developed a simple method of dimensionality reduction using IDF and metric entropy. In doing so, we provide both a critical evaluation and an extension of existing lexical feature extraction methods, achieving our second project objective: the critique and extension of the existing lexical feature sets.

In chapter 6, we formalised our concept of online learning. We identified decision trees as an accurate—but most importantly—fast model for binary classification, and detailed the theory behind an online decision tree using the Hoeffding bound. A previous implementation of the Hoeffding tree—known as VFDT—was discussed, and we identified a number of optimisations that we transfer through into our own work. Following this, we discussed a number of features specific to our implementation, before finally proposing an online ensemble classifier: the random Hoeffding forest. Thus, we complete objective III: the implementation of an online classifier.

In chapter 7, we performed an empirical evaluation of the various feature sets identified within our work. We detailed a number of experiments designed to optimise the parameters of the classifier, evaluate the contribution of our novel simple features, and compare the performance of each of our feature sets. In doing so, we achieve our fourth project objective: experimental design. Following this, we executed our experimentation. We identified the most discriminative simple features from existing literature, before demonstrating how our novel suggestions from chapter 4 contribute to enriching the simple feature set—our results giving the first ever empirical evidence in support of each specific simple lexical feature. We then

compared the performance of six different lexical feature sets. The results showed that simple lexical features were the fastest, but that situated features were more precise. By combining the simple and situated features into composite feature sets, we were able to maintain a high precision whilst also improving the recall of the classifier. This experimentation completes the last of our project objectives: the evaluation of the proposed feature sets. Thus, we successfully meet all of the project objectives that have been defined.

8.3 Future Work

8.3.1 Dimensionality Reduction

In section 5.3.2, we discuss the use of IDF and metric entropy as methods of reducing the dimensionality of our dictionary of terms. Whilst this practice is simplistic, our empirical evaluation demonstrated that it was effective in most cases. However, in the case of the uni-gram situated feature set, we saw an increase in the cumulative error rate of the classifier between the months of June and September. Whilst it is possible that this increase is simply a result of our use of a tree-based classifier, the fact that it was not observed with any of the other feature sets suggests that the dictionary was failing to respond to the changing lexical contents of the URLs over time. One possible solution to this problem may be to employ Latent Dirichlet Allocation (LDA). Latent Dirichlet Allocation uses the Dirichlet distribution to cluster observations into unlabelled topics (Blei, Ng and Jordan, 2002). Online variations of LDA exist (Hoffman, Bach and Blei, 2010), which could be utilised to identify topics within the n-gram segments. Each topic would then form an element of the feature vector. As the number of topics is less than the number of n-gram segments, the dimensionality of the feature vectors would be greatly reduced. Unfortunately, due to time constraints, this option could not be considered as part of our project—leaving the application of online LDA open for future research.

8.3.2 Data Collection

Our work considers the identification of phishing URLs in comparison to benign URLs. As in previous work, our benign URLs were collected from a web crawl of DMOZ, a peer-reviewed directory of websites. This was done in order to minimise the possibility of introducing noise into our data set. However, in a real-world deployment of our system, submitted URLs are likely to be more suspicious in nature than the URLs collected from DMOZ. As a result, it is unclear as to whether our evaluation truly represents the expected performance of the system over data submitted to anti-phishing organisations. Ideally, further research with the aid of such an organisation would be required to fully examine the performance of our system.

In section 7.3.1, we failed to identify any significant improvement in predictor performance when using our novel simple feature set compared to the existing feature set—despite earlier indications that a number of the novel features were more discriminative. We believe that this is likely due to the composition of our data set. A more ambiguous data set would allow us to tease out any potential differences in performance between the feature sets—furthering the need for future research as discussed earlier.

8.3.3 Classifier Optimisation

In section 7.2.2, we searched for the optimum values of five parameters of our learning model. Due to time constraints, we resorted to optimising each parameter one-by-one, fixing the value for each parameter once an optimum value was found. By fixing values individually, this method is unable to account for complex relationships between parameters, and as such is unlikely to provide optimal results. We believe that this method is sufficient for our purposes as our project is focussed on the relative strength of the particular features in use, rather than testing the limits of our chosen classifier.

It is also worth noting that the optimum set of values is likely different for each feature set in use. Again, due to time constraints, we optimised our classifier using the existing set of simple lexical features. This places a bias on the existing simple features, meaning that any performance increase seen using other feature sets is likely underestimated. Given more time, or an opportunity to revisit our work, it is likely that we would be able to improve the performance of our classifier through the use of a more sophisticated optimisation plan, such as beam search (Ponte, Paquete and Figueira, 2012).

8.3.4 Responding To Change

In section 6.6.3, we described how tree-based prediction models are unable to adapt to significant variation in the distribution being modelled. We attempted to mitigate this through using an online classifier, the rationale being that new branches may be generated quickly in response to changes in the training data. This is not a perfect solution, however, and inevitably the classifier will have to be re-trained from the ground up if the modelled distribution changes significantly. We believe that this is only likely to occur over a period of months or years, so the benefit over batch-based methods of not having to re-train the classifier every week is not lost.

One potential method of ensuring that the model can adapt to changing data is provided in the work of (Saffari et al., 2009). Here, they suggest the continual removal of the most erroneous tree from the random forest, replacing it with a new tree. Unfortunately, due to time constraints, we could not explore this possibility; however, we believe this to be an interesting area for future development.

8.4 Learning

8.4.1 Domain Specific

Through undertaking this project, I have explored a great number of areas of research that extend beyond the units I have been taught during my course. In particular, I have furthered my knowledge of machine learning—especially in the online context—as well as developing a deeper understanding of feature selection and evaluation. I have gained an insight into state of the art methods of phishing prevention, as well as an intimate knowledge of the various practices employed by fraudsters. In addition, I have also explored various techniques of dimensionality reduction, as well as improving my knowledge in the area of textual classification in general.

8.4.2 Technical

The implementation of my classifier has strengthened my knowledge of Java 8, giving me an opportunity to explore the use of JSP and servlets—something that I had never encountered previously. Other relevant technical areas of learning include SQL (H2), JSON processing, concurrency and LaTeX.

8.4.3 Academic

Working on this dissertation has given me an unparalleled opportunity to develop my academic skills. Through the literature review, I learned to research a particular domain with both breadth and specificity. I learned to critically evaluate the work of others, and to establish my contribution with respect to existing work. My command of the written word has also improved as a result, as well as my ability to construct a hypothesis and draw conclusions from experimental results.

Bibliography

Abdelhamid, N., Ayesh, A. and Thabtah, F., 2014. Phishing detection based associative classification data mining. *Expert Systems with Applications*, 41(13), pp.5948 – 5959. Available from: http://doi.org/http://doi.org/10.1016/j.eswa.2014.03.019.

Aburrous, M., Hossain, M.A., Dahal, K. and Thabtah, F., 2010. Associative classification techniques for predicting e-banking phishing websites. *Multimedia computing and information technology (mcit), 2010 international conference on*. pp.9–12. Available from: http://doi.org/10.1109/MCIT.2010.5444840.

Alpaydin, E., 2010. *Introduction*, MIT Press, pp.1–19. Available from: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6284961.

Anderson, R., Barton, C., Böhme, R., Clayton, R., Eeten, M.J.G. van, Levi, M., Moore, T. and Savage, S., 2013. *Measuring the cost of cybercrime*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp.265–300. Available from: http://doi.org/10.1007/978-3-642-39498-0_12.

APWG, 2007. APWG Phishing Activity Trends Report june 2016. http://docs.apwg.org/reports/apwg_report_june_2007.pdf. Accessed: 2016-07-18.

APWG, 2016. APWG Phishing Activity Trends Report 2nd quarter 2016. http://docs.apwg.org/reports/apwg_trends_report_q2_2016.pdf. Accessed: 2016-10-17.

Batenkov, D., Friedland, O. and Yomdin, Y., 2015. Sampling, metric entropy, and dimensionality reduction. *SIAM Journal on Mathematical Analysis*, 47(1), pp.786–796. http://dx.doi.org/10.1137/130944436, Available from: http://doi.org/10.1137/130944436.

Baykan, E., Henzinger, M., Marian, L. and Weber, I., 2009. Purely url-based topic classification. *Proceedings of the 18th international conference on world wide web*. New York, NY, USA: ACM, WWW '09, pp.1109–1110. Available from: http://doi.org/10.1145/1526709.1526880.

Baykan, E., Henzinger, M., Marian, L. and Weber, I., 2011. A comprehensive study of features and algorithms for url-based topic classification. *ACM Trans. Web*, 5(3), pp.15:1–15:29. Available from: http://doi.org/10.1145/1993053.1993057.

Blei, D.M., Ng, A.Y. and Jordan, M.I., 2002. Latent dirichlet allocation. *Advances in neural information processing systems*, 1, pp.601–608. Available from: https://papers.nips.cc/paper/2070-latent-dirichlet-allocation.pdf.

Blum, A., Wardman, B., Solorio, T. and Warner, G., 2010. Lexical feature based phishing url detection using online learning. *Proceedings of the 3rd acm workshop on artificial intelligence and security*. New York, NY, USA: ACM, AISec '10, pp.54–60. Available from: http://doi.org/10.1145/1866423.1866434.

Breiman, L., 2001. Random forests. *Machine Learning*, 45(1), pp.5–32. Available from: http://doi.org/10.1023/A:1010933404324.

Brin, S. and Page, L., 1998. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1), pp.107 – 117. Available from: http://doi.org/10.1016/S0169-7552(98)00110-X.

Cortes, C. and Mohri, M., 2004. Confidence intervals for the area under the roc curve. *Nips*. pp.305–312. Available from: `https://papers.nips.cc/paper/2645-confidence-intervals-for-the-area-under-the-roc-curve.pdf`.

Dhamija, R., Tygar, J.D. and Hearst, M., 2006. Why phishing works. *Proceedings of the sigchi conference on human factors in computing systems*. New York, NY, USA: ACM, CHI '06, pp.581–590. Available from: `http://doi.org/10.1145/1124772.1124861`.

Domingos, P., 2012. A few useful things to know about machine learning. *Communications of the ACM*, 55(10), pp.78 – 87. Available from: `http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=82151052&site=ehost-live`.

Domingos, P. and Hulten, G., 2000. Mining high-speed data streams. *Proceedings of the sixth acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp.71–80.

Feo, T.A. and Resende, M.G., 1995. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2), pp.109–133.

Feroz, M.N. and Mengel, S., 2015. Phishing url detection using url ranking. *2015 ieee international congress on big data*. pp.635–638. Available from: `http://doi.org/10.1109/BigDataCongress.2015.97`.

FFA, 2016. Ffa uk: Fraud, the facts 2016. `https://www.financialfraudaction.org.uk/wp-content/uploads/2016/07/Fraud-the-Facts-A5_24.11_LR.pdf`. Accessed: 2017-01-25.

Fodor, I.K., 2002. A survey of dimension reduction techniques. *Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*, 9, pp.1–18. Available from: `https://e-reports-ext.llnl.gov/pdf/240921.pdf`.

Ganjisaffar, Y., 2017. crawler4j: Open source web crawler for java. `https://github.com/yasserg/crawler4j`. Accessed: 2016-10-03.

Garera, S., Provos, N., Chew, M. and Rubin, A.D., 2007. A framework for detection and measurement of phishing attacks. *Proceedings of the 2007 acm workshop on recurring malcode*. New York, NY, USA: ACM, WORM '07, pp.1–8. Available from: `http://doi.org/10.1145/1314389.1314391`.

Gill, P.E. and Murray, W., 1972. Quasi-newton methods for unconstrained optimization. *IMA Journal of Applied Mathematics*, 9(1), p.91. `/oup/backfile/Content_public/Journal/imamat/9/1/10.1093/imamat/9.1.91/2/9-1-91.pdf`, Available from: `http://doi.org/10.1093/imamat/9.1.91`.

Harris, Z.S., 1954. Distributional structure. *Word*, 10(2-3), pp.146–162. Available from: `http://doi.org/10.1080/00437956.1954.11659520`.

Hoffman, M., Bach, F.R. and Blei, D.M., 2010. Online learning for latent dirichlet allocation. *advances in neural information processing systems*. pp.856–864. Available from: `http://papers.nips.cc/paper/3902-online-learning-for-latent-dirichlet-allocation`.

Huang, H., Qian, L. and Wang, Y., 2012. A svm-based technique to detect phishing urls. *Information technology journal, volume 11, issue 7*. pp.921–925.

Huh, J.H. and Kim, H., 2012. *Phishing detection with popular search engines: Simple and effective*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp.194–207. Available from: http://doi.org/10.1007/978-3-642-27901-0_15.

James, J., L., S. and Thomas, C., 2013. Detection of phishing urls using machine learning techniques. *Control communication and computing (iccc), 2013 international conference on*. pp.304–309. Available from: http://doi.org/10.1109/ICCC.2013.6731669.

Jurafsky, D., 2000. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Upper Saddle River, N.J. : Prentice Hall. Includes bibliographical references (p. 851-902) and index.

Kausar, F., Al-Otaibi, B., Al-Qadi, A. and Al-Dossari, N., 2014. Hybrid client side phishing websites detection approach. *International journal of advanced computer science and applications, volume 5 issue 7, 2014*. Available from: http://doi.org/10.14569/IJACSA.2014.050720.

Khonji, M., Iraqi, Y. and Jones, A., 2011. Lexical url analysis for discriminating phishing and legitimate websites. *Proceedings of the 8th annual collaboration, electronic messaging, anti-abuse and spam conference*. New York, NY, USA: ACM, CEAS '11, pp.109–115. Available from: http://doi.org/10.1145/2030376.2030389.

Khonji, M., Jones, A. and Iraqi, Y., 2011. A novel phishing classification based on url features. *Gcc conference and exhibition (gcc), 2011 ieee*. pp.221–224. Available from: http://doi.org/10.1109/IEEEGCC.2011.5752505.

Laskov, P., Gehl, C., Krüger, S. and Müller, K.R., 2006. Incremental support vector learning: Analysis, implementation and applications. *J. Mach. Learn. Res.*, 7, pp.1909–1936. Available from: http://dl.acm.org/citation.cfm?id=1248547.1248616.

Le, A., Markopoulou, A. and Faloutsos, M., 2011. Phishdef: Url names say it all. *Infocom, 2011 proceedings ieee*. pp.191–195. Available from: http://doi.org/10.1109/INFCOM.2011.5934995.

Le, Q.V. and Mikolov, T., 2014. Distributed representations of sentences and documents. *Icml*. vol. 14, pp.1188–1196. Available from: http://proceedings.mlr.press/v32/le14.pdf.

Ludl, C., McAllister, S., Kirda, E. and Kruegel, C., 2007. *On the effectiveness of techniques to detect phishing sites*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp.20–39. Available from: http://doi.org/10.1007/978-3-540-73614-1_2.

Ma, J., Saul, L.K., Savage, S. and Voelker, G.M., 2009. Identifying suspicious urls: An application of large-scale online learning. *Proceedings of the 26th annual international conference on machine learning*. New York, NY, USA: ACM, ICML '09, pp.681–688. Available from: http://doi.org/10.1145/1553374.1553462.

Ma, J., Saul, L.K., Savage, S. and Voelker, G.M., 2011. Learning to detect malicious urls. *ACM Trans. Intell. Syst. Technol.*, 2(3), pp.30:1–30:24. Available from: http://doi.org/10.1145/1961189.1961202.

Miyamoto, D., Hazeyama, H. and Kadobayashi, Y., 2009. *An evaluation of machine learning-based methods for detection of phishing sites*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp.539–546. Available from: http://doi.org/10.1007/978-3-642-02490-0_66.

Moore, T. and Clayton, R., 2007. An empirical analysis of the current state of phishing attack and defence. *Workshop on the economics of information security*.

Netcraft, 2017. Netcraft: Anti-phishing services. https://www.netcraft.com/anti-phishing/. Accessed: 2017-04-16.

Nguyen, H.H. and Nguyen, D.T., 2016. *Machine learning based phishing web sites detection*, Cham: Springer International Publishing, pp.123–131. Available from: http://doi.org/10.1007/978-3-319-27247-4_11.

Nguyen, L.A.T., To, B.L., Nguyen, H.K. and Nguyen, M.H., 2014. A novel approach for phishing detection using url-based heuristic. *Computing, management and telecommunications (commantel), 2014 international conference on*. pp.298–303. Available from: http://doi.org/10.1109/ComManTel.2014.6825621.

OECD, 2009. Online identity theft. Available from: http://doi.org/10.1787/9789264056596-en.

Oza, N.C. and Russell, S., 2001. Experimental comparisons of online and batch versions of bagging and boosting. *Proceedings of the seventh acm sigkdd international conference on knowledge discovery and data mining*. New York, NY, USA: ACM, KDD '01, pp.359–364. Available from: http://doi.org/10.1145/502512.502565.

PhishTank, 2016. Phishtank statistics: June 2016. https://www.phishtank.com/stats/2016/06/. Accessed: 2017-04-04.

Ponte, A., Paquete, L. and Figueira, J.R., 2012. *On beam search for multicriteria combinatorial optimization problems*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp.307–321. Available from: http://doi.org/10.1007/978-3-642-29828-8_20.

Powers, D.M., 2011. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. Available from: http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf.

R, R., A, K., S, M., V, P. and L, S., 2014. Enhancing the precision of phishing classification accuracy using reduced feature set and boosting algorithm. *2014 sixth international conference on advanced computing (icoac)*. pp.86–90. Available from: http://doi.org/10.1109/ICoAC.2014.7229752.

Saffari, A., Leistner, C., Santner, J., Godec, M. and Bischof, H., 2009. On-line random forests. *2009 ieee 12th international conference on computer vision workshops, iccv workshops*. pp.1393–1400. Available from: http://doi.org/10.1109/ICCVW.2009.5457447.

Salton, G. and Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5), pp.513 – 523. Available from: http://doi.org/http://dx.doi.org/10.1016/0306-4573(88)90021-0.

Shannon, C.E., 2001. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1), pp.3–55. Available from: http://doi.org/10.1145/584091.584093.

Sunil, A.N.V. and Sardana, A., 2012. A pagerank based detection technique for phishing web sites. *Computers informatics (isci), 2012 ieee symposium on*. pp.58–63. Available from: http://doi.org/10.1109/ISCI.2012.6222667.

V, P.K. and A, K., 2014. Performance study of classification techniques for phishing url detection. *2014 sixth international conference on advanced computing (icoac)*. pp.135–139. Available from: http://doi.org/10.1109/ICoAC.2014.7229761.

Van Der Maaten, L., Postma, E. and Herik, J. Van den, 2009. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10, pp.66–71. Available from: https://www.tilburguniversity.edu/upload/59afb3b8-21a5-4c78-8eb3-6510597382db_TR2009005.pdf.

Whittaker, C., Ryner, B. and Nazif, M., 2010. Large-scale automatic classification of phishing pages. *Ndss '10*. Available from: http://www.isoc.org/isoc/conferences/ndss/10/pdf/08.pdf.

Xiang, G., Hong, J., Rose, C.P. and Cranor, L., 2011. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Trans. Inf. Syst. Secur.*, 14(2), pp.21:1–21:28. Available from: http://doi.org/10.1145/2019599.2019606.

Zhang, Y., Hong, J.I. and Cranor, L.F., 2007. Cantina: A content-based approach to detecting phishing web sites. *Proceedings of the 16th international conference on world wide web*. New York, NY, USA: ACM, WWW '07, pp.639–648. Available from: http://doi.org/10.1145/1242572.1242659.

Zhao, P. and Hoi, S.C., 2013. Cost-sensitive online active learning with application to malicious url detection. *Proceedings of the 19th acm sigkdd international conference on knowledge discovery and data mining*. New York, NY, USA: ACM, KDD '13, pp.919–927. Available from: http://doi.org/10.1145/2487575.2487647.

Appendix A

Simple Lexical Features

Simple Lexical Feature	Earliest Study
IP address present in the hostname	Garera et al. (2007)
Number of '.' characters in the hostname	Whittaker, Ryner and Nazif (2010)
Number of '/' characters in the path	James, L. and Thomas (2013)
Number of hex characters in the URL	Aburrous et al. (2010)
Total length of the URL	Aburrous et al. (2010)
Number of tokens in the hostname (split on '.', '-', '_', '@')	Le, Markopoulou and Faloutsos (2011)
Length of the longest token in the hostname	Le, Markopoulou and Faloutsos (2011)
Length of the path	Le, Markopoulou and Faloutsos (2011)
Length of the longest directory in the path	Le, Markopoulou and Faloutsos (2011)
Max. no. of '.' characters in any directory	Le, Markopoulou and Faloutsos (2011)
Max. no. of special characters in any directory ('@', '-', '_', '~', ',')	Le, Markopoulou and Faloutsos (2011)
Number of special characters in the path ('@', '-', '_', '~', ',')	Le, Markopoulou and Faloutsos (2011)
Length of the file name	Le, Markopoulou and Faloutsos (2011)
Number of '.' characters in the file name	Le, Markopoulou and Faloutsos (2011)
Number of special characters in the file name ('-', '_')	Le, Markopoulou and Faloutsos (2011)
Number of arguments	Le, Markopoulou and Faloutsos (2011)
Total character length of the argument string	Le, Markopoulou and Faloutsos (2011)
Length of the longest argument value	Le, Markopoulou and Faloutsos (2011)
Max. number of special characters in an argument value ('-', '_')	Le, Markopoulou and Faloutsos (2011)
Number of '@' characters in the URL	Aburrous et al. (2010)
Number of special characters in the hostname ('@', '-', '_')	Zhang, Hong and Cranor (2007)
Number of '.' characters in the URL	Zhang, Hong and Cranor (2007)
Use of 'https'	Aburrous et al. (2010)
Length of the hostname	Ma et al. (2011)
Port number present in the hostname	Le, Markopoulou and Faloutsos (2011)

Table A.1: The simple lexical features identified from previous research

Simple Lexical Feature	Phishing URL Characteristics											
	1	2	3	4	5	6	7	8	9	10	11	12
IP address present in the hostname	✓											
Number of '.' characters in the hostname	✓		✓									
Number of '/' characters in the path				✓				✓				
Number of hex characters in the URL									✓	✓	✓	
Total length of the URL						✓						
Number of tokens in the hostname	✓	✓	✓									
Length of the longest token in the hostname	✓		✓			✓						
Length of the path				✓				✓		✓		
Length of the longest directory in the path								✓		✓		
Max. number of '.' characters in any directory								✓				
Number of special characters in the path								✓				
Length of the file name								✓				
Number of special characters in the file name								✓				
Number of arguments									✓		✓	
Total character length of the argument string									✓		✓	
Length of the longest argument value									✓		✓	
Max. number of special characters in an argument value									✓			
Number of '@' characters in the URL											✓	
Number of special characters in the hostname		✓	✓									
Number of '.' characters in the URL	✓		✓					✓				
Use of 'https'												
Length of the hostname			✓									
Port number present in the hostname												✓

Table A.2: The phishing characteristic coverage by existing simple lexical features

Appendix B

Situated Lexical Features

Term			
www	angelfire	google	rand
com	blogspot	dropbox	paypal
verify	members	cmd	js
submit	web	mobile	email
co	yahoo	user	includes
uk	wellsfargo	css	signin
html	themes	verification	images
templates	newmail	userid	inboxlight
moncompte	impaye	fav	apple
battle	log	appleid	webapps
https	support	archives	account
security	online	service	update
document	wordpress	dcabccfaadd	bookmark
isapidll	cartasi	visa	ebay
banking	mpp	uploads	admin
connect	jehfuq	mailbox	secure
icloud	offerid	confirm	fichiers
sitestate	asdafvg	usaa	plugins
vjoxkqwhtogydw	pages	webmail	websc

Table B.1: A sample of the terms identified in the April 2016 training data using IDF

Appendix C

Empirical Evaluation

Month	Tr. Rate (URL/s)	Class. Rate (URL/s)	AUC
April	5,739	7,199	0.89027
May	4,550	8,636	0.91212
June	6,950	8,822	0.88734
July	5,829	9,223	0.87390
August	5,438	9,216	0.87164
September	5,724	9,850	0.86123

Table C.1: Performance statistics over 6 months of data with a term buffer of 100 items

Month	Tr. Rate (URL/s)	Class. Rate (URL/s)	AUC
April	7,481	11,358	0.88946
May	5,976	7,565	0.90615
June	8,568	7,558	0.89239
July	8,627	6,290	0.87169
August	7,986	9,108	0.87127
September	7,546	6,184	0.86025

Table C.2: Performance statistics over 6 months of data with a term buffer of 1,000 items

Month	Tr. Rate (URL/s)	Class. Rate (URL/s)	AUC
April	5,818	10,779	0.89817
May	5,555	9,447	0.91986
June	8,604	9,881	0.89279
July	8,030	9,595	0.88739
August	8,020	9,598	0.87417
September	8,809	10,476	0.86520

Table C.3: Performance statistics over 6 months of data with a term buffer of 10,000 items

Month	Tr. Rate (URL/s)	Class. Rate (URL/s)	AUC
April	6,114	11,797	0.85448
May	3,061	6,019	0.91143
June	5,128	8,839	0.89228
July	5,825	8,906	0.87763
August	5,921	8,538	0.86285
September	6,288	9,793	0.85965

Table C.4: Performance statistics over 6 months of data with a term buffer of 100,000 items

Simple Feature	Prop. Freq.
Length of the longest token in the hostname	0.16364
Length of the hostname	0.15303
Length of the path	0.14849
Length of the longest directory in the path	0.11666
Number of '.' characters in the hostname	0.07424
Length of the file name	0.06970
Total length of the arguments	0.06667
Number of hex characters in the URL	0.03333
Number of '/' characters in the path	0.03182
Length of the longest argument value	0.02727
Number of tokens in the hostname (split on '.', '-', '_', '@')	0.02121
Number of special characters in the hostname ('@', '-', '_')	0.02121
HTTPS present	0.01970
Max. number of special characters in any one directory ('@', '-', '_', '~', ',')	0.01364
Max. number of '.' characters in any one directory	0.01364
Number of '@' characters in the URL	0.01061
IP address present in the hostname	0.00455
Number of '.' characters in the file name	0.00455
Number of special characters in the path ('@', '-', '_', '~', ',')	0.00303
Number of special characters in the file name ('-', '_')	0.00303
Total number of '.' characters in the URL	0.00000
Max. number of special characters in any one argument value ('-', '_')	0.00000
Number of arguments	0.00000
Port number present in the hostname	0.00000
Total length of the URL	0.00000

Table C.5: A table showing the normalised proportion of nodes testing on each simple feature within a Hoeffding tree trained over 6 months of data using only the existing simple lexical features.

Simple Feature	Prop. Freq.
Number of '.' characters in the URL	0.00000
Presence of an IP address in the hostname	0.00000
Max. number of special characters in any one argument value ('-','_')	0.00000
Port number present in the hostname	0.00000
Number of '@' characters in the arguments	**0.00000**
Number of ':' characters in the hostname	0.00000
Hash present in path (\/[a-fA-F0-9]+[\/\.])	0.00000
Presence of a non key-value argument	**0.00000**
Length of the shortest argument key	**0.00000**
Length of the shortest directory	**0.00000**
Number of '@' characters in the URL	0.00000
Number of '.' characters in the file name	0.00000

Table C.6: A table showing the simple lexical features not selected by the Hoeffding tree. Features proposed within our work are emboldened.