Real time detection of malicious webpages using machine learning techniques

By Shafi Ahmed

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Intelligent Systems Research Centre
Faculty of Life Sciences and Computing
London Metropolitan University
February 2015

I would like to dedicate this thesis to my parents and my teachers.

Acknowledgements

I am very grateful to my main academic supervisor Professor Hassan B. Kazemian for his guidance, advice and constructive feedback throughout this long journey. To him, research through effective and clear communication, provides an opportunity to improve our lives. I also thank my other supervisors Dr. Shanyu Tang, Dr. Sahithi Siva and Dr. Igor Schagaev for their continuous support and motivation.

I would like to thank Technology Strategy Board for their generous support (UK [Grant no. KTP006367]) which gave me the opportunity to produce this thesis. My KTP supervisors Simon Boot, David Jai Parsad and Kate, and KTP advisor Jan Stringer encouraged me all the way during the ups and downs of the project. KTP's business partner Viridian Partnership LLP with which I am still involved with, has allowed me to work with more than 30 Web Projects (e-commerce, social networks and portal sites) built in various technologies (.NET, ASP.NET MVC, MySQL, SQL Server, C#, VB.NET, Python, Ruby, JavaScript, Android and the beloved Raspberry PI, etc.). The inspiration for the thesis came when I started working for the KTP project. I specially thank David Cranley,

Louise Cranley, Gary Green, Monika Jasinska and Nizza Samuel for their kindness and unparalleled support. I also thank Eddy Young, Nicola Fitzgerlad and Ian Liddicoat from the Publicis Groupe UK for their encouragement and allowing flexibility in my work. While working for Publicis Groupe UK and Viridian I have understood that applying the theories from computer science to a business is just as important as carrying out research as an academic. And if somehow you can come combine both, you can appreciate both fields and understand that collaboration is essential.

I also thank 'Jamee Nana', 'Abdul Haque bhai', 'Shuheb bhai' and my friends for their moral support. Lastly, I would like to thank my family (my dad, my mum, my brothers, my sister, my wife, my daughter and the rest) for their unparalleled support and deep love. I stayed away from my family for a long period of time, but their encouragement and advice made this journey easy.

Abstract

In todays Internet, online content and especially webpages have increased exponentially. Alongside this huge rise, the number of users has also amplified considerably in the past two decades. Most responsible institutions such as banks and governments follow specific rules and regulations regarding conducts and security. But, most websites are designed and developed using little restrictions on these issues. That is why it is important to protect users from harmful webpages. Previous research has looked at to detect harmful webpages, by running the machine learning models on a remote website. The problem with this approach is that the detection rate is slow, because of the need to handle large number of webpages. There is a gap in knowledge to research into which machine learning algorithms are capable of detecting harmful web applications in real time on a local machine.

The conventional method of detecting malicious webpages is going through the black list and checking whether the webpages are listed. Black list is a list of webpages which are classified as malicious from a user's point of view. These black lists are created by trusted organisations and volunteers. They are then used by modern web browsers such as Chrome, Firefox, Internet Explorer, etc. However, black list is ineffective because of the frequent-changing nature of webpages, growing numbers of webpages that pose scalability issues and the crawlers' inability to visit intranet webpages that require computer operators to login as authenticated users.

The thesis proposes to use various machine learning algorithms, both supervised and unsupervised to categorise webpages based on parsing their features such as content (which played the most important role in this thesis), URL information, URL links and screenshots of webpages. The features were then converted

to a format understandable by machine learning algorithms which analysed these features to make one important decision: whether a given webpage is malicious or not, using commonly available software and hardware. Prototype tools were developed to compare and analyse the efficiency of these machine learning techniques. These techniques include supervised algorithms such as Support Vector Machine, Naïve Bayes, Random Forest, Linear Discriminant Analysis, Quantitative Discriminant Analysis and Decision Tree. The unsupervised techniques are Self-Organising Map, Affinity Propagation and K-Means. Self-Organising Map was used instead of Neural Networks and the research suggests that the new version of Neural Network i.e. Deep Learning would be great for this research.

The supervised algorithms performed better than the unsupervised algorithms and the best out of all these techniques is SVM that achieves 98% accuracy. The result was validated by the Chrome extension which used the classifier in real time. Unsupervised algorithms came close to supervised algorithms. This is surprising given the fact that they do not have access to the class information beforehand.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background and related work

When the World Wide Web started, there was not an immediate need to improve the security of webpages. Web developers and designers focused on textual information and did not make huge use of graphics, JavaScript files or stylesheets. If the first Google search page [9] is taken as an example, it had very few features. This allowed the Google search page to load faster over a slow Internet connection. With the rise of broadband speed, users started to spend more time online. They started visiting more websites and the businesses took advantage of this phenomenon by providing more information and services online. Government organisations too provided information online and allowed access to information 24 hours a day. Users, instead of visiting places physically, could now visit webpages and got the information they wanted. This trend continued, and will continue to grow, unless any radical transformation in information access takes place.

Due to the open nature of the Internet, there are no vetting processes and individuals accessing various webpages have to rely on blacklists to determine whether a webpage is malicious or not. This thesis provides an alternative method using machine learning techniques to make this judgement without the help of a blacklist.

Security threats are increasing day by day and are very common [10]. Take for example online payments. The payment processing capabilities allow customers to carry out transactions online. Banks collaborate with businesses and provide facilities so that their customers can pay for products online. Many malicous websites take advantage of this and pose as 'genuine websites'. Users think that the websites are legitimate and provide personal information to buy products. For example emails are sent to a user from the websites and the user thinks that they were sent from legitimate businesses. These email have links to a malicious webpage. The user clicks the link, visits the malicious webpage created by the malicious attacker, which looks exactly like the bank's webpage. The user tries to log in but cannot. But in the meantime the malicious attacker gets hold of the username and password which it then uses to get hold of the original account.

Modern browsers have a new way to secure themselves. They look at publicly available blacklists of malicious webpages. These blacklists are updated after a few days or a month. The problem is that the blacklists do not safeguard the sudden changes within a webpage. Although the web crawler visits the webpages every few days, the websites are capable of causing damages within a short a period of time i.e. within those few days. At this point, users will already get affected, because the browser thought of it as a secure webpage and accessed

it, because it was not in the black list. On the other hand, a webpage may be hacked and injected with malicious code visible only to some particular users or a group of users from an organisation or a country. The blacklists will not be able to blacklist those either. Some crawlers do not validate the JavaScript code because JavaScript executes only in a browser. This allows client vulnerabilities to pass through easily. Even though some of the scripts which are assumed to be safe, but they can load remote malicious scripts and then execute them on the computer. Some scripts create iFrames and then load external malicious webpages. These external webpages get hold of the cookies and steal the identity. The users then browse this malicious webpage, get their computers infected and are then easily tracked by remote users from elsewhere. The users may also run malicious executable files without even knowing that the executable file has already access to the system.

Apart from the server side attacks, there are also client side attacks that can only be detected within a browser itself. These client side attacks can be categorised into the cross site scripting, clickjacking and download by attacks. These three attacks are described in the next sections.

1.2 Cross Site Scripting

1.2.1 Introduction

Cross site scripting injects malicious code from unexpected sources and causes various problems for the user [11]. This malicious code can get hold of the cookies, browsing history and then sends them over to the malicious webpage. There have been many attempts to prevent these sort of attacks [12]. It not only affects the user but also affects the server. The webpage is used as the vehicle to transfer infections to multiple users. The malicious code then executes in the user's browser. The problem has been intensified with the addition of scripting capabilities that did not exist at the initial stages of the Internet. With the addition of scripting capabilities, the users enjoy better user experience but have now become prone to these additional security problems. These scripts can run on the client's browser as they do not execute on the web server. Even if the web developer has built the webpage only using HTML, an attacker can inject scripts to make it susceptible to scripts. These scripts can then get access to the authentication cookies.

1.2.2 Affects

The XSS vulnerability affects the users as well the webpages hosting the webpage [11]. For example, a user visits a webpage and decides to buy from the webpage. The user adds the items to the basket and wants to checkout. Then he fills in a form to register. Each of these users are uniquely identifiable by the webpage through the use of cookies. The malicious attacker will be able to look at the cookies and impersonate the users and thus buy products, without the knowledge of the users.

1.2.3 Types of XSS attacks

There are various types of XSS attacks out of which three are tackled in this thesis. The first is reflection based attacks that take place with emails or Twitter and Facebook messages containing links to the malicious webpage [11]. This link directs the users to a malicious form that submits information to the attackers webpage and thus the attacker gets access to the user cookies which are used as identification mechanisms. The second is DOM Attack, i.e. attack based on document object model where an attacker inserts script tags into the webpage. This script tag has a source tag that links to the attacker's webpage. Or the

script in the JavaScript code writes something on the form based on the URL on the webpage. In the third type of attack, malicious code is stored on the web server which has been used as a target by the attacker.

1.2.4 Problems for the web administrators

Apart from the fact that the above incident anger users, the webpage owner too faces technical issues. For example, if the administrator of an e-commerce website browses the website for testing purposes, the attacker can get hold of the cookies of the administrator, log in to the backend and manage products. They can add, edit or reduce the prices of the products. When someone suspects that something is wrong is with the e-commerce website, the administrators will look at the logs only to find that the malicious activities were carried out by no other than the administrator.

1.2.5 Administrator access

To avoid the above scenario, the administrator should not access the webpage from a publicly available webpage to make sure that there are no contacts with the malicious scripts [13]. This is very hard to implement and there should be policies to enforce it. One way to do it, is to allow access to the administrators

only from a certain location or URL, which will be unknown to public. Also the IP address of that URL will also be blocked from external users. The firewall have to be set up properly so that this does not happen. The basic assumption is that the attacker will not be aware of how to even access as an administrator, even if the malicious attacker gets access to the administrator cookies. But there are ways to hide the actual IP address and replace with a fake IP address (in this case the valid IP address that has access to the administrator section). The attacker will obviously not have the response from the webpage but the purpose to inject the webpage with malicious code can be successful. But if the administrators have access to the public webpage, they can deploy a local version can be deployed to a machine within a secure environment. So that even if the local version is injected with malicious code the attacker will not be able to take any advantage.

1.2.6 DOM based attacks

Almost all HTML tags are wrapped by either 'greater than' and 'less than'. To inject the script tag, these two characters are needed. Several combination of characters can generate > [1]. The combinations of letters that generate the letters are dependent on browser version and the default language. The combinations are quite vast (see Figure 1.1). A browser cannot be trusted because of

<	<	<	<	<
%3C	<	<	<	<
&lt	<	<	<	<
<	<	<	<	<
&LT	<	<	<	<
<	<	<	<	<
&#60	&#x3c	&#X3c	&#x3C	&#X3C
&#060	&#x03c	&#X03c	&#x03C	&#X03C
&#0060	&#x003c	&#X003c	&#x003C	&#X003C
&#00060	&#x0003c	&#X0003c	&#x0003C	&#X0003C
&#000060	&#x00003c	&#X00003c	&#x00003C	&#X00003C
&#0000060	&#x000003c	&#X000003c	&#x000003C	&#X000003C
<	<	<	\x3c	\u003c
<	<	<	\x3C	\u003C

Figure 1.1: Values for <[1].

these extensive possibilities and some precautions are suggested one of which is to encode the data entered by the user and the data displayed to the user. This is known as sanitisation. In terms of how the webpage is deployed to the user, the operations team have to make sure that the firewall or any other forms of preventative measures are kept up to date.

Apart from the server side prevention techniques, the users can prevent themselves from the XSS attacks by disabling the JavaScript. This would have been acceptable at the beginning of the Internet when scripting were hardly in use. But these days when scripting is essential, the appropriate approach would be to detect the malicious XSS code and then stop it from executing. Kirda and Jovonovic [14] have provided a client side solution to this problem. This was

built as a windows application which allows a user to use it as a personal web firewall i.e. not dependent on the web developers or the security. Not only it has minimal user intervention, it also stops access to session and cookies which are the main targets for XSS attackers. The firewall looks at each of the incoming and outgoing connections and decides whether the connections are safe. It looks at the referrer header, POST requests and also tries to mitigate advanced XSS attacks.

1.3 Clickjacking

Another security threat that is difficult to detect is clickjacking [15]. This is a relatively new threat that has become more prominent with the advancement of modern browsers. Clickjacking does not not use security vulnerabilities, rather it uses the browsers' most common feature i.e. hyperlinks. The user is encouraged to click a link to a page. But this particular webpage has two webpages one which is displayed to the user and the other i.e the malicious page is hidden from the user. The hidden webpage executes malicious the code even though the user will think that the information is on the correct webpage. This technique is very hard to detect by inspecting the source code and there have not been many successful

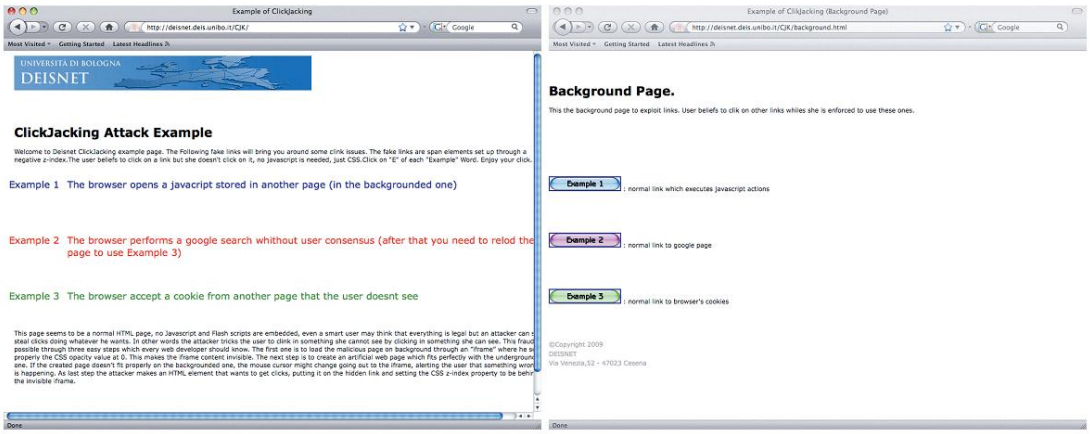

Figure 1.2: Clickjacking shows a displayed page and a hidden page [2].

ways to prevent it from happening. There are two techniques that are used to hide the hidden page. The first technique is to make the hidden page transparent using the CSS display property, which is set to none. The second technique places the object behind the original page using the CSS z-index property. The z-index property is set to the lowest value so that it stays behind all the time. This hidden webpage has hyperlinks behind the original hyperlinks. The position of these hyperlinks are exactly the same. So, when a user clicks the hyperlink displayed on the webpage, the user actually clicks the hyperlink on the hidden webpage. Just like the cross site scripting technique this click inject malicious code into the webpage. Similar to XSS attacks the clickjacking technique can be prevented by using NoScript. But this hinders the user experience and so is not a pragmatic approach. Figure 1.2 shows an example of clickjacking.

1.3.1 Types of clickjacking

There are various types of clickjacking out of which three are discussed in this thesis. The first method of clickjacking launches a small window via JavaScript and the user remains unaware of it. In the second method, the link executes a search via a search engine and also sends the query to the attacker's webpage. And in the third method, the link sends a new cookie to the web browsers and to steal the cookie. The third method which is a harder option, is to implement clickjacking on someone else's webpage. Such a webpage are likely to be already popular as the attacker will want the users to visit the webpage and click the hyperlinks. But if the webpage is already infected, the webpage will have a hidden page with hyperlinks behind the displayed page. If such a a webpage is affected, the web administrator will be able to detect assuming that regular checks are carried out.

Clickjacking are also spread by email messages that support HTML format. The displayed email will have hyperlinks with the malicious hyperlinks behind the main hyperlinks. This will allow the attacker to carry out the same damage to a user similar who downloads a malicious executable. The users may also get infected because malicious JavaScript which get hold of cookies allow the

attacker to impersonate the user (similar to the XSS example). The users may also get infected by accepting a cookie from the attackers webpage and thus will be traceable by the attacker.

1.3.2 Preventative measures

The easiest method to prevent clickjacking is to look at the status bar of the browser before clicking any links [12]. If the hyperlink looks suspicious, then the user should not click the link. The problem with this method is that the user may not know this. Depending on the implementation of the hidden page, the user may look at the source code and then decide whether the page is malicious or not. If the hidden page is behind the displayed page, the users will not be able to see the hidden page's source code. Otherwise, they can see the source code of the displayed page. Another option is to use text based browsers which do not execute any scripts on the browser. Thus the user will be safe from any attacks. This may not be the best option because the webpage nowadays are more than likely to use scripts and therefore will be dysfunctional in the text browser. Although HTML parsers can correctly see the HTML to track iframe, display or opacity property, these tags are used extensively in many webpages, thus triggering an alarm based on the existence of these, defeats the purpose. One

suggestion is to create a plugin for the browser to detect the number of clickable objects overlapping within an iframe and an z-index [2]. But the technique is not full proof in HTML5 browsers which has sandboxing properties that allows to avoid the barrier. This sandboxing attribute is present in iframes in the HTML5 specification [2].

1.4 Drive by downloads

Drive by download occurs when a file downloads on a user's PC without the knowledge of the user [16, 17]. This malicious executable then installs itself on the user's computer. This is a popular method which has been used by [18] to spread malware infection on the Internet. It uses three components in its attack: the web server, the browser and the malware. An attacker finds a web server to serve the malware. The webpage exploits any incoming user. These exploits use code to execute commands on the user's browser. The web server provides the malware which the browser downloads. The targeted browser has a known vulnerability that the attacker exploits. Internet Explorer had many instances of ActiveX loopholes that the attackers had used and are still using. There are some potential solutions to these problems [18]. The first solution is to

completely isolate the browser from the operating system so that any arbitrary code are not at all executed on the browser. Another solution is for web crawlers to visit webpage and see whether they are hosting any malware content. But the attackers can avoid this by using a URL that does not have a corresponding hyperlink. Crawlers by its nature only visits URLs that have a corresponding hyperlink.

1.5 Novel features

With wide ranging new threats that appear every day, attackers will devise new ways to avoid barriers raised by administrators and web developers. For improved security, an automated tool is ever more important to detect the vulnerabilities. One alternative approach to this automated approach is for web developers to secure and enhance their webpages. But there is only a certain extent that a developer can secure to secure a webpage. Web developers are bound by the web frameworks they use [17, 19, 20]. If the web frameworks fail to take preventative measures, the users' machines get infected and the webpages become vulnerable. This thesis takes the research further by applying several supervised machine learning techniques such as Naïve bayes Classifier, K-Nearest Neighbour, Random

Forest, Support Vector Machine, Linear Discriminant Analysis and Quadratic Discriminant Analysis and three unsupervised machine learning techniques like SOM, K-Means and Affinity Propagation. Moreover, the novel unsupervised techniques of K-means and Affinity Propagation have not been applied to detect malicious webpages by any other researchers in the past. The research uses a combination of both supervised and unsupervised techniques to further improve the efficiency of models.

1.6 Contributions

The aim of this thesis is to detect malicious webpages through machine learning and the contributions (listed below) provide a clear pathway to meet it.

- Provide an different approach to blacklists to predict whether webpages are safe, by analysing webpage content and other features in real time using machine learning techniques.
- Speed up the execution time of the learning process of the machine learning techniques by taking advantage of multiple computers.
- Create a mechanism to build Chrome browser extensions that can communicate with models from a local computer.

1.7 Organisation

Chapter 2 provides an overview of the security problems faced by modern webpages and also gives an overview of the current methods to tackle malicious webpages using machine learning techniques. Chapter 3 describes a method to repre-

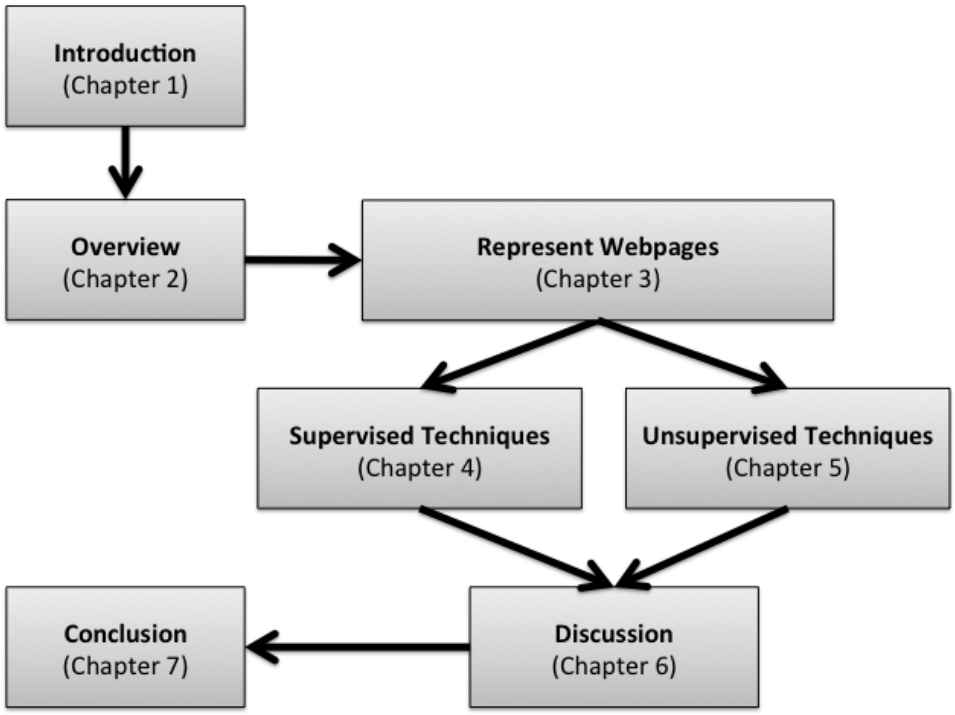

Figure 1.3: Organisation for Chapters

sent webpages in a form that is understandable by the machine learning models. Chapter 4 provides the results of the simulations carried out using the supervised machine learning techniques which include Support Vector Machine, K-Nearest

Neighbour, Latent Discriminant Analysis, Quadratic Discriminant Analysis, Decision tree and Random Forest. Chapter 5 provides results of the simulations carried out using the unsupervised machine learning techniques. This include the K-means, Self-organising map and Affinity Propagation. Chapter 6 discusses the findings from the supervised and unsupervised techniques and provides suggestions for future work. Chapter 7 concludes the thesis.

CHAPTER 2

Overview of machine learning algorithms and their usage to detect malicious webpages

2.1 Introduction

With wide ranging threats that are appearing each day, attackers will devise new ways to avoid barriers raised by administrators and web developers. For improved security, the security within a browser and the server needs to maintain safety through an automated tool that will learn to detect the vulnerabilities. One alternative approach to this automated approach is for web developers to secure and enhance their websites themselves. But there is only a certain extent that a developer can work to secure a webpage. Web developers are bound by the web frameworks [19] they use. If the frameworks fail to take preventative measures, the users' machines get infected, the webpages become vulnerable. With the expected rise of devices that access the Internet, it is now critical to improve the security of webpages.

Machine learning has seen itself being used in many places. A simple view of the machine learning process is that given a training set of data, the model trains itself and then when the new sets of data are passed on to it, it predicts the result. Recent improvements in hardware (especially in CPU and in GPU [21]) performance seen the use of machine learning increase by manifolds. Take for example, the detection of malicious webpages. The security companies are trying

to make use of such models and have seen some success. On the other hand, malicious attackers are increasing in strength and are looking for any chance to combat with new exploits. The problems are likely to rise in the future and new methods will have to be developed to tackle this rising phenomenon.

One of the ways this problem of malicious webpages could be resolved is through the use of client side detection methods. These methods are not very efficient, because the client machine may not be fast enough and the amount of computation required may not be enough. There are two types of classification that are used to detect malicious webpages. One is lightweight [22] and another is full fledged classification [23]. Full fledged classification involves many features e.g. host information, WHOIS, URL structure, mysterious characters in the dataset etc. The lightweight classification process use less features. This allows the lightweight algorithm to run faster because of less computation. But the disadvantage is that the detection rate is not as good as the full fledged implementation. Some researchers argue that the lightweight ones should be used on the client as they are quite capable of detectin malicious webpages to a satisfactory level [24, 25].

2.2 Previous work in this domain

Kin and Thi [26] carried out one of the first work that used machine learning to detect malicious webpages. This work ignored webpage content and looked at URLs using a bag-of-words representation of tokens in the URL with annotations about the tokens' positions within the URL. The noteworthy result from this work is that lexical features can achieve 95% of the accuracy of page content features. Garera's work [27] used logistic regression over 18 hand-selected features to classify phishing URLs. The features include the presence of red flag key words in the URL, features based on Google's page rank, and Google's webpage quality guidelines. They achieve a classification accuracy of 97.3% over a set of 2,500 URLs. Though this thesis has similar motivation and methodology, it differs by trying to detect all types of malicious activities. It also uses more data for training and testing. Spertus [28] suggested an alternative approach and looked at identifying malicious webpages and Cohen [29] has used the decision trees for detection and Dumais et al. [30] has used inductive learning algorithms and representations for text categorisation. This thesis has used similar techniques but applied them to webpages which have more complex structures. Guan et al [31] focused on classifying URLs that appear in webpages. Several URL-based

features were used such as webpage timing and content. But this thesis has used more features with better accuracy. Mcgrath and Gupta [32] did not construct a classifier but nevertheless performed a comparative analysis of phishing and non-phishing URLs. With respect to the data sets, they compare non-phishing URLs drawn from the DMOZ Open Directory Project to phishing URLs from a non-public source. The features they analyse include IP addresses, WHOIS thin records (containing date and registrar-provided information only), geographic information, and lexical features of the URL (length, character distribution, and presence of predefined brand names). The difference to this thesis is the use of different types of features i.e. content, screenshots, url and visual features of webpages. These features affect whether a webpage is malicious or not. Prvos et al. [33] performed a study of drive-by exploit URLs and used a patented machine learning algorithm as a pre-filter for virtual machine (VM) based analysis. This approach is based on heavyweight classifiers and is time consuming. Provos et. al [33] used the following features in computer simulation content based features from the page, whether inline frames are 'out of place', the presence of obfuscated JavaScript, and finally whether iFrames point to known exploit websites. Please note, an 'IFrame' is a window within a page that can contain another page. In their evaluations, the ML-based pre filter can achieve 0.1% false positives and

10% false negatives. This approach is very different to this thesis as the features are very primarily focused on iFrames. Bannur et al.'s [34] research is most similar to this thesis but it uses a very small dataset and this thesis uses other types of visual features.

2.3 Problems of using machine learning to optimise security of webpages

Current machine learning techniques that are being used to detect malicious webpages have yet to progress to deal with the complexity of learning problems because the majority of the research efforts in machine learning applications have concentrated on supervised algorithms. Malicious webpages are characterised with a large amount of evolving dynamic information. This evolving information requires feature selection/extraction, not only to reduce the dimensionality for machine learning, but also to capture the evolving characteristics. To discover the evolving patterns in data, machine learning techniques have to be combined into feature selection. New machine learning techniques and feature selection techniques are required to identify continuous behaviour in data.

2.4 Reasons for model representation

Applying machine learning techniques to web content requires them to be understandable to probabilistic reasoning. Initially, various natural language webpage representations were considered from the computational linguistics community [35]. Such representations used to address problems such as the part-of-speech tagging and they are not useful for the types of clustering and classification problems that will be looked at in this work. Rather, a vector space representation of web content has been used. In this representation, web content are cast as vectors in a very high dimensional space. Since probabilistic models can be computationally expensive to apply and may lack robustness in spaces with high dimensionality, some simple initial methods for dimensionality reduction has been examined in Chapter 3.

2.5 Machine learning techniques

The features found in each webpage are used to create very high dimensional feature vectors. Most of the features are generated by the "bag-of-words" representation of the URL, page links, content and visual features. The high dimensionality

of these feature vectors poses certain challenges for classification. Though only a subset of the generated features may correlate with malicious webpages, it is not possible to know which features are relevant in advance. More generally, when there are more features than labeled examples, there are possibilities that statistical models will be susceptible to over-fitting.

In this section, the discussion takes place on machine learning techniques that will be used for classification. Though individual classifiers differ in their details, the same basic protocol applies to all the models that are considered. The classifiers are trained on labeled examples to learn a decision rule that can ultimately be applied to unlabelled examples. Given an input x, the trained classifiers return a real-valued output $h(x)$ that sets a limit to obtain a binary prediction. The reason why this thesis uses binary prediction is to make it easier for a user to make a quick decision as to whether a webpage is malicious or not.

2.6 Supervised machine learning techniques

2.6.1 Support Vector Machine

SVM (see Figure 2.2) is widely regarded as one of the excellent models for binary classification of high dimensional data [36]. SVM and any other supervised

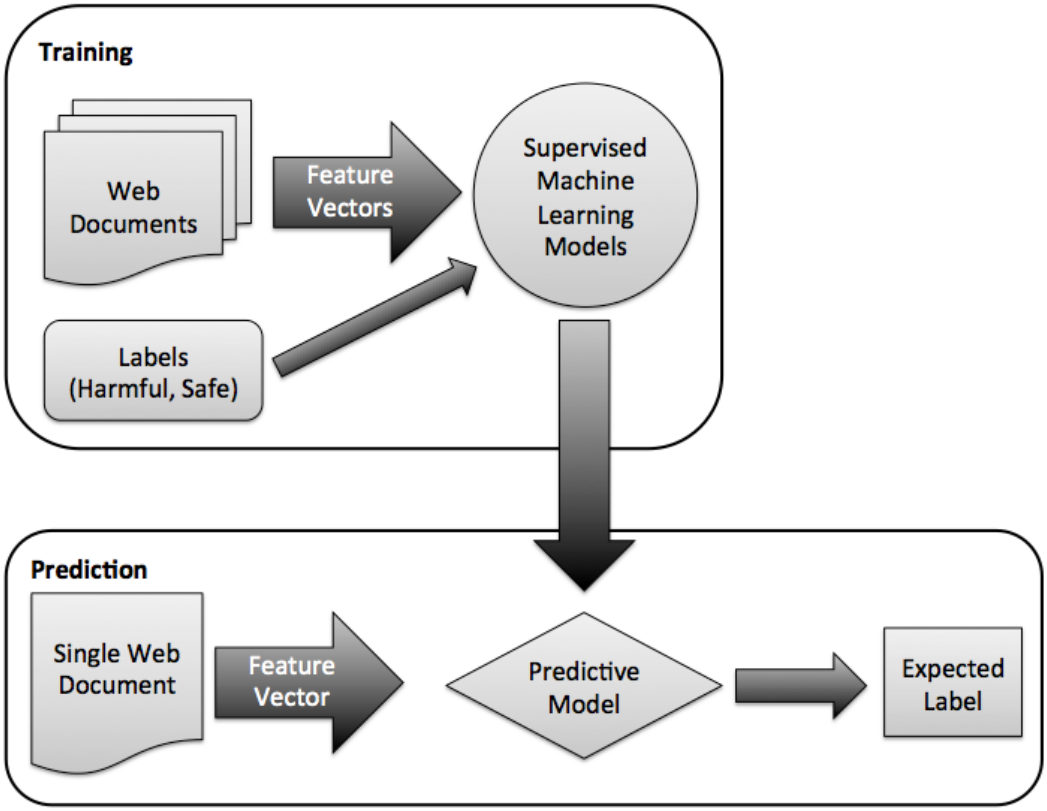

Figure 2.1: Supervised architecture for classifying webpages

classifiers use a similar technique to classify webpages as is shown in Figure 4.1.

SVM is modelled as

$$y(x) = \sum_{n=1}^{N} \lambda_n t_n x^T x_n + w_0 \tag{2.1}$$

The sign of this distance indicates the side of the decision boundary on which the example lies. The value of $y(x)$ is limited to predict a binary label for the feature vector x. The model is trained by first specifying a kernel function $K(x, x')$

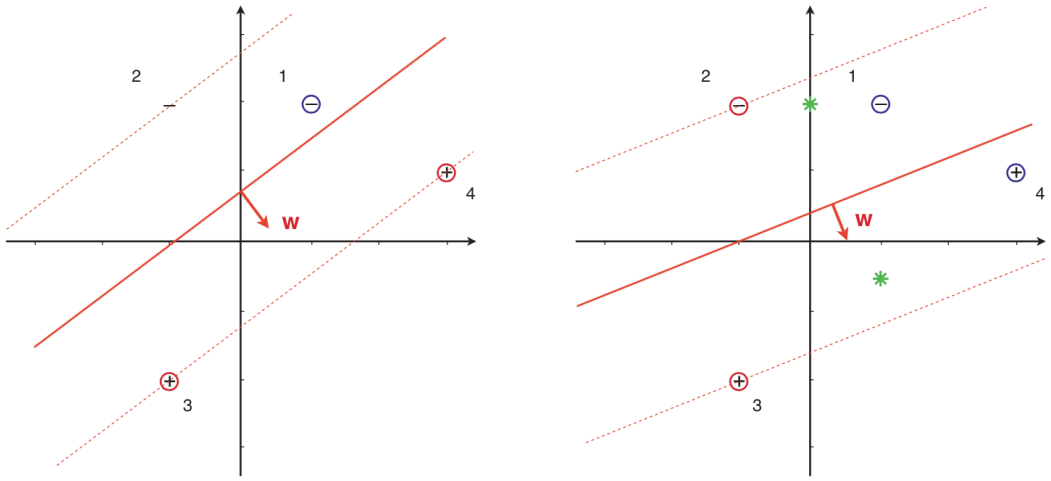

Figure 2.2: An example of Support Vector Machine classification where the soft margin classifier learned with $C = 5/16$, at which point x is about to become a support vector. On the right the soft margin classifier has learned with $C = 1/10$ all examples contribute equally to the weight vector. The asterisks denote the class means, and the decision boundary is parallel to the one learned by the basic linear classifier [3].

(this thesis uses RBF and linear kernels) and then computing the coefficients α_i that maximise the margin of correct classification on the training set. The required optimisation can be formulated as an instance of quadratic programming, a problem for which many efficient solvers have been developed [37].

2.6.2 Naïve bayes

Naïve bayes (see Figure 2.3) are mostly used in spam filters [38] and it assumes that, for a given label, the individual features of URLs are distributed independently of the values of other features [39]. Letting $P(x|y)$ denote the

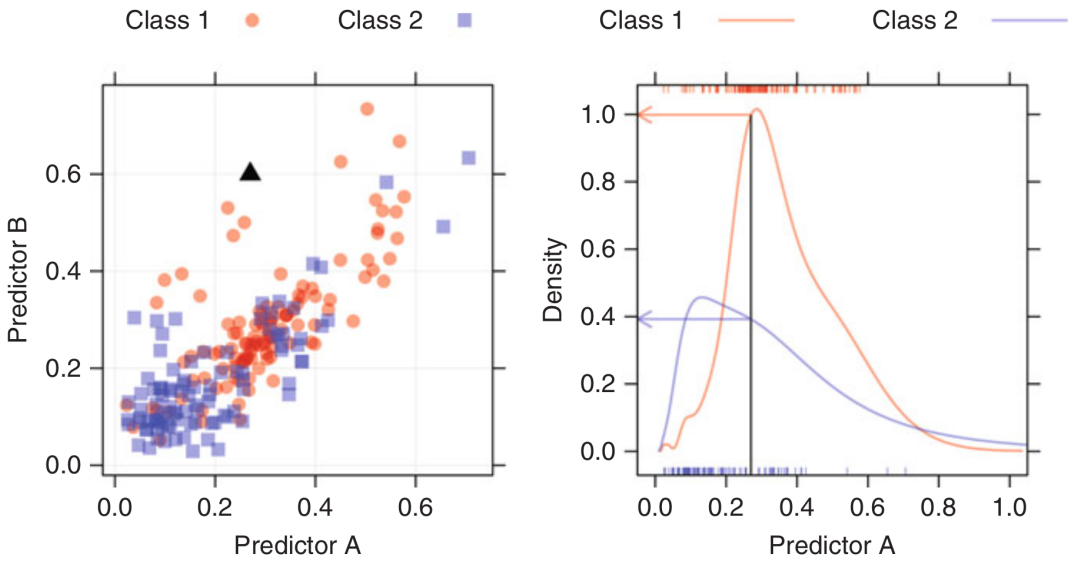

Figure 2.3: An example of Naïve Bayes classification [4]. The left picture shows a plot of two class data where a new sample represented by the solid triangle, is being predicted. On the right the conditional density plots of predictor A created using a nonparametric density estimate. The value of predictor A for the new sample is shown by the vertical black line.

conditional probability of the feature vector given its label, the model assumes $P(x|y) = \prod_{j=1}^{d} P(x_j|y)$. To calculate the conditional probability the thesis uses a product of the probability densities for each individual predictor.

By using Bayes rules it is assumed that malicious and safe webpages occur with equal probability. The posterior probability is computed so that the feature vector x belongs to a malicious webpage as:

$$P(y = 1|\mathbf{x}) = \frac{P(\mathbf{x}|y = 1)}{P(\mathbf{x}|y = 1) + P(\mathbf{x}|y = 0)} \tag{2.2}$$

Finally, the right hand side of Equation 2.2 can be thresholded to predict a binary label for the feature vector $\mathbf{x}$. The Naïve bayes classifier is trained by computing the conditional probabilities $P(x_j|y)$ from their maximum likelihood estimates [39]. For real-valued features, $P(x_j|y)$ is modelled by a Gaussian distribution whose mean and standard deviation are computed over the j_{th} component of feature vectors in the training set with label y. For binary valued features, $P(x_j = 1|y)$ is estimated as the fraction of feature vectors in the training set with label y for which the j_{th} component is one. The model parameters in the Naïve bayes classifier are estimated to maximise the joint log-likelihood of URL features and labels, as opposed to the accuracy of classification. Optimising the latter typically leads to more accurate classifiers, notwithstanding the increased risk of over-fitting.

2.6.3 Decision Trees

A Decision Tree (see Figure 2.4 for an example) uses a tree-like graph or model of decisions and their possible outcomes [40].

Decision Tree is a flow-chart like structure. The internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents class label [41]. Classification rules are represented by paths from roots

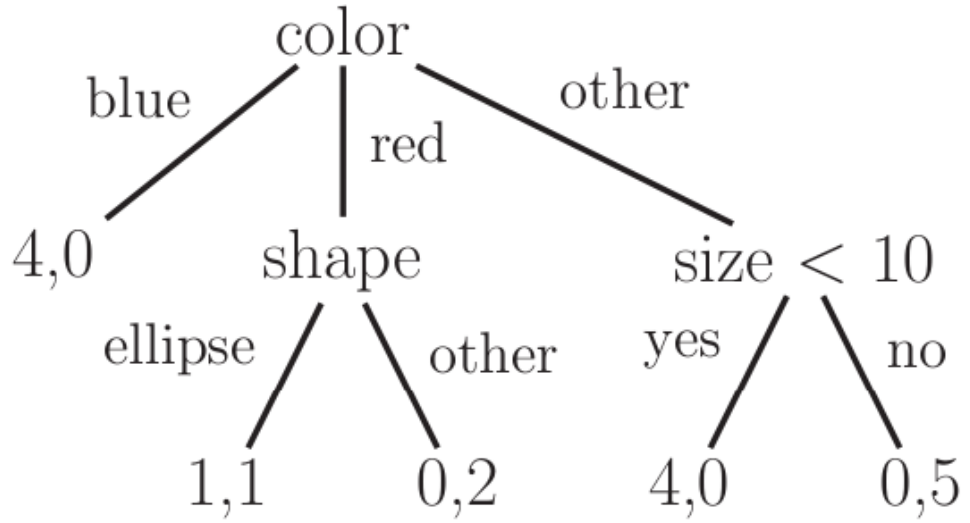

Figure 2.4: A leaf labeled as $(n1, n0)$ means that there are $n1$ positive examples that match this path, and $n0$ negative examples. Most of the leaves are "pure", meaning they only have examples of one class or the other; the only exception is leaf representing red ellipses, which has a label distribution of $(1, 1)$. Positive distinguishes from negative red ellipses by adding a further test based on size. However, it is not always desirable to construct trees that perfectly model the training data, due to overfitting [5].

to the leaves.

2.6.4 Random Forest

Random Forests (see Figure 2.5) are an ensemble learning method for classification (and regression) that operates by constructing a multitude of decision trees during training time. The algorithm for inducing a Random Forest was developed by Leo Breiman and Adele Cutler [42]. The term came from random decision

forests that was first proposed by Tin Kam Ho of Bell Labs in 1995 [43].

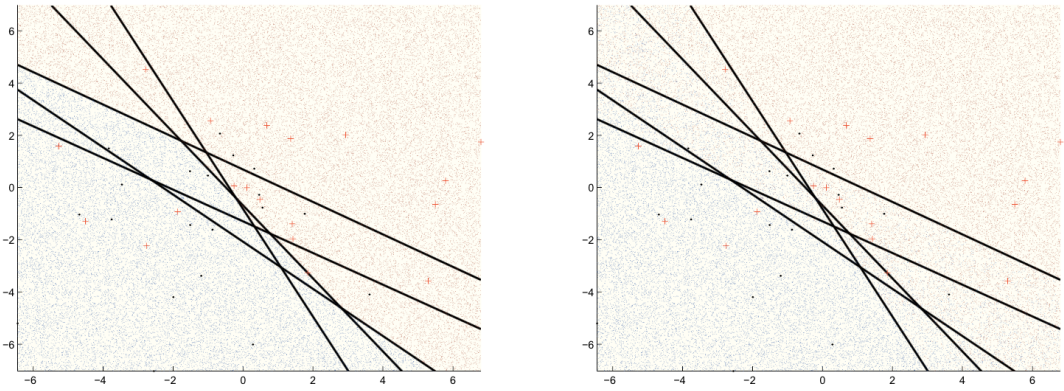

Figure 2.5: An ensemble of five basic linear classifiers built from bootstrap samples with bagging. The decision rule is majority vote, leading to a piecewise linear decision boundary. On the right if the votes are turned into probabilities, it is seen that the ensemble is effectively a grouping model: each instance space segment obtains a slightly different probability [3].

2.6.5 Quadratic Discriminant Analysis

Quadratic Discriminant Analysis or QDA (see Figure 2.6) assumes that the measurements from each class are normally distributed [44]. QDA does not assume that the covariance of each of the classes is identical. Likelihood ratio test is used when the normality assumption is true.

2.6.6 Linear Discriminant Analysis

Linear Discriminant Analysis or LDA (see Figure 2.7) finds a linear combination of features which characterises or separates two or more classes of objects or

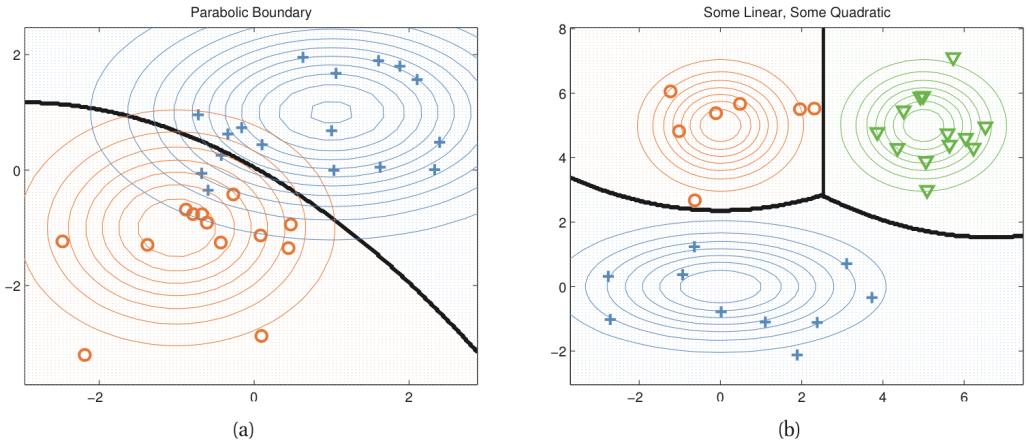

Figure 2.6: Linear decision boundaries using Quadratic Discriminant Analysis for the 2 and 3 class case shows that the two clusters are now separate [5].

events. The resulting combination may be used as a linear classifier or to reduce dimensions before later classification [45].

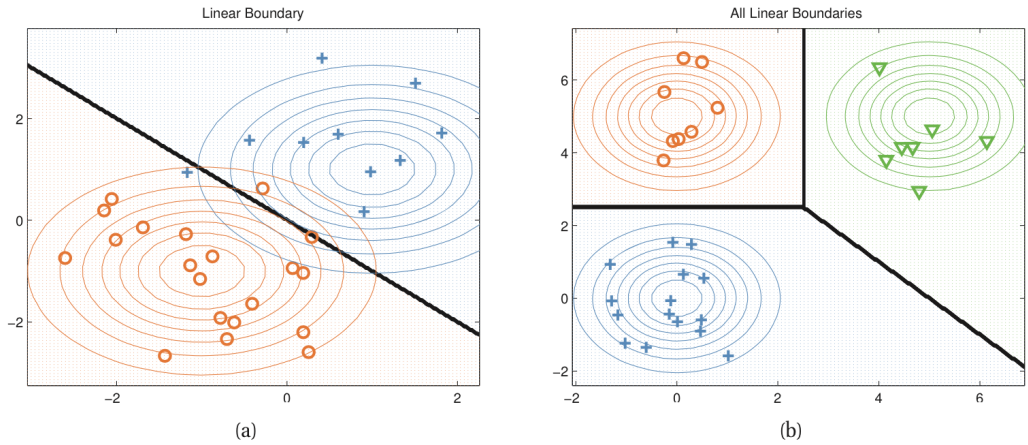

Figure 2.7: Linear decision boundaries using Linear Discriminant Analysis for the 2 and 3 class case shows that the two clusters are now separate [5].

LDA works when the measurements made on independent variables for each observation are continuous quantities [46, 47].

2.6.7 K-Nearest Neighbour

K-Nearest Neighbour (see Figure 2.8) classifies the label of a new point $\hat{x}$ with the most frequent label $\hat{t}$ of the k nearest training instances [48]. It is modelled

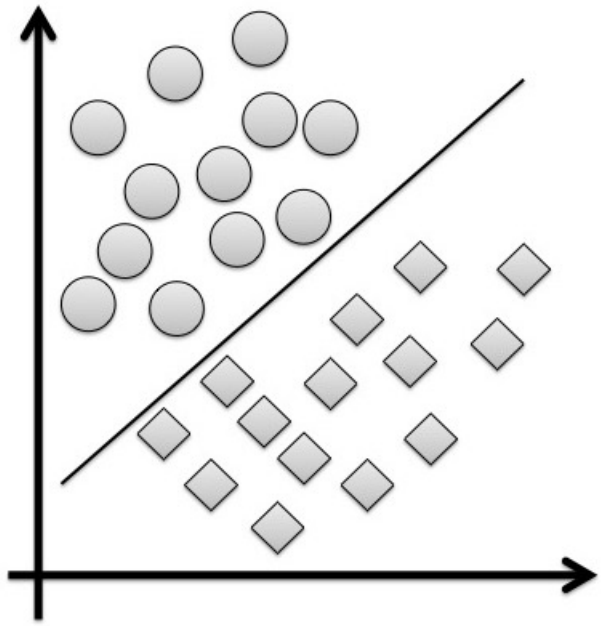

Figure 2.8: The nearest neighbour decision boundary separates the two classes.

$$\hat{t} = \arg\max_{\mathcal{C}} \sum_{i:x_i \in N_k(\boldsymbol{x},\hat{x})} \delta(t_i, \mathcal{C}) \tag{2.3}$$

where:

- $N_k(\boldsymbol{x}, \hat{x}) \leftarrow k$ points in $\boldsymbol{x}$ closest to $\hat{x}$

- Euclidean distance formula: $\sqrt{\sum_{i=1}^{D}(x_i - \hat{x}_i)^2}$
- $\delta(a, b) \leftarrow 1$ if $a = b$; 0 o/w

The model does not require any optimization and it trains itself using cross validation to learn the appropriate k. k regularizes the classifier, as $k \rightarrow N$ the boundary becomes smoother. $\mathcal{O}(NM)$ is used as space complexity, since all training instances and all their features need to be kept in memory. K-Nearest neighbor uses a very simple technique for classification, and cannot handle large training dataset as shown in the results section.

2.7 Unsupervised machine learning techniques

So far, the discussion has focused on supervised machine learning techniques. This section focuses on the unsupervised machine learning techniques where the class information is not available beforehand.

2.7.1 Affinity Propagation

Affinity Propagation (see Figures 2.9 and 2.10) is an unsupervised machine learning technique created by Frey and Dueck [49] where each data point acts as centroids. These data points choose the number of clusters. The following represent

the centroid for datapoint i

$$c_i \in \{1, ..., N\} \tag{2.4}$$

The goal is to maximise the following function

$$S(\mathbf{c}) = \sum_{i=1}^{N} s(i, c_i) = \sum_{k=1}^{N} \delta_k(\mathbf{c}) \tag{2.5}$$

The similarity of each point to the centroid is measured by the first term in and the second term is a penalty term denoted as $-\infty$. If some data point i has chosen k as its exemplar that is $c_k \neq k$, but k has not chosen itself as an exemplar i.e. $c_k = k$, then the following constraints could be presented.

$$\delta_k(c)(\mathbf{c}) = \sum_{i=1}^{N} s(i, c_i) = \sum_{k=1}^{N} \delta_k(\mathbf{c}) \tag{2.6}$$

$$\delta_k(\mathbf{c}) = \begin{cases} -\infty & \text{if } c_k \neq k \text{ but } \exists_i c_i = k \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

A factor graph can represent the objective function and it is possible to use

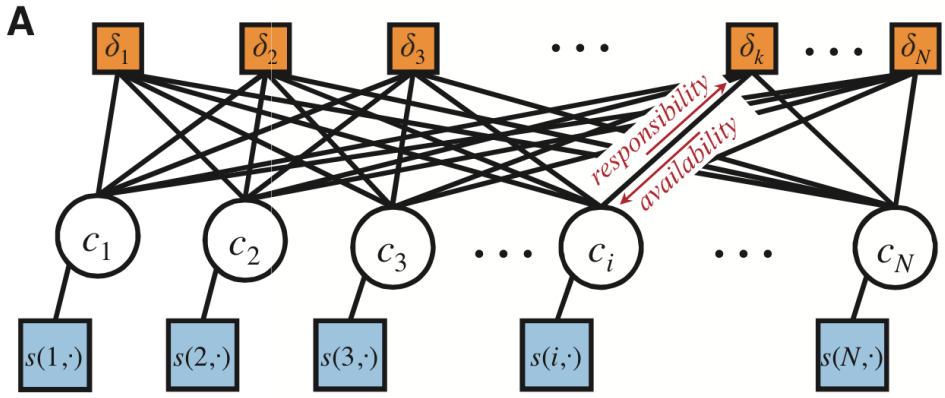

Figure 2.9: Affinity Propagation Factor Graphs for Affinity Propagation. Circles are variables, squares are factors. Each node has N possible states. [5].

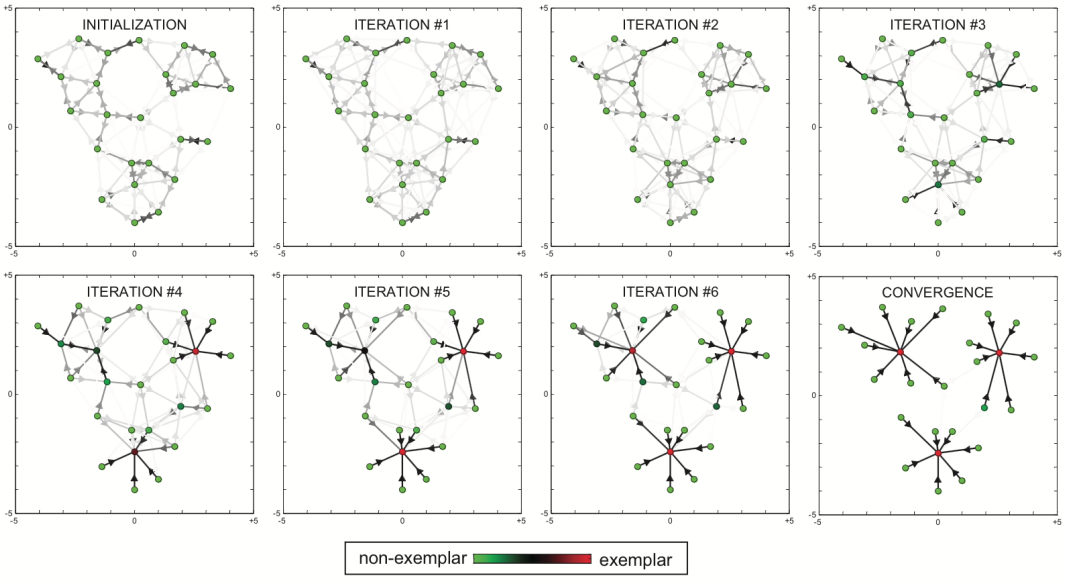

Figure 2.10: Example of Affinity Propagation. Each point is colored coded by how much it wants to be an exemplar. This can be computed by summing up all the incoming availability messages and the self-similarity term. The darkness of the ik arrow reflects how much point i wants to belong to exemplar k [5].

N nodes, each with N possible values or with N^2 binary nodes. Each variable

node c_i sends a message to each feature node δ_k and each factor node δ_k sends

a message to each variable node c_i. The number of clusters is controllable by scaling the diagonal term $S(i,i)$, which shows how much each data point would like to be an exemplar. Affinity Propagation has been developed very recently i.e. in 2009 and it has a very good performance as shown later in the results section.

2.7.2 Self-Organising Map

A SOM (see Figure 2.11) consists of a fixed lattice (typically 2-dimensional) of processing elements. Each processing element has an associated (initially random) prototype vector. In a SOM different parts of the Neural Network respond similarly to certain input patterns. The weights of the neurons in the SOM are set to either to small random values or sampled evenly from the subspace spanned by the two largest principal component eigenvectors. The SOM is then fed by a large number of example vectors that represent, as close as possible, the kinds of vectors expected during mapping [50]. The examples are usually administered several times. The training utilises competitive learning. When a training example is fed to the network, its Euclidean distance to all weight vectors is computed. The neuron with weight vector most similar to the input is called the best matching unit (BMU). The weights of the BMU and neurons close to it in the SOM lattice are adjusted towards the input vector. The magnitude of the

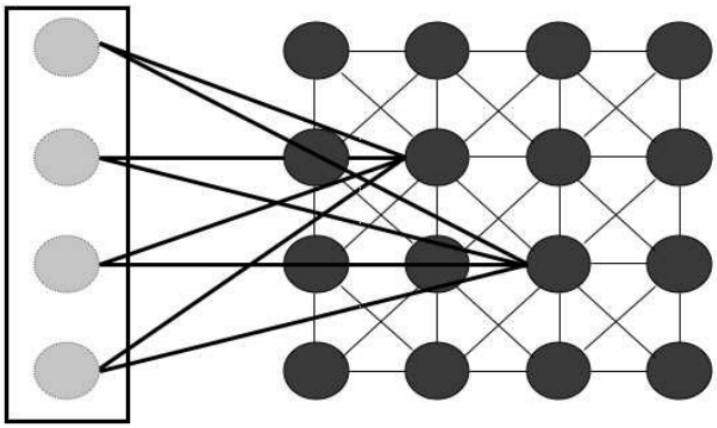

Figure 2.11: SOM architecture shows the input nodes (on the left) which do no computation, and the weights are modified to change the activations of the neurons. However, the nodes with the SOM affect each other in that the winning node also changes the weights of neurons that are close to it [6].

change decreases with time and with distance from the BMU.

2.7.3 K-Means

K-Means (see Figure 2.12) is a hard-margin, geometric clustering algorithm, where each data point is assigned to its closest centroid [51]. It is modelled using hard assignments $r_{nk} \in \{0, 1\}$ s.t. $\forall n \sum_k r_{nk} = 1$, i.e. each data point is assigned to one cluster k. The geometric distance is calculated using the Euclidean distance, l^2 norm:

$$||x_n - \mu_k||_2 = \sqrt{\sum_{i=1}^{D}(x_{ni} - \mu_{ki})^2} \tag{2.8}$$

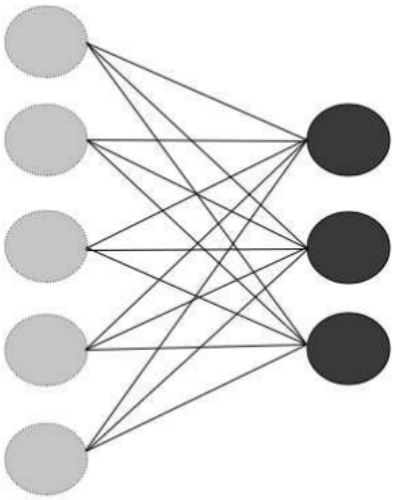

Figure 2.12: A single layer Neural Network can implement the K-Means solution. [6]

where:

- μ_k is cluster centroid
- D is the no of points
- x is the one of the points.

The Mini-Batch K-Means algorithm uses mini batches to reduce the computation time while still attempting to optimise the same objective function [52]. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence times of K-Means, Mini Batch K-Means produces results. Mini batch

K-Means converges faster than K-Means, but the quality of the results is reduced. In practice, the difference in quality can be quite small, as shown later.

2.8 Online learning

Online learning algorithms solve online classification problem over a sequence of pairs that are generally represented by a feature vector and a label [53]. After predicting, the algorithm receive the actual label and record the error rate. The algorithm then predicts the hypothesis for the next time step. The models are presented in order of increasing complexity with respect to the objective functions and the treatment of the classification margin. Whenever there are mistakes, the classical algorithm updates its weight vectors. Because the update rule is fixed, the model cannot account for the severity of the misclassification. As a result, the algorithm can overcompensate for mistakes in some cases and under compensate in other cases.

2.9 Gaps in the current state of research and the approach of this thesis

Other work that detect malicious webpages using machine learning techniques are focused either on blacklists [33] or URL information [8] or content or execution trace. Moreover, other work have focused on a single or up to few features at most, but this thesis uses contents, URLs, links and screenshots from webpages which allow to understand the webpages in a comprehensive manner. It also looks at using unsupervised machine learning techniques and tries to improve the performance of the classifiers by using multiple machines.

2.10 Summary

This Chapter looked at the machine learning techniques used in the thesis. These techniques are used in Chapters 4 and 5. In Chapter 5 the SOM technique was improved using multicore machines whereas in Chapter 4 the techniques were improved using a combination of features. The incorporation of web content opens various areas to explore. Up to this point, other work have used linear classification because it scales easily to large scale problems. However, content

contains high level structure and semantics that are potentially useful.

With the knowledge gained through this literature review process, the thesis, in the next Chapter, describes the features used in the simulation and provides a comprehensive framework that converts webpages into a format readable by machine learning models.

CHAPTER 3

Representation of web content for machine learning techniques

Computing is not about computers any more. It is about living.

Figure 3.1: Text from the first webpage

The hire charge is computed on a daily basis.

Figure 3.2: Text from the second webpage

3.1 Vector space representation

Data mining problems generally use the vector space to represent web content [54]. Each webpage either uses a Boolean or a numerical vector where each dimension in the vector corresponds to a distinct term in the webpage content. A given webpage has in each component a numerical value specifying some function f of how often the term corresponding to the dimension appears in the webpage. By varying the function f, alternative term weightings [55] can be produced. A term is a sequence of alpha numeric characters which is separated either by white space or tabs or newline characters or punctuation marks. Also, all upper case letters in a webpage are converted to lower-case to ignore capitalisation. Figures 3.1and 3.2 shows example webpages. The first webpage contains a famous quote from Nicholas Negroponte.

Table 3.1 shows the results of parsing these two webpages into single-word

Term	Vector for webpage 1	Vector for webpage 2
a	0	1
about	2	0
any	1	0
basis	0	1
charge	0	1
computed	0	1
computers	1	0
computing	1	0
daily	0	1
hire	0	1
is	2	1
it	1	0
living	1	0
more	1	0
no	0	1
not	1	0
on	0	1
the	0	1

Table 3.1: A simple vector representation of the sample webpages

terms, and then representing them as vectors with simple term frequencies (i.e., term counts) in each component. Such a representation is known as a bag of words [56], since the relative position of terms in the webpage, and hence the language structure, is not recorded in the resulting vectors.

In this thesis, the terms that defined the dimensions of the vector space are word stems. For example, the words "computed", "computers" and "computing" would be converted to 'comput". Porter [57] has developed a popular algorithm for word stemming and this algorithm has been incorporated into WAC's HTML

Comput is not about comput ani more It is about liv.

Figure 3.3: Stemmed text from the first webpage

The hire charg is comput on a daili basi

Figure 3.4: Stemmed text from the second webpage

parser (WAC is the name of the tool developed for this thesis). The two sample webpages presented earlier would look like if the contents were stemmed using Porter's stemming algorithm. Table 3.2 shows the vector representation of the stemmed version of the webpages. Stemming may be useful to help reduce similar terms in most cases but in other cases the results of stemming can be counter productive. Frakes [58] compared various stemming methods to unstemmed representations and showed that both representations perform equally in many cases.

Some researchers have defined the dimensions of a vector space with multi word phrases such as "Prime Minister David Cameron" and "personal computer". These multi-words appear frequently as sequences of words [59], or by applying NLP to detect meaningful phrases [60] or manually look for specific phrases [28, 61]. Going back to stemming, previous results using multi-word terms are mixed. Some researchers report that using terms can help improve accuracy for classification tasks [29] whereas others have found them to be as effective as

Term	Vector for webpage 1	Vector for webpage 2
a	0	1
about	2	0
ani	1	0
basi	0	1
charg	0	1
comput	2	1
daili	0	1
hire	0	1
it	1	0
is	2	1
liv	1	0
more	1	0
not	1	0
on	0	1
the	0	1

Table 3.2: A vector representation of the stemmed version of the stemmed webpages

single-word terms [30].

3.2 Frequency based vectors

This thesis uses frequency based vectors where $\xi(t_i, d)$ denotes the number of occurrences of term t_i in webpage d. The function f can be applied to $\xi(t_i, d)$ and produce the value for the i-th component of the vector for webpage d. The identity function $f(\alpha) = \alpha$ is applied to the term counts [62]. Other common functions applied to terms were defined by [63, 64, 65]. But TFIDF is probably the most popular function applied to webpages. This function uses term frequencies

(TF) in each webpage used as part of the weighting function alongside the inverse webpage frequency (IDF) of each term in the entire collection. IDF is usually defined as

$$\text{IDF}(t) = log(\frac{N}{n}), \tag{3.1}$$

where N is the total number of webpages in the collection and N_t is the number of webpages in which term t appears at least once. The TFIDF weight for a term t in a webpage d is the product of the term frequency and the inverse webpages frequency for that term, returning:

$$\text{TFIDF}(t, d) = \xi(t, d).\text{IDF}(t). \tag{3.2}$$

.

3.3 Alternative representations

Although TFIDF weighting has been used in the part primarily for retrieval, connections between this weighting scheme and probabilistic classification using the naïve bayes algorithm have been recently explored [66]. Parametric distribution such as bounded Gaussian or Poisson distribution can capture the probability of

words appearing different numbers of times in webpages [67, 68]. Alternatively, a simple Boolean representation of webpages can be used, that records whether or not a given term appears in a webpage. In this case, the following are found

$$f(n) = \begin{cases} 1 & \alpha \geqslant 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Most rule-base methods [29, 69] use a underlying Boolean model, as the antecedents of the classification rules only considers word presence and absence in webpages. Boolean vector representation has also been used in probabilistic classification models [70]. Although it may look like a bad approach, sometimes it has its uses in various applications. Neither the parametric distribution of the Boolean representations have been used in the thesis though.

3.4 Reduce dimensionality

The World Wide Web is one of the most diversified environments filled with content. To deduce vector space representations from this content would be a humongous task. This problem asks for a solution that is scalable and reducing dimensionality is one that has been used extensively by many researchers.

One way to tackle it is to look at English Literature where many words such as prepositions, conjunctions and pronouns provide structure in language rather than content. These words can thus be removed from the vectors. Words listed in [71] can also be removed. Where there are common words that appear once or twice among webpages play insignificant role and can also be removed [72].

3.5 Features

3.5.1 Webpage content

The semantic features included the TFIDF vectors which were derived from the webpages and the whole process has been described in Sections 3.1 - 3.4 in detail. In summary, the webpages have all their HTML tags removed, then the stop words were removed and then the remaining text were used in the simulations.

3.5.2 URLs

URLs identify webpages and have been used in the thesis as unique identifiers. Many malicious webpages have suspicious looking characters in their URLs and in their contents. Sometimes the URLs have spelling mistakes too. The lexical features of URLs were fed into the machine learning techniques. If there were

spelling mistakes or suspicious characters in the URL, then they were regarded as suspicious.

3.5.3 Webpage links

Webpages have many links that give out further information e.g. webpages that link to malicious webpages are likely to be malicious. The simulations extracted all the links from each webpage and they were fed into the models too.

3.5.4 Visual features

All the features that have been mentioned so far are text based e.g. source code, stripped HTML, domain names, URL etc. A complimentary feature was the use of image based features. First, screenshots of webpages were downloaded by passing the URLs to PhantomJS (a headless webkit browser with JavaScript API). It took each URL, saved a screenshot of the webpage and converted them to PNG file format. Images were then converted to a format understandable by the models. There are two popular techniques that are generally used i.e. Speeded Up Robust Features (SURF) and Scale Invariant Feature Transform (SIFT) [73]. The simulation used SURF because it had less stringent licensing options compared to SIFT [74]. The idea is that malicious webpages will look similar because are

likely to be have less input from designers whereas safe webpages will have better designs.

3.6 Summary

In this Chapter a framework has been built to extract various features from webpages to feed into the models. The features include content, source code, URLs and visual features from webpages which were extracted using crawlers and parsers. This framework lays foundation for the next two Chapters where various machine learning techniques (both supervised and unsupervised) use these features.

CHAPTER 4

Computer simulation results using supervised models to detect malicious webpages

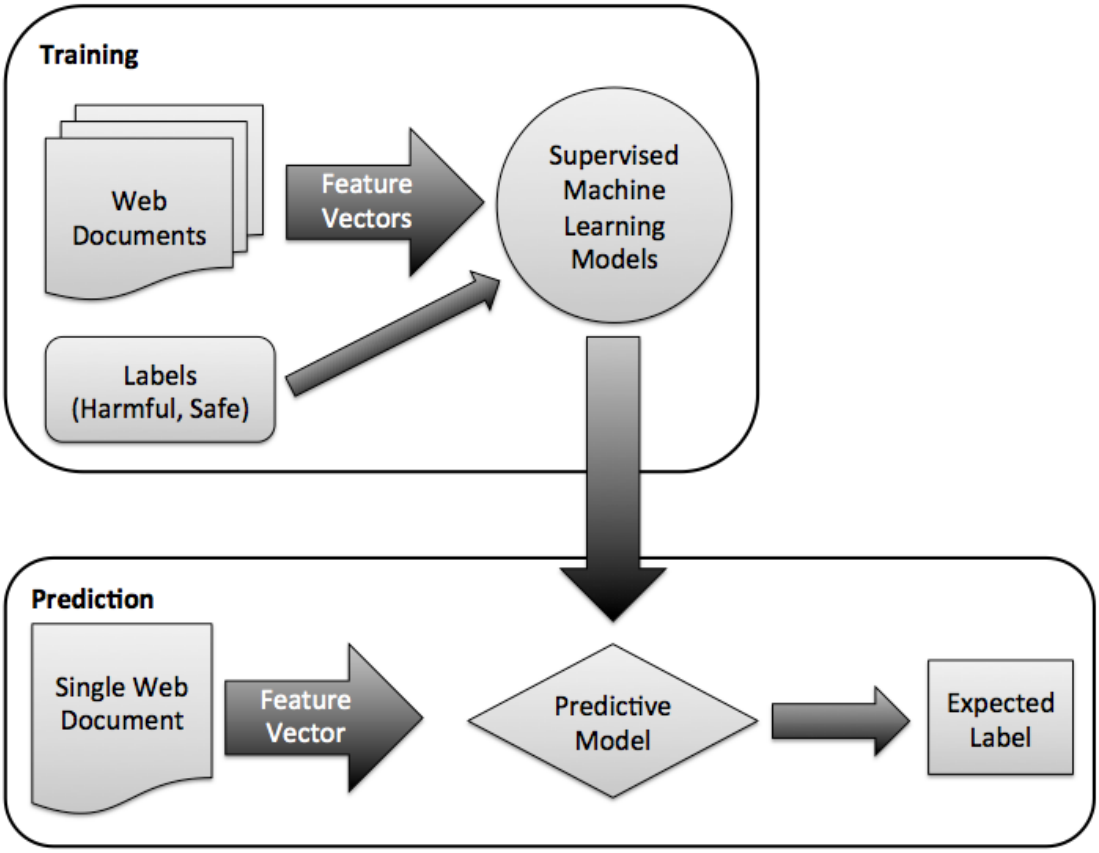

Figure 4.1: Architecture for classifying webpages using supervised techniques

4.1 Introduction

This Chapter focuses on the supervised models and discusses the results from these models. These models are popular in text classification.

Chapter 3 mentioned that this thesis used the bag of words approach, due to its simplicity and to avoid computational complexity. Figure 4.1 shows how supervised models have been used to classify the webpages. It is very similar to

unsupervised models but the only difference is in the training phase, where the labels are used to train it.

4.2 Results

Figure 4.2 shows the architecture of the tool that carried out the simulation for this thesis. The simulation was carried out on a machine running on Intel Xeon E3-1220 CPU with 4 Cores each having a speed of 3.1 GHz. The machine had 12 GB of RAM. First, 100,000 webpages were downloaded using a crawler based on gevent which uses libevent [75]. Out of these, 70% were used in training and the rest i.e. 30% were used in test. These downloaded webpages were then converted into feature vectors. Then a tool named Web Application Classifier (WAC) took these vectors as inputs, used the machine learning algorithms described in the previous section to create the predictive models. These predictive models read vectors of the new webpages to produce an output that indicated whether a webpage is safe or not. Browsers like Chrome were able to connect to these predictive models stored inside WAC to indicate whether a webpage is malicious or not. Chapter 3 described the features that were gathered after the webpages were preprocessed and cleansed before placing inside the predictive models.

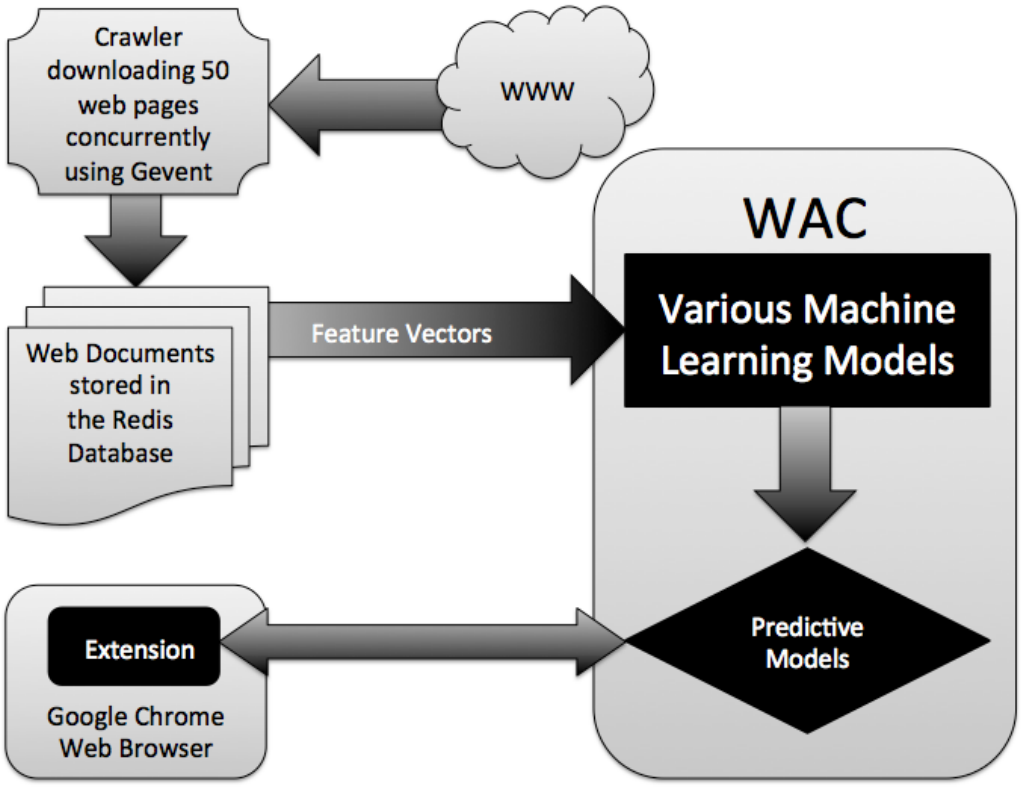

Figure 4.2: The architecture of Web Application Classifier (WAC)

4.2.1 Data sources

The downloaded webpages were divided into two sets i.e. malicious and safe. The source for the list of safe webpages were gathered from primarily from Alexa. The malicious ones were gathered from various sources primarily from [76]. Two types of representation of each webpage were created. One contained all the HTML code and other only had English characters.

4.2.2 Evaluation of the supervised machine learning techniques

It is possible to use each classifier without considering the others. But this can be potentially misleading in two ways. First, if two classifiers are highly correlated with the response and with each other, then the univariate approach will identify both as important. Some models will be negatively impacted by including this redundant information. Pre-processing approaches such as removing highly correlated classifiers can alleviate this problem [77]. Second, the univariate importance approach will fail to identify groups of classifiers that together have a strong relationship with the response. For example, two classifiers may not be highly correlated with the response; but, their interaction may be correlated. Univariate

correlations will not capture this predictive relationship. A good idea would be to further investigate these aspects of the classifiers instead of using the rankings as the only method to understand the trends. Knowing which relationship to look at sometimes requires domain knowledge about the data.

When there are two classes, one approach to use is the area under the ROC curve [78] to quantify classifier relevance. Here, the classifier data are used as inputs into the ROC curve. If the classifier could perfectly separate the classes, there would be a cutoff for the classifier that would achieve a sensitivity and specificity of 1 and the area under the curve would be one. Figures 4.11, 4.12, 4.13, 4.11, 4.15, 4.14, 4.16, 4.17 and 4.18 show the area under the curve for all the supervised classifiers.

Machine learning models which were discussed previously were used on different combinations of features. First the webpages were classified from just content features, then other types of features were added and the gain was re-examined. It was found that the highest accuracy is obtained by combining URL, page-link, semantic TFIDF and SURF features. This combination of features was used as the optimal feature configuration. Finally the machine learning techniques were trained on data sets with varying ratios of malicious and safe webpages. As mentioned in earlier, 70% of the labeled webpages were used for training and 30% for

testing. The ratio of malicious to safe webpages is the same in testing as training for the supervised machine learning techniques. The supervised classification performance were evaluated in terms of precision and recall.

4.2.3 Supervised techniques

Figures 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9 and 4.10 show visual representations of webpages which are safe and malicious, after supervised classification was carried out. There are clear separations between the two types of webpages and the diagrams illustrate the nature of decision boundaries of various classifiers. The intuition conveyed by these figures does not resemble the larger datasets. In high-dimensional spaces, linear classifiers easily separate classes. Table 4.1 shows the results of the supervised techniques. The accuracy for all the supervised models improved as the number of webpages increased. Overall, SVM outperformed the rest. When SVM is considered, the accuracy values for the word-based document representation are remarkably low in case of a small number of webpages. As soon as the number of webpages exceeds 500, the accuracy increases. With other models, a similar trend is observed at the beginning. But then as the number of webpages increased, the accuracy also increased. The results suggest that the models were able to generalise better as more patterns emerged from various

sources.

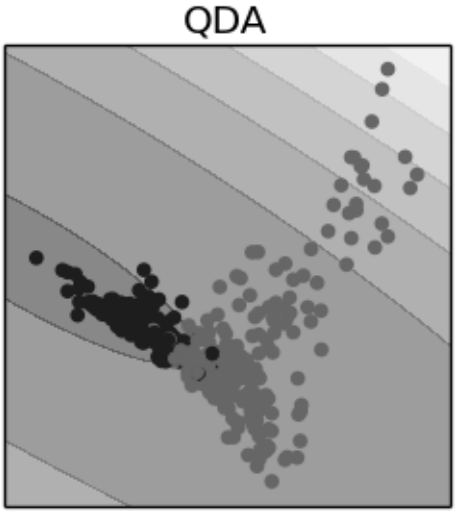

Figure 4.3: Visual representation of QDA model show clear boundaries between malicious and safe webpages

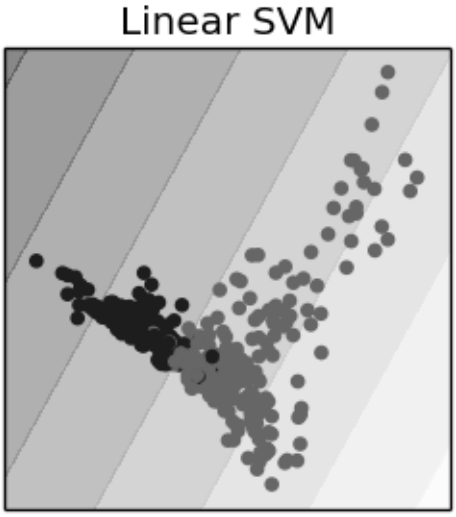

Figure 4.4: Visual representation of Linear SVM model show clear boundaries between malicious and safe webpages

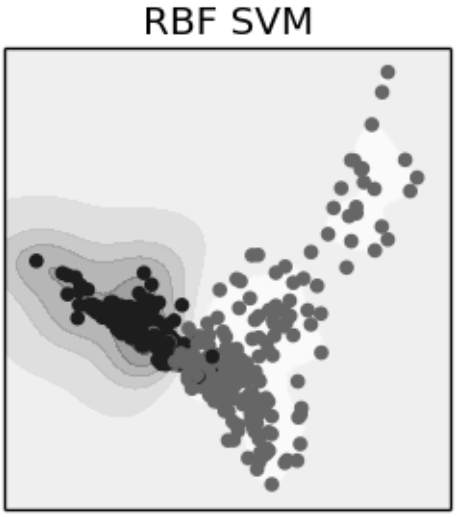

Figure 4.5: Visual representation of RBF SVM model show clear boundaries between malicious and safe webpages

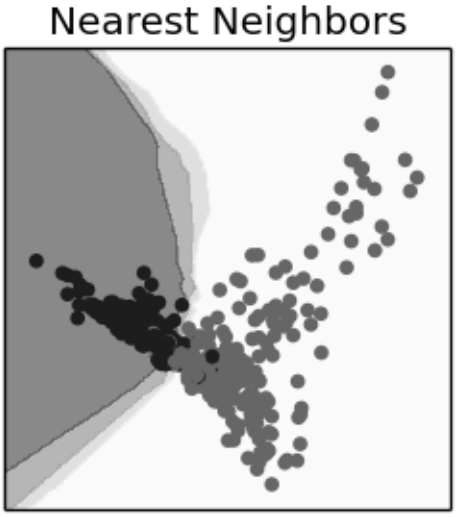

Figure 4.6: Visual representation of Nearest Neighbour model show clear boundaries between malicious and safe webpages

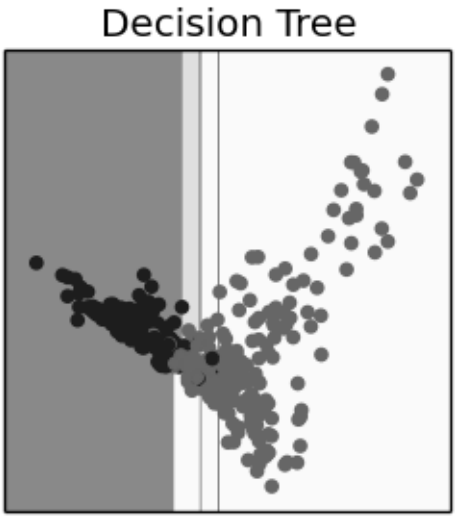

Figure 4.7: Visual representation of decision tree model show clear boundaries between malicious and safe webpages

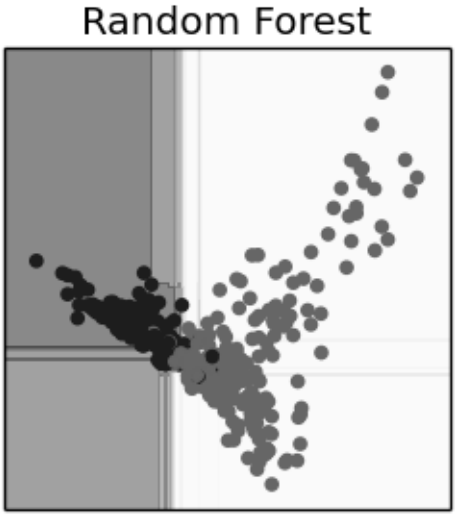

Figure 4.8: Visual representation of Random Forest model show clear boundaries between malicious and safe webpages

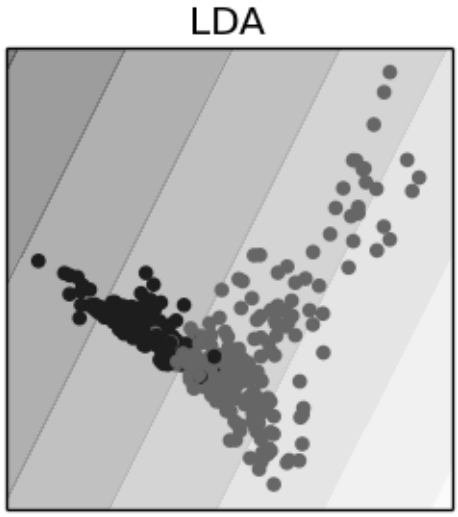

Figure 4.9: Visual representation of LDA model show clear boundaries between malicious and safe webpages

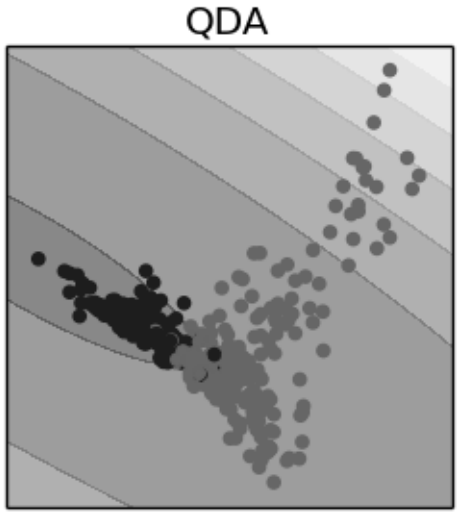

Figure 4.10: Visual representation of QDA model show clear boundaries between malicious and safe webpages

Figure 4.11, 4.12, 4.13, 4.11, 4.15, 4.14, 4.16, 4.17 and 4.18 show the Receiver Optimistic Characteristic of the supervised machine learning techniques and it is clear that SVM performs the best out of the four. K-Nearest Neighbour performs the worst, because it had less access to training data due to memory constraints. The supervised models were also run against one of the most popular datasets provided by [8]. The data file is SVM based and therefore the data was converted into recognisable for it for to be fed into the machine learning models. All the supervised machine learning models scored over 90%. Table 4.2 shows the results of the simulations.

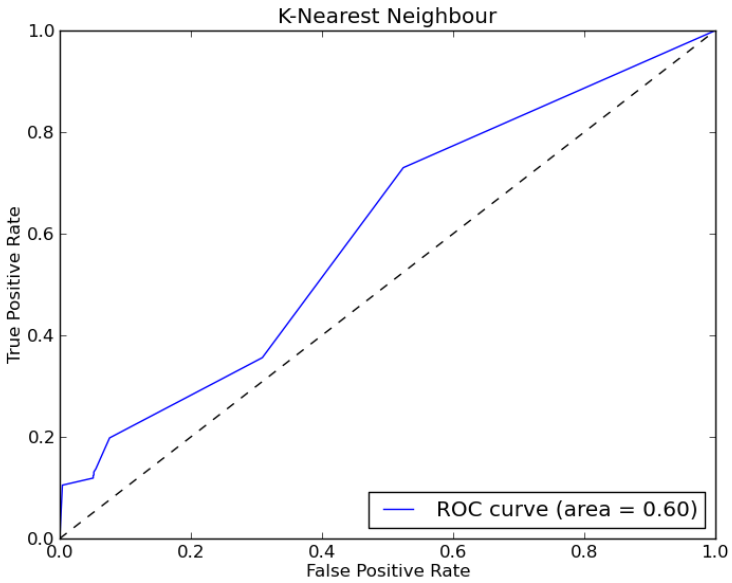

Figure 4.11: K-Nearest Neighbour's ROC curve is just above average.

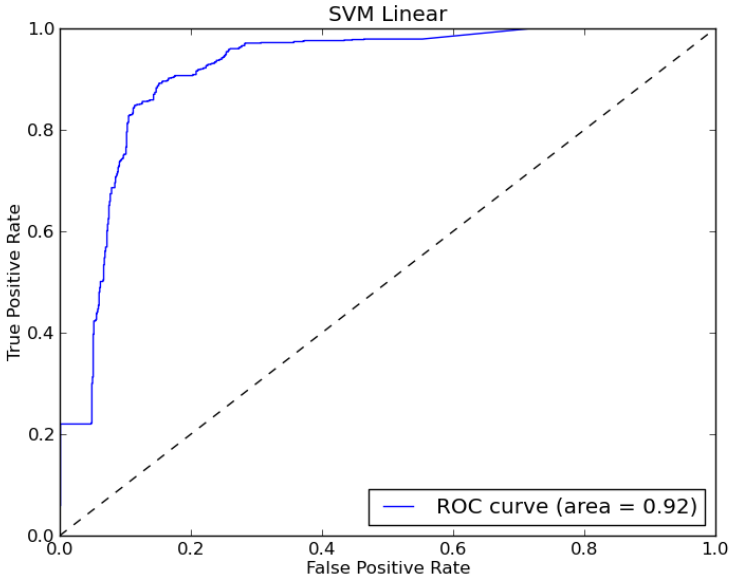

Figure 4.12: SVM Linear's ROC curve very close to 1.

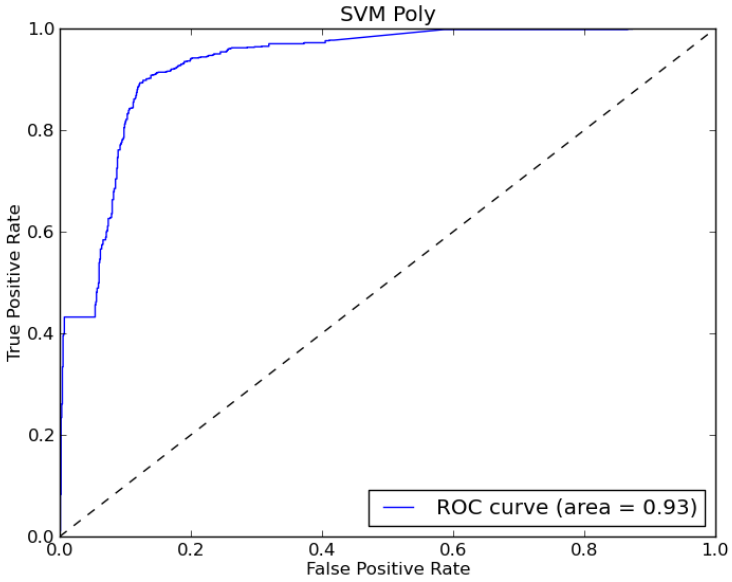

Figure 4.13: RBF SVM's ROC curve is also close to 1.

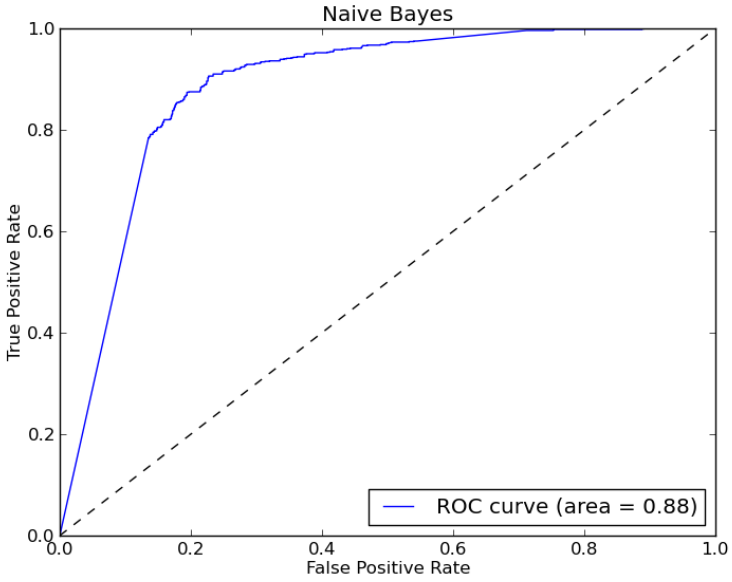

Figure 4.14: Naïve Bayes's ROC curve is just behind the SVM.

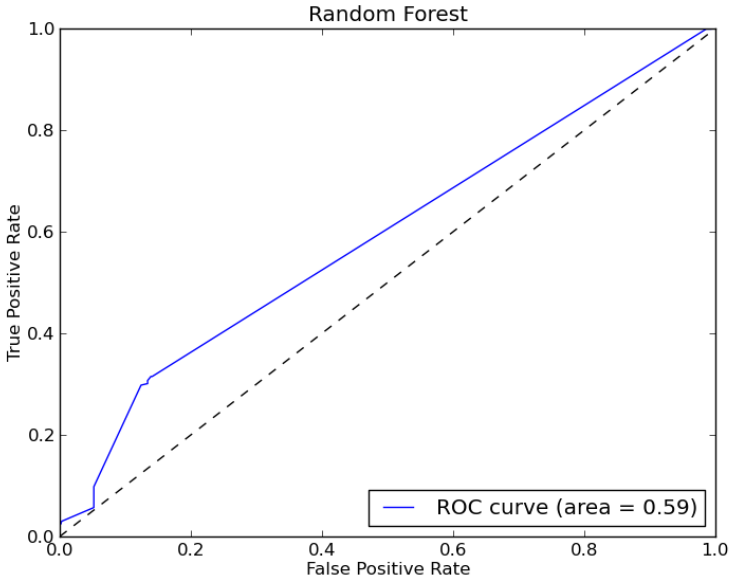

Figure 4.15: Random Forest's ROC curve is just above average.

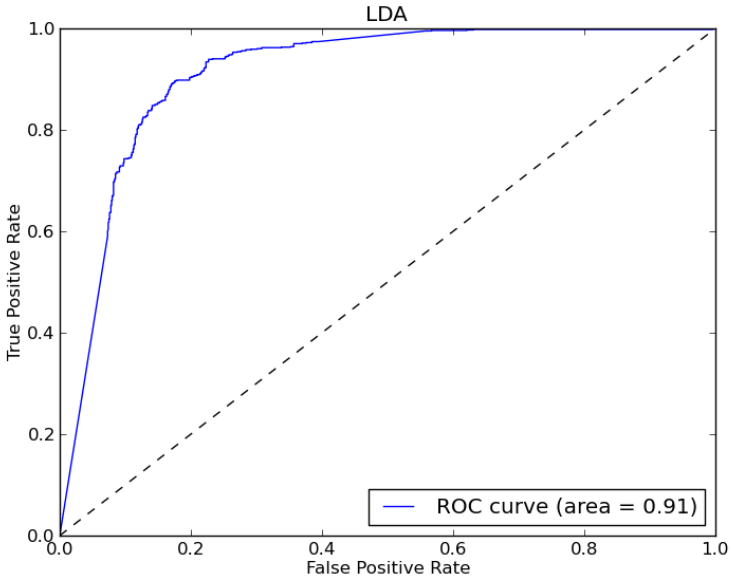

Figure 4.16: LDA's ROC curve is quite close to one.

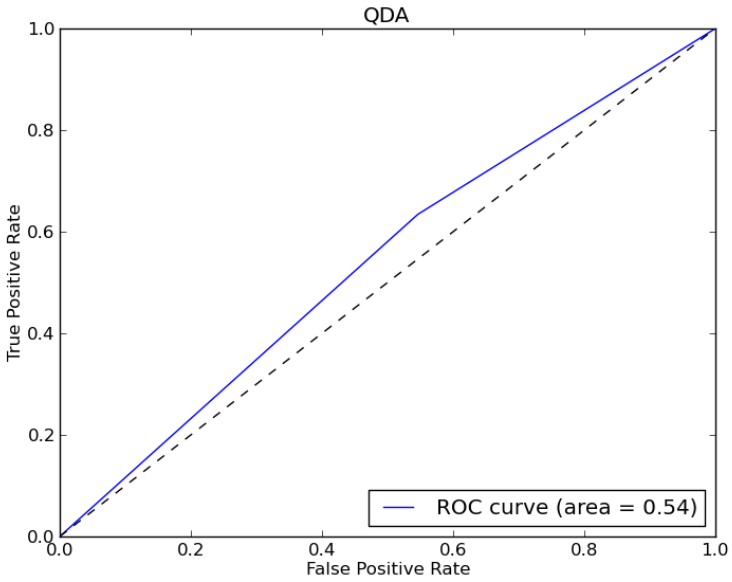

Figure 4.17: QDA's ROC curve is average.

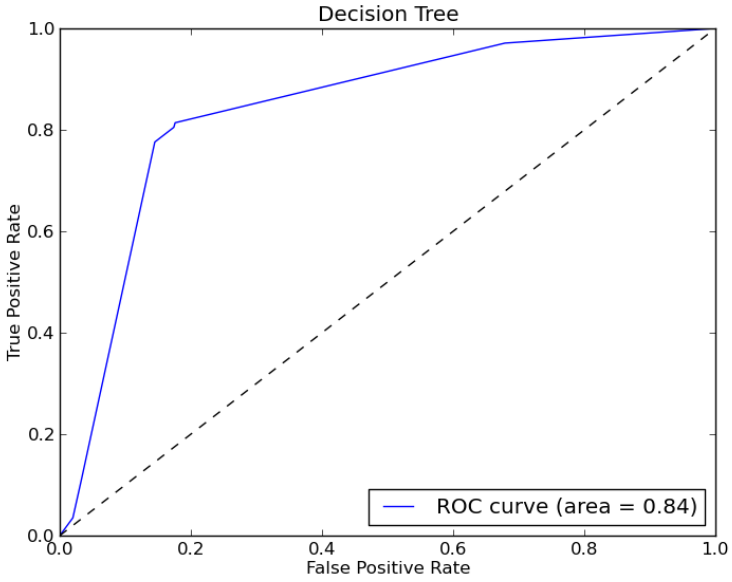

Figure 4.18: Decision Tree's ROC curve is good.

The previous ROCs do not have cross validation and it is important to see what effect cross validation can have on the results. Figures 4.19, 4.20, 4.21, 4.19, 4.23, 4.22, 4.24, 4.25 and 4.26 provide ROCs with cross validation. The figures show that the results are not very different to that without cross validation.

Table 4.1: Results of comparisons of supervised machine learning techniques that detect malicious webpages

No. of webpages	RBF SVM	Linear SVM	NB	KNN	RF	DT	QDA	LDA
50	80	79	77	74	77	67	56	58
100	82	83	78	75	77	70	67	67
500	86	92	78	79	77	75	67	68
5000	93	97	84	91	78	77	68	68
100000	93	98	89	95	80	79	68	68

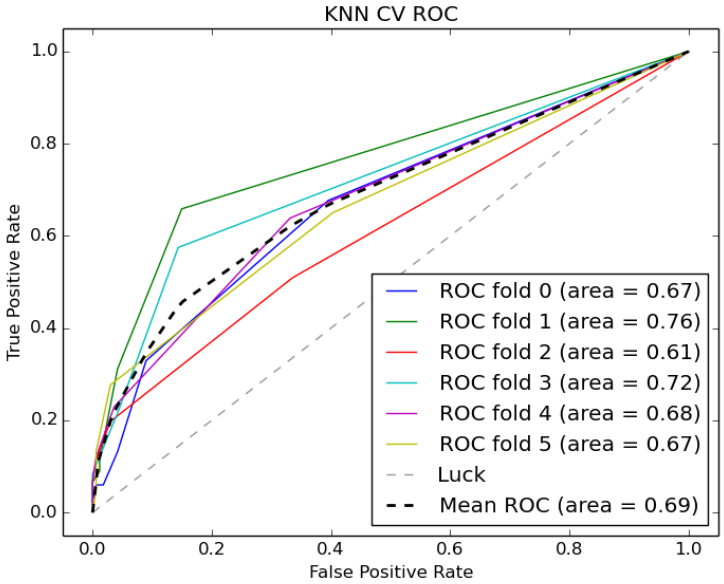

Figure 4.19: K-Nearest Neighbour's cross validated ROC curves are just above average.

Figures 4.27, 4.28, 4.29, 4.27, 4.31, 4.30, 4.32, 4.33 and 4.34 provide confusion matrices for all the supervised techniques.

4.2.4 Online learning

Online learning uses a different approach than the traditional batch processing which cannot learn incrementally [79]. The problem tackled in this thesis used data from an incoming list of malicious webpages and safe webpages. This allowed the predictive models inside WAC to train automatically as new data came in.

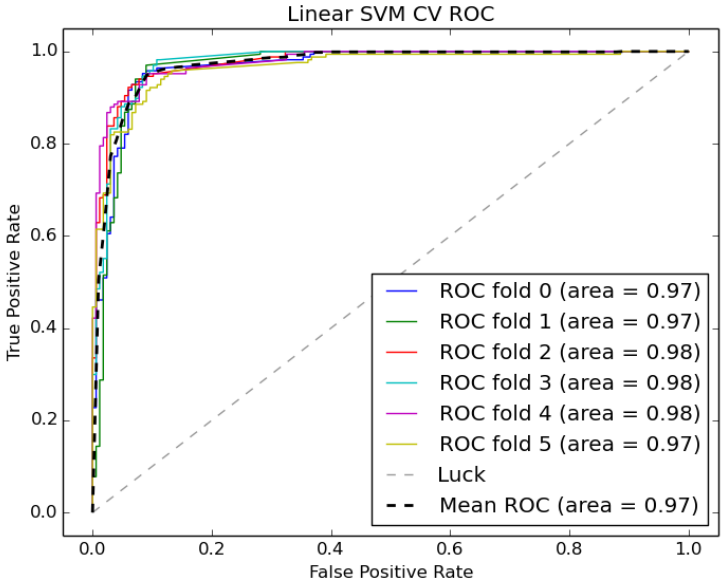

Figure 4.20: SVM Linear's cross validated ROC curves are very close to 1.

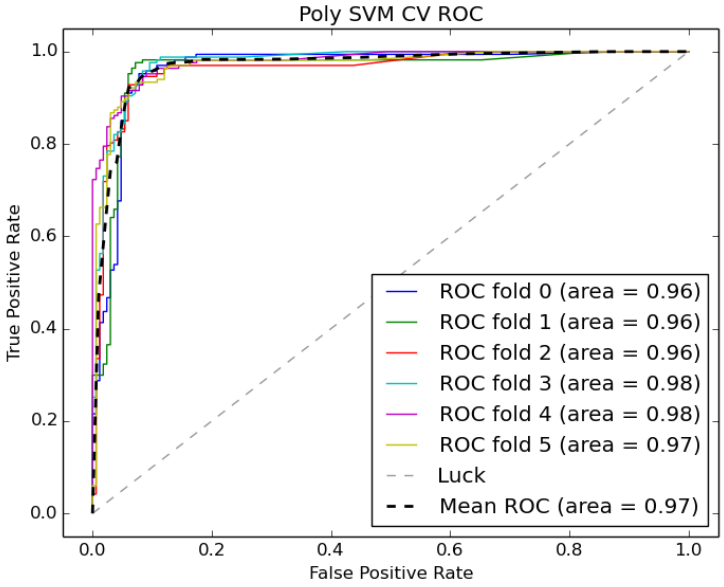

Figure 4.21: SVM Poly's cross validated ROC curves is also close to 1.

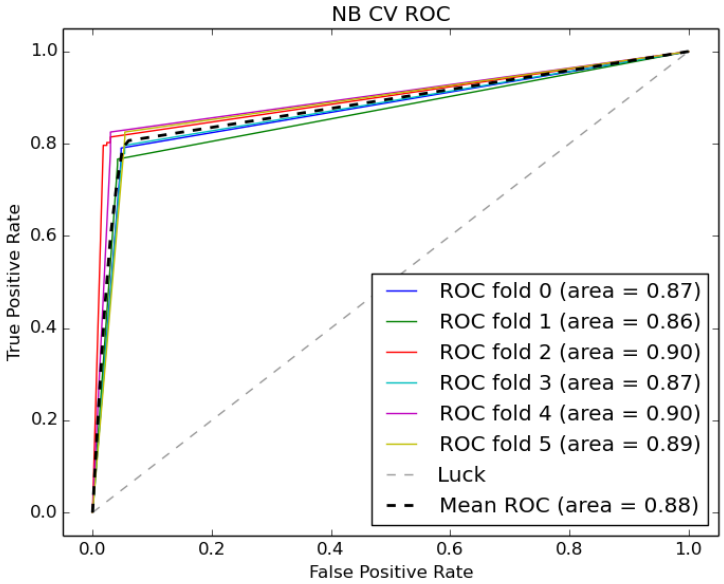

Figure 4.22: Naïve Bayes's cross validated ROC curves are just behind the SVM.

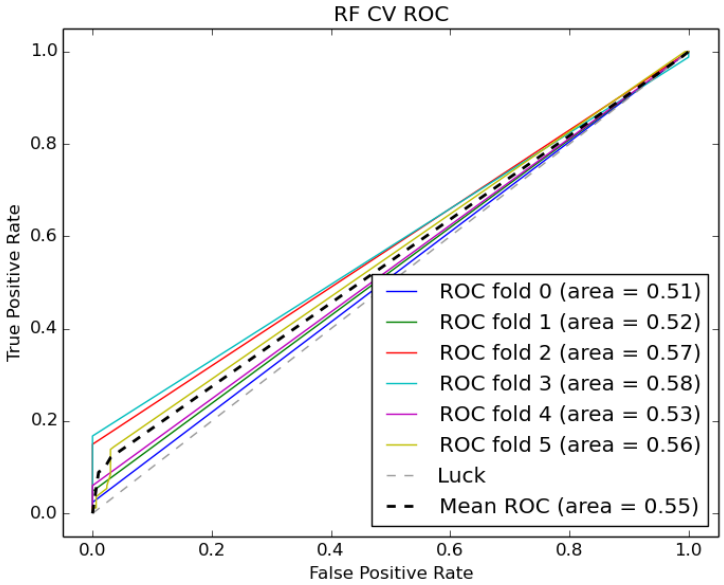

Figure 4.23: Random Forest's cross validated ROC curve are just above average.

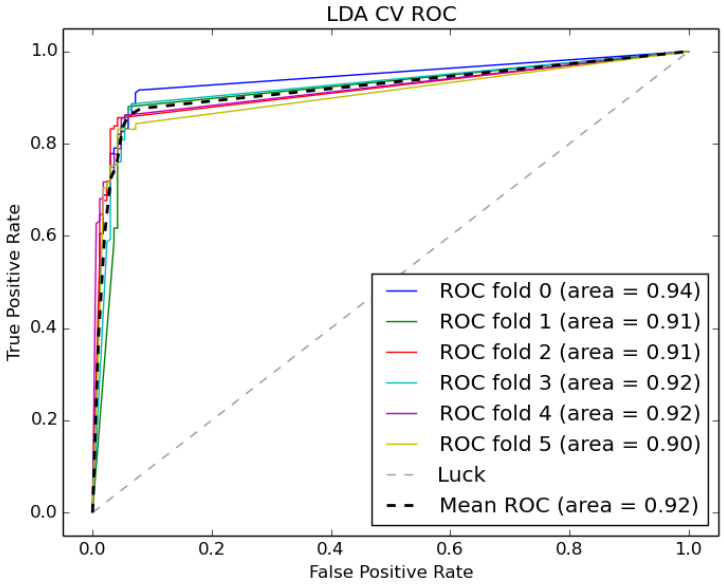

Figure 4.24: LDA's cross validated ROC curve are quite close to one.

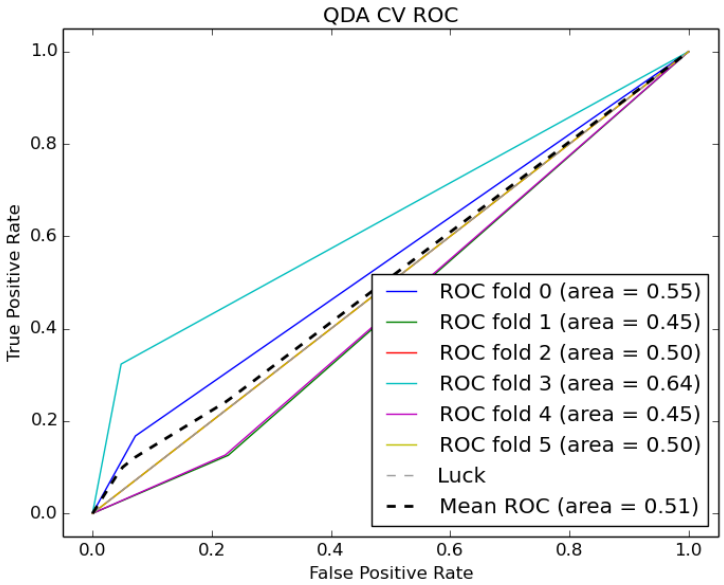

Figure 4.25: QDA's cross validated ROC curves are average.

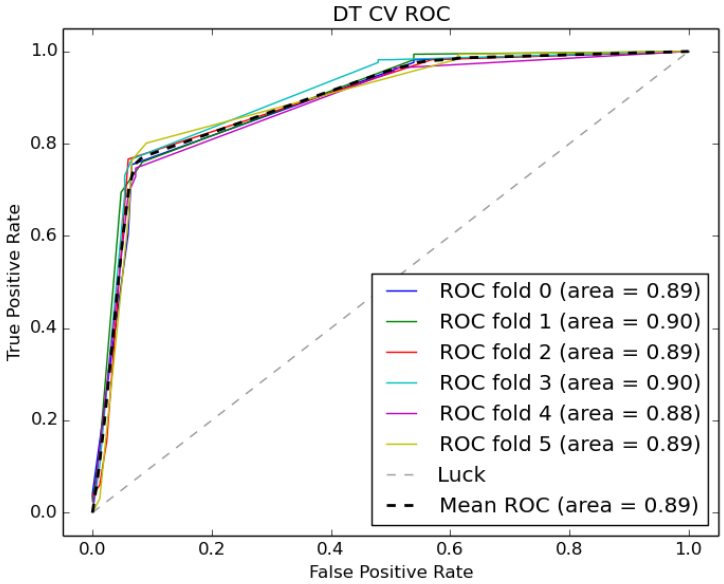

Figure 4.26: Decision Tree's cross validated ROC curves are good.

Generally the batch machine learning techniques optimise Equation 4.1.

$$f(\theta) = \frac{1}{N} \sum_{i=1}^{N} f(\theta, \mathbf{z_i}) \tag{4.1}$$

where $z_i = (x_i, y_i)$ in the supervised case and $f(\theta, \mathbf{z_i})$ is some kind of loss function. For example, it is possible to use $f(\boldsymbol{\theta}, \mathbf{z_i}) = \log p(y_i | \mathbf{x_i}, \boldsymbol{\theta})$ to maximise the likelihood.

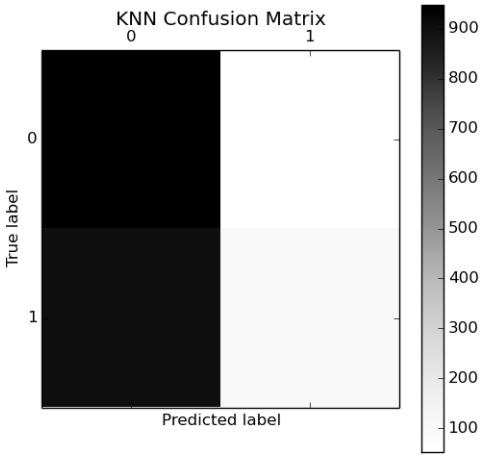

Figure 4.27: K-Nearest Neighbour's confusion matrix.

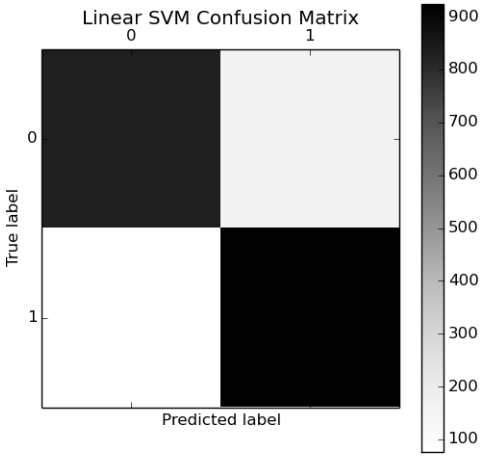

Figure 4.28: Linear SVM's confusion matrix.

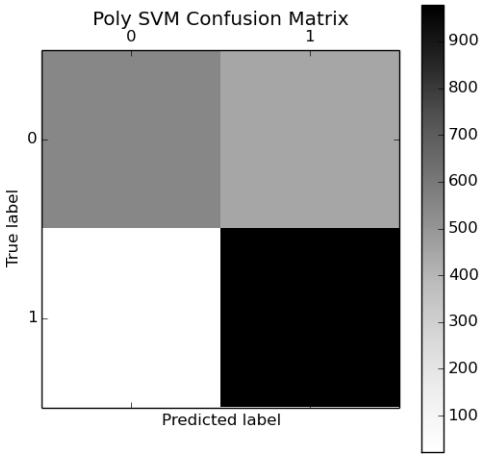

Figure 4.29: RBF SVM's confusion matrix.

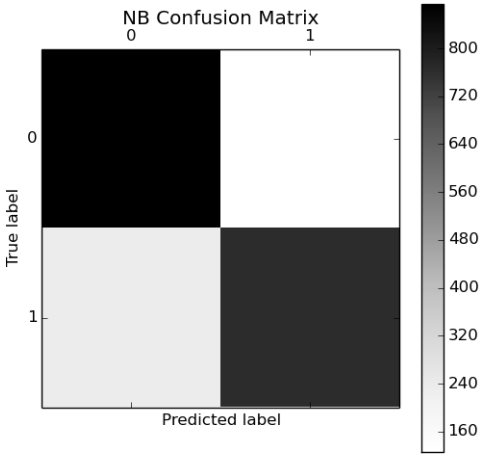

Figure 4.30: Naïve Bayes's confusion matrix.

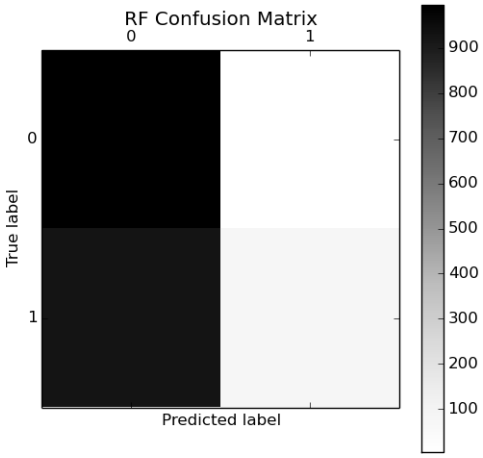

Figure 4.31: Random Forest's confusion matrix.

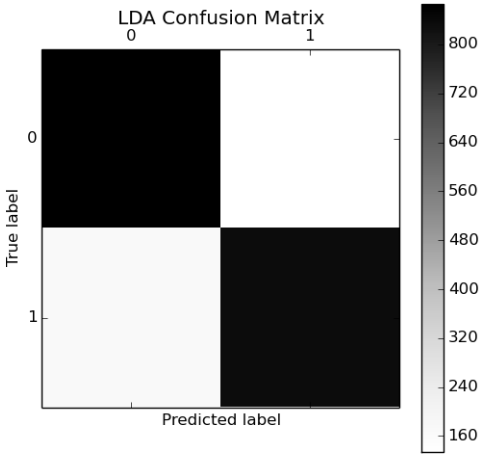

Figure 4.32: LDA's confusion matrix.

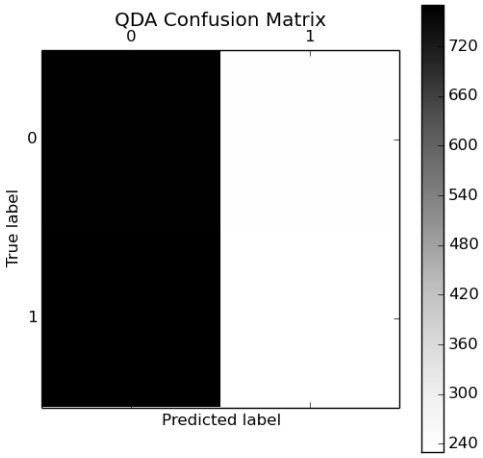

Figure 4.33: QDA's confusion matrix.

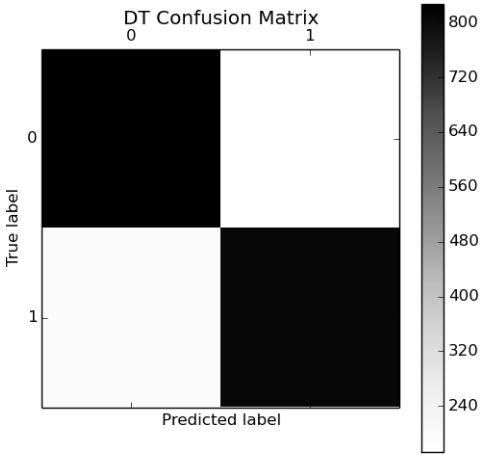

Figure 4.34: Decision Tree's confusion matrix.

4.3 Chrome extension

4.3.1 Introduction

Recent developments in the web browsers allow users to take benefits from various browser extensions. Unfortunately some of the extensions have been the major reasons for security issues. But most of the extensions are useful. Chrome extension are known for their secure environment [80, 81]. There have been research on Chrome applications extensively [82]. The thesis covered various machine learning techniques but no implementations for practical case so far has been presented. This section goes through the details of a Chrome extension that uses the most successful model of the all the supervised models described before.

4.3.2 Architecture of the Chrome extension

The Chrome extension has been built using the architecture shown in Figure 4.37 using the following steps.

- **Step 1:** Grab the webpages as soon as the user loads the webpage
- **Step 2:** Extract the features and then send the features to the predictive models. The predictive models then send a response

- **Step 3:** Content Script notifies the background script with the response
- **Step 4:** Background script shows the response.

4.3.3 Chrome extension's user interface

The user interface for the Chrome extension has various options. Figure 4.35 shows the possibilities for the Chrome extension. The first option is to display an icon in the top right hand corner. The second option is to add the user interface to the context menu. But this option is not suitable because the user has to interact only after the webpage has loaded. The last option is to present a user interface in the Chrome context menu, with an options page, or use a content script that changes how pages look. This last option involves interaction from the user and so will take time to give feedback to the user. So the first option is the most suitable one.

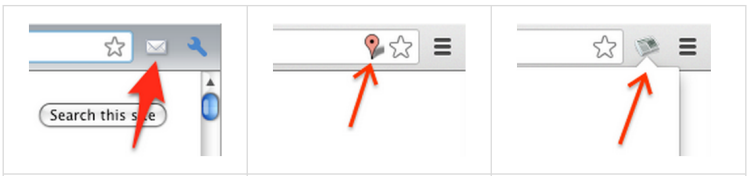

Figure 4.35: The three pictures demonstrates various examples of Chrome extensions [7]

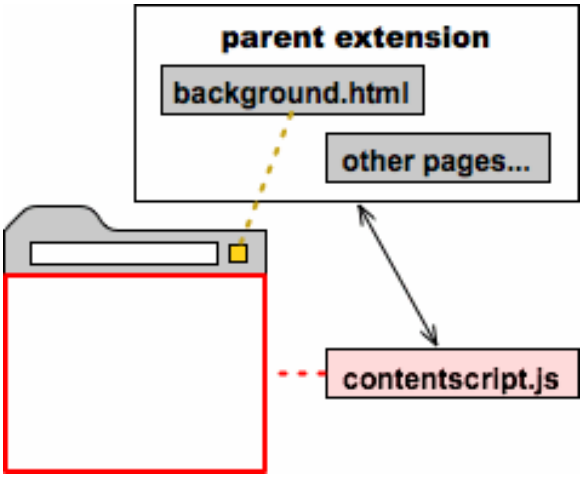

Figure 4.36: A chrome extension with its various parts and their relationships to each other[7]

4.3.4 Chrome extension composition

The extension has various files out of which the required files are manifest files combined with one or more HTML files. Also, there are JavaScript files and image files. The JavaScript files have code that can determine the functional aspects of the extension. The image files are needed for the user interface. Before building and distributing the extension the files mentioned above were then put in a single folder. A zip program builds a special zip file with a .crx extension. The manifest file, called manifest.json, stores information about the extension. For examples it stores the list of important files and lists the capabilities that the extension may use.

Table 4.2: Results based on a different dataset provided by [8]

Classifier	Accuracy - no. of true positives (%)
Naïve bayes	91
Support Vector Machine (RBF)	97
Support Vector Machine (Linear)	92
K-Nearest Neighbour	85
Decision Tree	86
Random Forest	83
LDA	85
QDA	82

4.3.5 Chrome extension's architecture

The Chrome extension has a background page which is an invisible page containing the main logic. If the extension needs to interact with webpages that the user loads, then the extension uses a content script. Each action of an extension has a background page (see Figure 4.36) which is defined by background.html and has JavaScript code that controls the behaviour of the browser action. Extensions can contain ordinary HTML pages that display the extension's UI. The extension interact with webpages via a content script (see Figure 4.36) which is some JavaScript executing in the context of a loaded page within a web browser. Content scripts aren't completely cut off from their parent extensions. A content script can exchange messages with its parent extension whenever necessary.

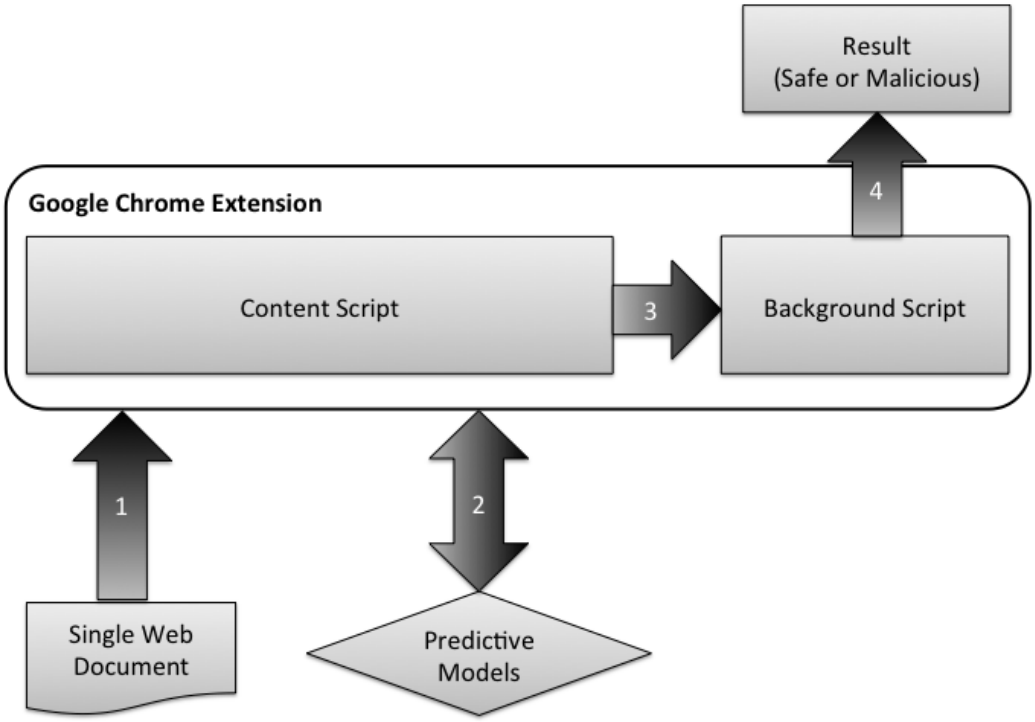

Figure 4.37: The Chrome extension uses 4 steps to decide whether a webpage is malicious or not.

Figure 4.38: The Chrome extension shows that the webpage is safe.

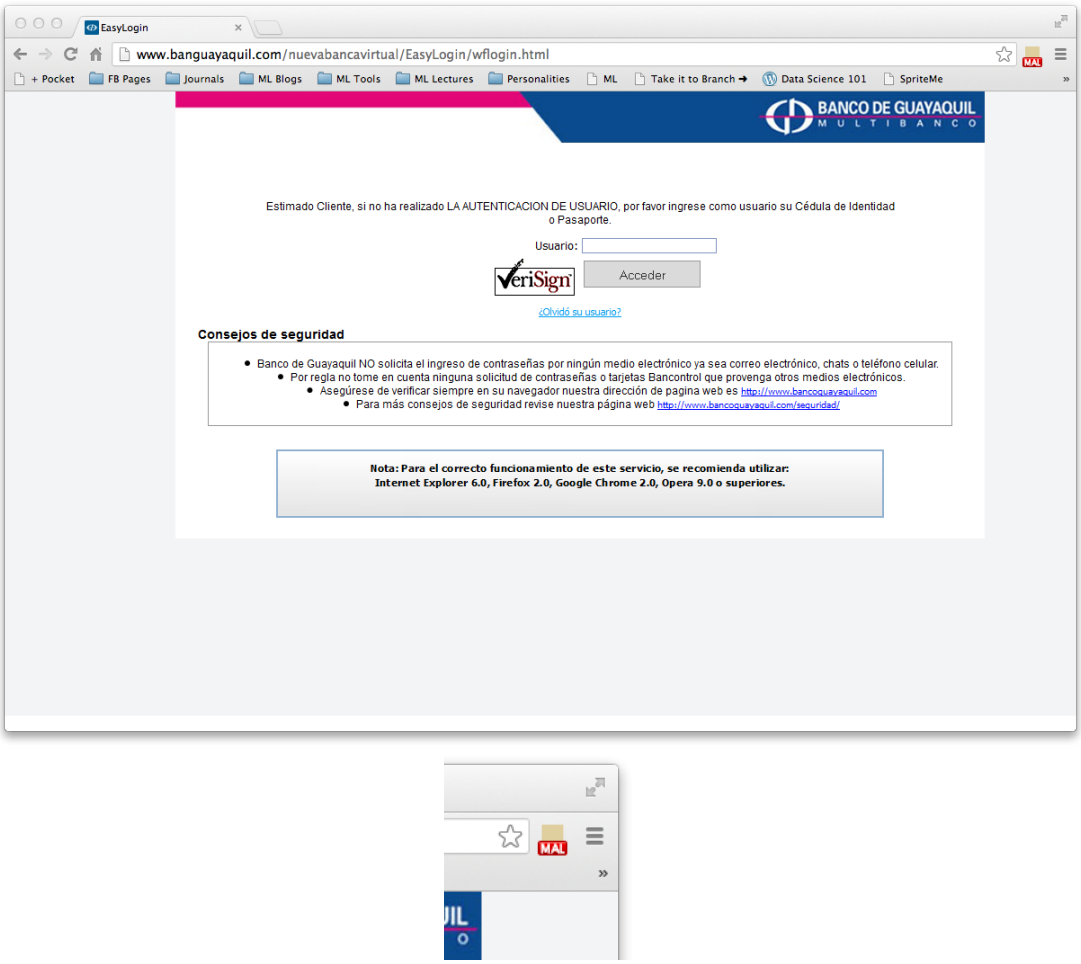

Figure 4.39: The Chrome extension shows that the webpage is malicious.

4.3.6 Results from the Chrome extension

The content script looks at the loading document and sends the loaded source code to the predictive classifier. The classifier then parses, creates the features and then responds with whether it thinks that the webpage is safe or malicious. The background script receives the response and shows whether it is malicious (see Figure 4.39) or safe (see Figure 4.38). 'Heavyweight' classifiers are more accurate but has a poor prediction time as they load the same page within their 'environment', but use more features and so has a higher accuracy. 'Lightweight' classifiers does the opposite i.e. use less features and use the features only available via the browser. The Chrome extension gets all the features from the browser and sends them to the classifier which uses more features, thus have a quick prediction time and yet higher accuracy.

4.4 Summary

This Chapter dealt with all the supervised models and have shown that the results are encouraging. Also, it has been shown that the best performing supervised model can be used for practical purposes as a Chrome extension. This extension determined whether a webpage is malicious or not in real time. The next Chapter

deals with simulations with unsupervised machine learning techniques.

CHAPTER 5

Computer simulation results using unsupervised machine learning techniques to detect malicious webpages

One of the issues that was encountered in the last Chapter (although this was not mentioned) is the execution time to train the supervised models. This Chapter initially deals with this particular problem by using MapReduce to improve the efficiency of an unsupervised machine learning technique i.e. Self-Organising Map and then uses various other unsupervised machine learning techniques to cluster webpages.

5.1 SOM

One of the most used algorithm for unsupervised categorisation technique is SOM [83]. This Chapter makes use of SOM to group webpages into clusters and the approach is made more effective by eliminating one of its fundamental problems i.e. slow speed, by using MapReduce programming model [84]. SOM has been widely used both in the data mining and artificial intelligence community [85, 86]. Although this topic of clustering documents via SOM has been handled in [87] it was based solely on the users' navigational behaviour. The most comprehensive coverage of SOM has been done in [88], but the documents have been journals and not webpages. Although both journals and webpages are quite similar in content presentation, there is one fundamental difference i.e. additional HTML tags are

present in webpages. Therefore, in order to capture or understand the readable contents from webpages, some special measures are needed i.e. the webpages are parsed first and the content is chosen from large amounts of HTML tags.

To improve SOM's speed multiple computers have been used in [89], which used a Beowulf cluster based on Linux boxes. The time for processing the SOM reduced to a large extent. But this system is prone to hardware failures. Continuing from Chapter 2, the update formula for a neuron of a SOM with weight vector $W_v(t)$ is

$$W_v(t+1) = W_v(t) + \Theta(v,t)\alpha(t)(D(t) - W_v(t)) \tag{5.1}$$

where $\alpha(t)$ is a decreasing learning coefficient and $D(t)$ is the input vector. $\theta(v,t)$, the neighbourhood function is dependent on the distance between the BMU and neuron v. The SOM maps associate output nodes with groups or patterns in the input data set. One single neuron wins whose weight vector is closest to the input vector. SOM does not use a threshold value but rather selects a winning output neuron when a pattern is entered. In general, Neural Networks are trained in a supervised mode, but the SOMs are trained in an unsupervised mode. Neural network can only be applied to certain types of problems i.e. it takes values

between -1 and $+1$ and the inputs are within the range. SOM's inputs are normalised either with Multiplicative Normalisation or Z-axis normalisation [50].

5.1.1 SOM's training process

A initial higher learning rate results to a quicker training process because it decays over time, but SOM may fail if it is too high. This is because the random movements of the weight vector exceed the allowed threshold for any pattern to be determined. The other method is to use a relatively high learning rate and reduce it as the training progresses. This trains the SOM very quickly at the beginning and is then controlled while the training continues. SOM uses the weighted connections between the input layer and the output layer as its storage. With each iteration, the weights are calibrated. These calibrations produce a SOM network that returns better results, when the same training data is presented to it next time. More data is presented to SOM network as iterations continue, and the weights are calibrated. But at some point the calibration stops because the particular set of weights are no needed and at this point, the entire matrix is reset to new random values, and a new learning cycle begins. The final weight matrix is then taken as the best weight matrix from each of the cycles. As mentioned before, the SOM trains in an unsupervised fashion and the output is

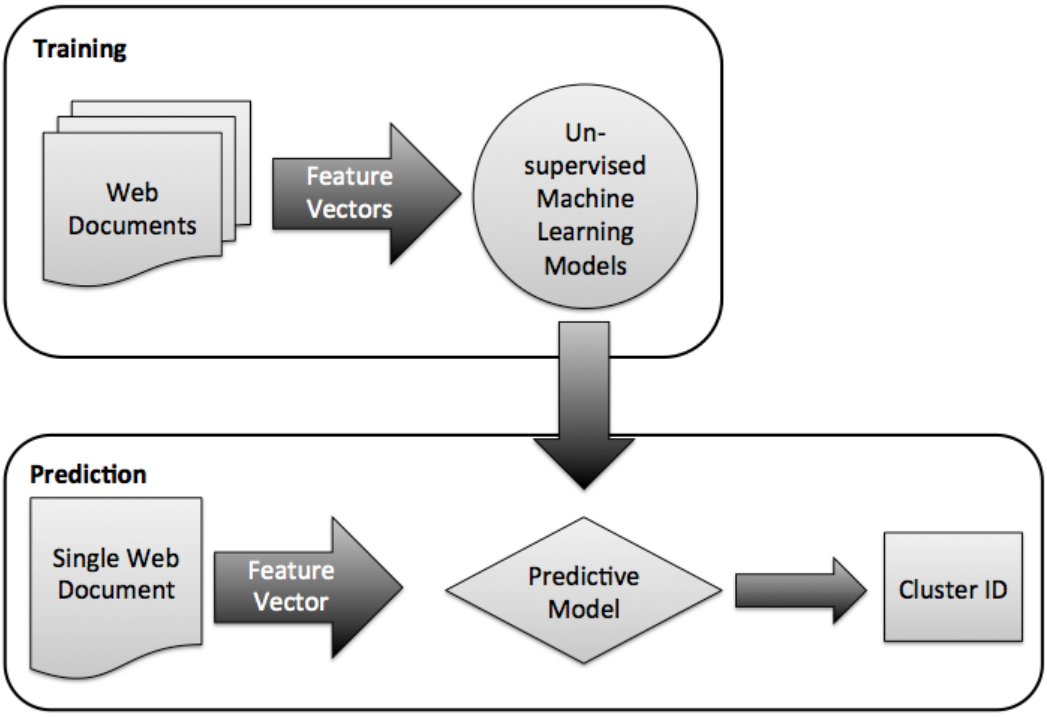

Figure 5.1: Architecture for classifying webpages using unsupervised techniques

normally anticipated, but the expectation is that outputs new groups. Summing up the above discussion, when SOM is applied to webpages, the process depicted in Figure 5.1 is used. The webpages are first converted into feature vectors and are then fed into SOM network which then produces a predictive model. This predictive model is then used by each single webpage to place them into one of the two categories i.e. safe/malicious.

5.2 Datasets

For this thesis, DMOZ Open Directory Project [90] and Yahoo's random URL generator [91] have been used as the sources for safe webpages. For unsafe webpages, OpenDNS [92] and Spamscatter [93] have been used. From these sets of webpages, the real-valued features in each feature has been normalised and lie between 0 and 1. Values outside this range in the testing set were converted to zero or one as appropriate. The normalisation equalised the range of the features in each feature set, both real-valued and binary. One further complication arose due to undefined, or missing, features which were handled by using an extra binary feature which defined to indicate whether the feature was defined.

5.3 Webpage clustering and WAC

Webpage clustering assigns webpages to a set of categories. Rather than trying to simply assign webpages to pre-defined categories using a labelled set of data for training, clustering algorithms discover distinct categories using an unlabelled set of data. Webpages in the dataset are then assigned (often as a by-product of the clustering process) to these newly discovered categories. Webpage clustering

can be defined as following: a set of D of m webpages, denoted by $d_1....d_m$. Each partitioned webpage is assigned to a soft partition in which each webpage is probabilistically assigned to multiple clusters. Hard partitions can simply be thought of as a case assigned to the single cluster for which it has the greatest probability. The clustering task has two important components. The first is determining how many clusters to partition the data into (i.e. choosing K). The second is how to assign each webpage to these respective clusters.

WAC categorises a webpage into various clusters and thus will allow to classify them as malicious or safe. The algorithm for the WAC system is shown in Figure 5.2. The next sections will look at how to parallelise SOM implementation.

5.4 How MapReduce works

MapReduce is a programming model used by Google [9] in many of its products. Google has also developed an implementation for this model. This model takes large amount of input and distributes the processing among huge numbers of cheap computers. This allows it to complete processing within short period of time compared to processing running on a single processor. But the implementation which deals with parallelising computations, distributing data and also

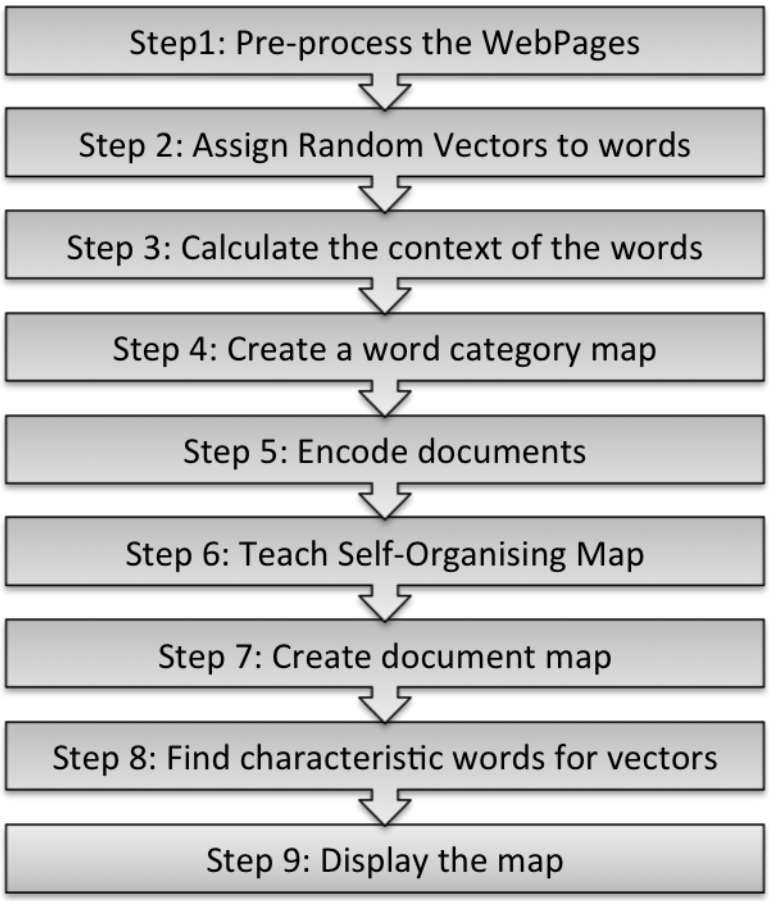

Figure 5.2: Steps involved in the WAC tool.

tackling failures with hardware are quite complex and so an abstraction level has been created in MapReduce. This abstraction layer reduces the complexity to the developer.

MapReduce [84] is resistant to hardware failures which are normal for normal workstations (without RAID support). Figure 5.3 shows the overall flow of a MapReduce operation. A MapReduce program initially shards the input files into M pieces (which is equivalent M map tasks) of typically 16 megabytes to 64

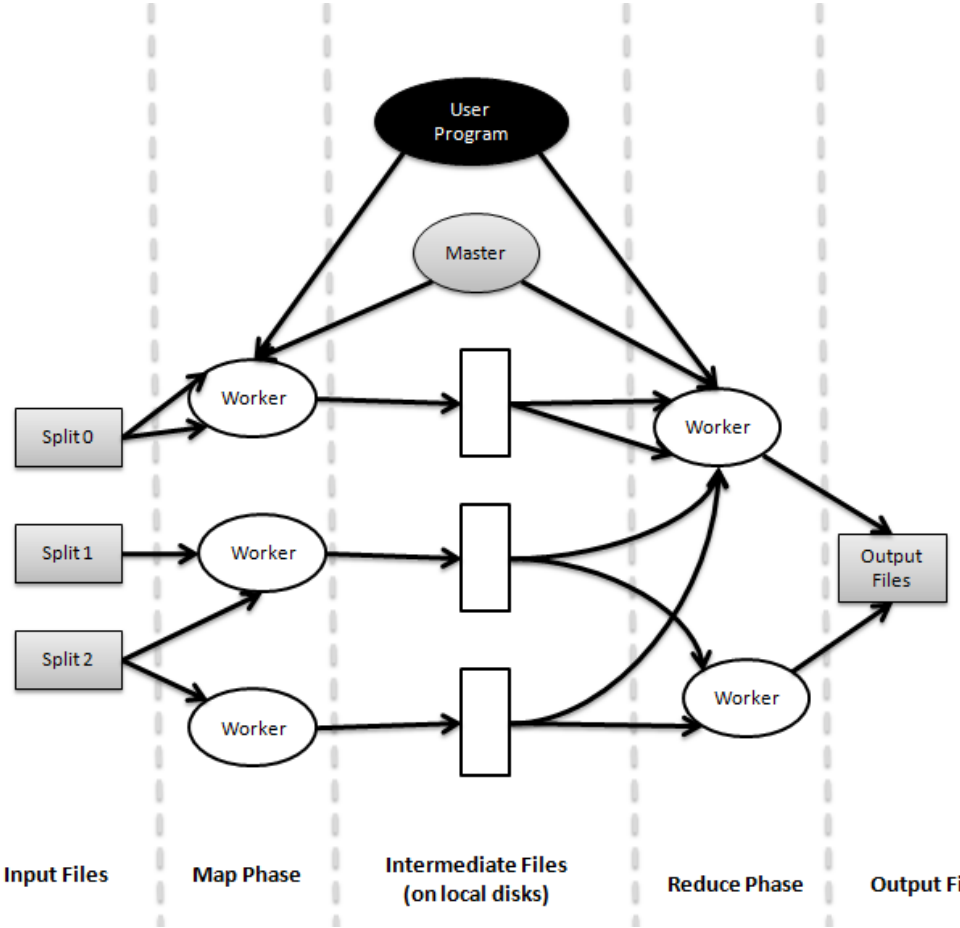

Figure 5.3: Overall view of MapReduce. The master node assigns the workers (also referred as nodes).

megabytes per piece. It then starts up many copies of the program on a cluster of machines. One of the copies is referred as master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task. A worker who is assigned a map task reads the contents of the corresponding input shard. It parses key/value pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory. Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers. When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. If the amount of intermediate data is too large to fit in memory, an external sort is used.The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce

function is appended to a final output file for this reduce partition. When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

5.5 Hadoop

Hadoop is based on HDFS (Hadoop File System) and the implementation takes ideas from Google File System [94]. It has been used in large organisations such as Facebook [95], Twitter [96], Linkedin [93] etc. This is a distributed file system for applications that use computationally intensive applications and works with large amounts of data. This file system is extensively used by Google as the primary storage mechanism.

5.6 Beowulf cluster and its problems

This Beowulf cluster implementation is a parallel computation model used only on Linux based machines [97]. Any programs written and then run on a cluster usually runs faster. But the program has to take care of the hardware failures and the parallelism involved in the cluster. There is no room for fault-tolerance, error

detection and work restart capabilities. This is probably a good solution for time-boxed applications that demands reliable and timely execution of a particular task e.g. finds the Euclidean distance between two points. Moreover this cluster does not consider manageability which forces the programmer to manage each resource separately in the cluster rather than a single file system. Firstly, the parallel applications under a Beowulf cluster use message passing model rather than shared memory. Secondly, Beowulf cluster focuses on developers and does not take architectural model, testing and binary compatibility into account. This leads to writing the application possibly being written again to take advantage of clustering in order to make any significant changes to the program. And lastly, the developer is often responsible for system design and administration, which takes time and energy away from work on the actual application.

5.7 The chosen solution

Comparing the two algorithms, MapReduce has an upper hand in terms of parallelism. One of the downside of MapReduce is that it restricts the programming model. But the counter argument is that it provides a good model for managing problems dealing with large datasets. And in particular problem with large

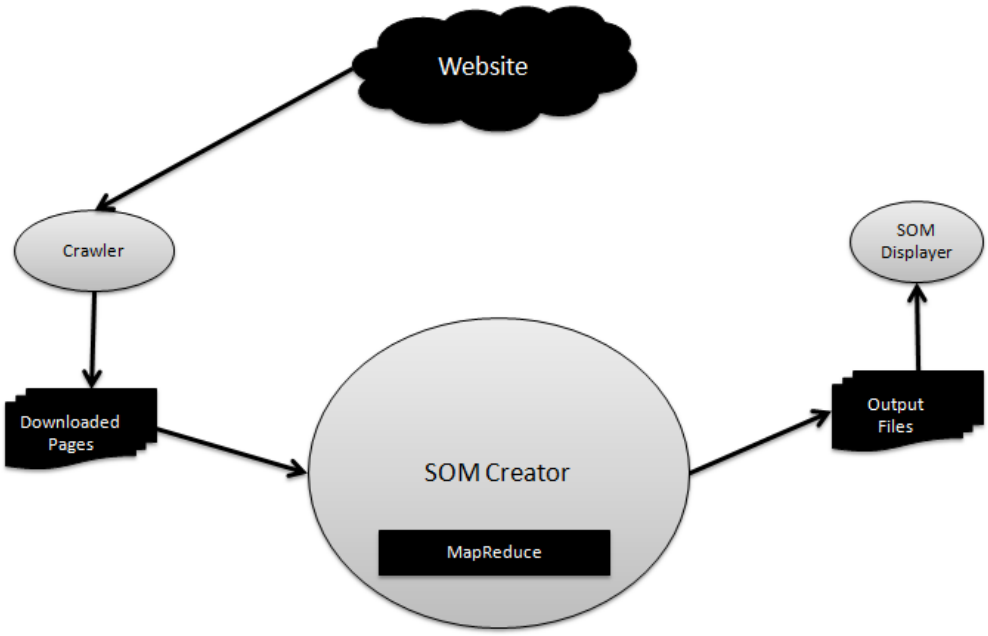

Figure 5.4: WAC implementation incorporating Map and Reduce

datasets, MapReduce provides fast execution without worrying about the underlying hardware infrastructure and rather are focused on the application i.e. solving the problem. Based on these facts mentioned in [98] it has been decided to use Hadoop which is a similar implementation to [84].

WAC has used SOM algorithm at first but in order to take advantage of the MapReduce, the algorithm had to be revised slightly. No changes have been made to the algorithm, but the way the calculations have been done has changed. An intermediate step has been added which acts as a buffer has been used. This buffer allows the system to be run the computation in parallel.

The main strength of the WAC system is that no matter how many webpages are in a website, each page is assigned a tag (see Figure 5.4 for WAC's architecture). Tags are used in blogs, forums, pictures and videos on-line for identifying and grouping of the similar content. Also, the distribution content inside the webpages of a website is vividly clear to the user as the topographic map represents 'concentration' and 'hollowness'.

5.8 Results

WAC consists of various tools which have been built . These tools have been integrated so that the product produces the output as a SOM. The reason WAC was designed with various tools is because it is easier to test the system separately and find bugs in the system separately. WAC consists of:

1. Crawler: This tool downloads all the webpages from a target webpage and stores the files in the local file system. This is done using a crawler that 'visit and downloads' the home page of the target webpage and retrieves all the links. Then the crawler repeats the same process i.e. downloads all the webpages referred by the links and so on. A list of the 'crawled webpages' is maintained so that the same is Webpage is not downloaded twice.

2. HTML Parser: Generally, HTML pages have tags around each element which are not necessarily part of the original content, and are not meant for users but for web browsers. These browsers use these tags to understand the layout and the formatting of the content. After the HTML Pages have been downloaded from the Web, the webpages are parsed using this tool. Parsing involves removing the tags from the HTML pages and retrieve text that is understandable and readable to a normal user.

3. SOM Creator: This is the heart of the WAC system and creates the SOM based on the techniques mentioned in [88] apart from the use of search. This creates a SOM which the users can browse where tags are tags are attached to the map at place of higher concentration of documents or commonly known as clusters.

4. SOM Displayer: This tool displays the SOM on a grid that allows user to click on nodes and the documents assigned to the node are shown. As the clicks on one of the link, a Web Browser opens showing the Webpage connected by the hyperlink.

Class	Use
Mapper	This class takes an input pair of values and produces an intermediate key/value pairs. The programming model groups the values and forwards the values to the reduce class.
Reducer	This class receives the intermediate values to a corresponding intermediate key. Generally, one or none output is produced per reduce method call. Sometimes the computer's main memory is not enough for large datasets. So an iterator is used instead for easy handling.
Driver	This class is the main program that contains the main method of the program. It sets the input and output folders and the configurations needed for jobs to run.

Table 5.1: Description of MapReduce Classes

5.8.1 Java Classes

The design of the implementation is composed of three classes i.e. Mapper, Reduce and the Driver classes. See Table 5.1 for more details. At the end of the job run, the results are aggregated by the reduce class and are used to display the final SOM. The same program can be theoretically run on many machines without any modification to the program and the test results demonstrate it clearly.

5.8.2 Software

The system has been implemented using the Java Programming Language [99]. Data has been downloaded from websites by the crawler (part of WAC) and the pages have been stored in the File System. The creation of SOM has been accelerated by using MapReduce. The algorithm used four machines. Each system consisted of 2 GHz processor and 2GB of RAM. The systems in cluster were networked using 100 Mbps Ethernet links. Table 5.2 shows how the implementations compare with the increasing number of nodes. The MapReduce model is used where the number of nodes is more than 1. Figure 5.5 shows the trend as the number of nodes increases. The system is faster than earlier implementation on a single processor and is now more scalable. Compared to Beowulf cluster, the system is more easily manageable and usable. Beowulf has the disadvantage of providing wrong results if one of the machines breaks down. On the other hand WAC also has the advantage of being interoperable in various operating systems as the system itself is written with Java. So, as long as the machine has a Java virtual machine, the WAC tool can operate without any problems on any Operating System. [84] has used MapReduce with great success at [9] and the success has been rediscovered in WAC as well with the revised 'MapReduced

Number of Nodes	Number of Webpages	Time to complete the test	Performance improvement
1	979	292	N/A
2	979	284	$\approx 2\%$
3	979	261	$\approx 18\%$
4	979	244	$\approx 19\%$

Table 5.2: Results for SOM

SOM algorithm'. The algorithm has been used as part of WAC, which now takes less time for categorisation.

5.9 Results from all the unsupervised techniques

Going back to the last Chapter, the same webpages are now fed into some unsupervised machine learning techniques i.e. Mini Batch K-Means, K-Means and Affinity Propagation. Figures 5.6, 5.7 and 5.8 show that the three unsupervised machine learning techniques clearly separate the malicious and safe webpages. The parameters for the Affinity Propagation had to be readjusted to get the desired number of clusters. Table 5.3 shows the results of the simulations. The table uses silhouette coefficient as the main factor to determine the effectiveness of the unsupervised algorithms. Silhouette coefficient (which can take values between -1 and +1) is a useful measure that combines the ideas of cohesion and separation.

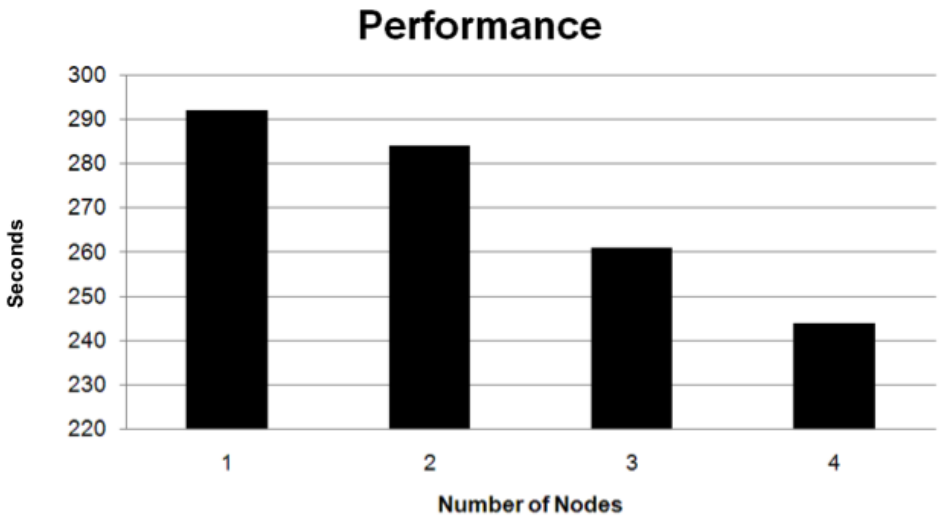

Figure 5.5: Chart shows the improvements.

In general, a good classification will have higher separation and lower cohesion which corresponds to silhouette coefficient being close to 1. It is clearly understandable that Affinity Propagation performs the best out of all the unsupervised machine learning techniques because the silhouette coefficient is closest to 1 in its case. The other two unsupervised machine learning techniques performed well and are very similar in performance when are compared to each other. The Mini Batch K-Means used less time and memory than its counterpart K-Means but still achieved the same accuracy. This is important in a practical context where resources such as CPU time and memory are limited. This can become a serious issue when there are large amounts of data involved. The issue with the unsupervised machine learning techniques that the results can vary and is not something that can be relied upon yet when compared to the supervised machine learning techniques as discussed in the previous Chapter.

5.10 Combined use of supervised and unsupervised machine learning techniques

The last Chapter showed the use of supervised techniques and it was decided to use the two techniques together to determine the malicious webpages. The

Table 5.3: Results of comparisons of unsupervised machine learning techniques that detect malicious webpages

Classifier	Silhouette Coefficient
Mini Batch K-Means	0.877
Affinity Propagation	0.963
K-Means	0.877

images of the webpages were clustered using unsupervised techniques and then fed into the models. Surprisingly, the efficiency did improve by a few percent.

5.11 Summary

The unsupervised techniques have shown that they are not far away from the supervised techniques and are good enough at classifying webpages. Also, the combination of both the supervised and the unsupervised techniques improved the efficiency of the models. This Chapter has also showed that the performance of machine learning techniques can be improved using multiple machines.

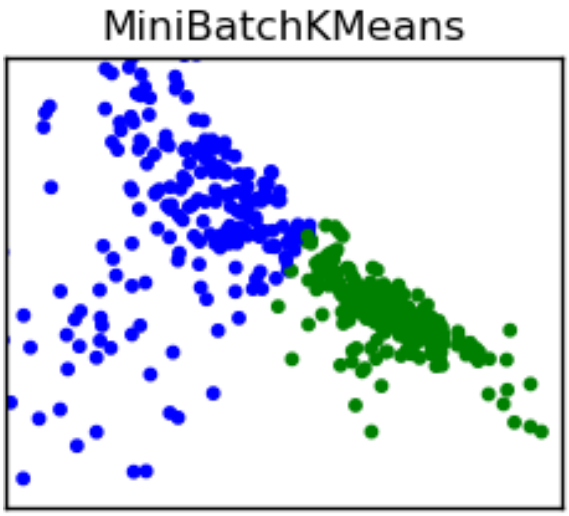

Figure 5.6: Visual representation of Mini Batch K-Means model shows clear boundaries between malicious and safe webpages

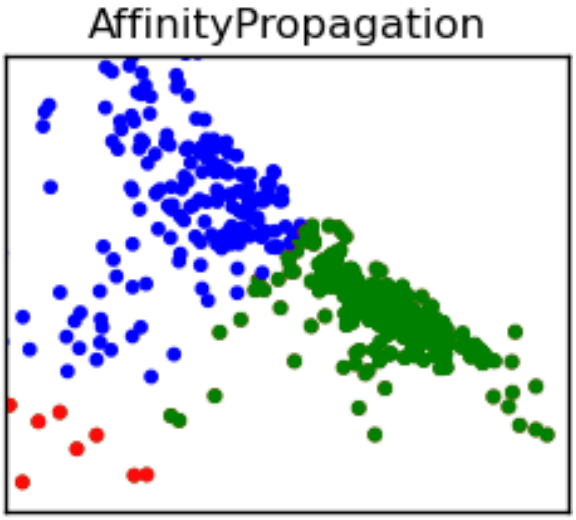

Figure 5.7: Visual representation of Affinity Propagation model shows clear boundaries between malicious and safe webpages. The red dots are outliers.

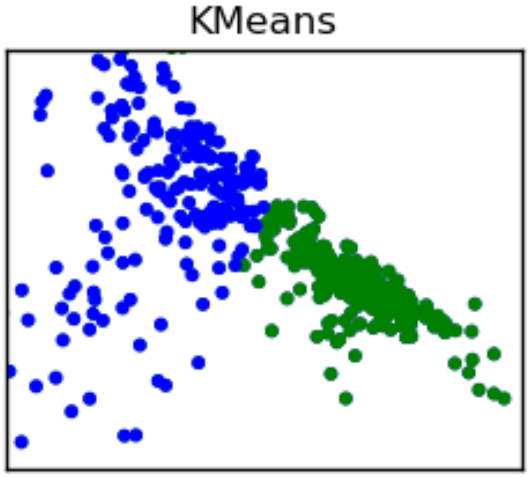

Figure 5.8: Visual representation of K-Means model shows clear boundaries between malicious and safe webpages

CHAPTER 6

Discussion and future work

The past two decades have radically changed the modern life, especially due to the fact that webpages have become an essential part of daily life. Webpages have become a vital and convenient way for people to conduct business, gather new information, discover new worlds and perform many other functions via desktops, tablets and smartphones which all have one thing in common - browsers to view webpages. The convenience of the Internet has a dramatic impact in improving the lives yet also left the users exposed to malicious webpages. The Internet does not have the quintessential properties found in the physical world to assess any webpages and lacks rules that are present in organisations such as banks and governments. Malicious attackers have benefitted from this and now a click to a malicious webpage's URL is enough to lead users to harm. The research community has dealt these threats by developing blacklisting services that compile a list of 'malicious' websites and also built client-side systems that analyse webpages as they are visited. These services are helpful, but are only useful for detecting known threats and fails to keep up with boundless stream of new malicious websites. Client-side systems are slow and may accidentally expose their users to threats. Not only are they available on browsers but webpages are also used largely inside mobile applications either in IOS, Android or Windows Mobile operating systems. As a result, the applications are now vulnerable to

these diversified attacks. To address the limitations of existing approaches, this thesis has developed an approach that is realtime, accurate, scalable and fast.

6.1 Contribution to knowledge

The central finding of this thesis is that machine learning techniques can alleviate the disadvantages of blacklists and heavyweight & lightweight classifiers. This has been achieved by constructing a realtime, accurate, middleweight and fast classification system. Before, malicious webpage detection generally used supervised and batch machine learning techniques. However, this thesis demonstrated that the approach of using a larger feature set especially the visual features had clear advantages over fewer feature sets. Moreover, the thesis demonstrated the benefits of online learning over standard batch learning for malicious URL detection (see Chapter 4). Online learning overcame the limitations of batch learning because of its ability to process large-scale data sets with fewer computational resources and to incorporate fresh training data incrementally. The thesis constructed a system for real-time feature collection, classification and showed the supervised and unsupervised classification techniques can work together to build better classifiers. In a future where there will be more data, the results from the

thesis demonstrated that scalable and adaptive detection is feasible, representing a significant advance over previous work.

One of the major contributions of this thesis was to explore a range of machine learning technique that used a wide range of features and the use of unsupervised machine learning techniques. The simulations show that a classifier can mine information contained in the URL, links, content and visual features from webpages. Beyond URL features, the most significant drops in error rates were obtained from the SURF visual features. The SURF visual features are more expensive to compute, but the SURF features provide a way to include visual information without a huge computational cost. Simulations on an up-to-date URL feed illustrate that the proposed approach can identify malicious webpages at about 98% accuracy and silhouette coefficient of up to 0.97. The features considered from webpage content more than halve the error rate that is obtained from URL features alone. The level of performance worsens but does not significantly deteriorate on unbalanced data sets with different ratios of malicious and safe webpages. It is interesting to see that the unsupervised machine learning techniques performed very well in their performance despite the fact that they do not have access to the class information about the data. This is an encouraging sign that perhaps in the future, they will overtake supervised machine learning techniques through

Deep Learning methods. This will have huge repercussions because there will be very less time involved in the training phase and data labelling. The classifiers will get close to decision making similar to human brains.

One of the major success of this thesis is improving the performance of the classifiers through the use of multicore and multiple machines. The Chapter on unsupervised algorithms demonstrates that the multi machine machine approach do indeed improve the run time of the classifiers. MapReduce proved to be a very worthy option when dealing with multi machine approach.

The final significant contribution of this thesis is the use of Chrome extension in the browser. This allows to detect whether a webpage is malicious or not very quickly often under a second. The lightweight classifiers are very fast but are less accurate whereas heavyweight classifiers are more accurate but take more time. This extension enables the classifier to be 'middleweight' with very high accuracy and yet less prediction time. This has the advantages of both the lightweight and the heavyweight classifiers. Chrome which is now becoming one of the most popular browsers in the market, has a large reach to the users in the world and this extension will certainly enable users feel more secure about browsing webpages. The indicator next to the address bar in the Chrome browser indicates to the user whether the webpage is malicious or not. The user can immediately decide

to act on the information.

The thesis has dealt with various types of malicious activities and it did not categorise the malicious activities. This caters for new malicious activities i.e. when they appear the models will categorise them.

6.2 Limitations

There are few limitations in this thesis, one of which is that malicious webpages are affected outside of features that have used in this thesis. In some cases, they can get affected with a combination of features. This thesis uses classification accuracy as the sole indicator. Despite these limitations, the thesis's contribution is encouraging: there are improved accuracy through additional features i.e. visual features. A further improvement would be to improve the execution time for classifying the webpages using GPU [84] and MapReduce [98]. Latest devices have access to GPUs, and this will improve the classification time. An extensive look at Deep Learning is needed in the future that will essentially detect webpages as malicious or not.

6.3 Future Work

This thesis opens up many areas for further research. To succeed, the approach of the thesis to malicious webpage relies heavily on features and large sources of training data. There are a number of ways to expand the work of this thesis, few of which are mentioned in the sections below.

More broadly, research is needed to determine other features of webpages automatically. Deep Learning [100] would be one possible solution. Realistically, Deep Learning is only part of the larger challenge of building intelligent machines. But Deep learning cannot represent causal relationships or perform logical inferences or integrate abstract knowledge. It can play a part in Bayesian inference or deductive reasoning. At present, the approach to detecting malicious webpages relies on a source of labeled examples. Instead, is it possible to use Deep Learning to build features and build a classifier to be used by already built Chrome extension. However, trusting the Deep Learning technique presents interesting challenges. Addressing these challenges would empower blacklisting without building the features which takes huge amount of time. The findings of this thesis have a number of important implications for future practice.

One improvement would be to move the classifiers to the browser and write

them in JavaScript, because the and GPUs are becoming powerful day by day. Browsers on the phones are now becoming more popular [101] and in the near future it would be possible to build the extension for mobile users as well. Another area to look at would be to construct the machine learning models within the Chrome browser using JavaScript. This is not possible at present because JavaScript is interpreted and the performance of the machine learning models will be painfully slow. But Chrome has a compiled version of JavaScript which runs on top of V8 [102]. This will make it a possibility in the future because the performance of the compiled JavaScript will be significantly faster than the interpreted versions. This would mean that the current implementation in thesis of a separate classifiers and the browser will all integrate together. This will allow the browser to be a standalone executable. The users will have access to an intelligent browser that will detect a malicious webpage on its own.

Looking ahead, it can be said that the combination of systems and machine learning will play a very important role in future research challenges. With more access to big data in the coming days it will be difficult to analyse it manually. Machine learning provides a methodical way to understand and build models from big data. Also, domain expertise and an understanding of limitations are important in developing a machine learning approach to solve problems. There

are two ways these solutions can work to detect malicious webpages. The first being the centralised way where all the classifiers can stay at a centralised location. The other option is that the classifiers will be stored in local computers and mobile phones. This thesis, with its focus on detecting malicious webpages, can cater both approaches and is thus a successful application of this insight.

CHAPTER 7

Conclusion

This thesis has looked at identifying malicious webpages in real time using various second and third generation machine learning techniques, and has been successful in providing a robust framework with a real working tool in achieving its goal. The thesis set out to determine the effectiveness of several supervised and unsupervised machine learning techniques that used various types of features such as content, URL, links and screenshots from more than 100,000 webpages. The supervised machine learning techniques did well compared to the unsupervised machine learning techniques. But the unsupervised techniques were not very far off and came very close. The results from the Chrome extension that took advantage of the best classifier of the supervised machine learning techniques, demonstrated that it has the advantages of both the heavyweight and the lightweight classifiers. Moreover, it avoids the disadvantages of the heavyweight and the lightweight classifiers. One of the most significant outcome from this thesis is that the visual features played a very important role in single handedly identifying a webpage to be either malicious or not. This is very significant and paves the way for Deep Learning techniques to detect a webpage to be malicious just like a human being.

The thesis also looked at improving the run time efficiency of the machine learning techniques. It used the MapReduce programming model to improve the

performance of the Self-Organising Map by distributing its execution over several machines. This reduced its execution time and opened up the possibility to use GPUs in the future.

APPENDIX A

Publications

1. Comparing machine learning techniques to detect malicious webpages (Expert Systems with Applications, USA, 2014)

2. Clustering Web Pages using MapReduced SOM (Journal of Communication and Computer, USA, May 2010)

3. Effective Web Technologies for a Web Design Agency (KTP Conference, University of Brighton, 2010)

4. Using Machine Learning to Optimise the Performance and Security of Web Applications (Third International Conference on Internet Technologies and Applications, Glyndwr University, North Wales, 2009)

APPENDIX B

Notation and abbreviations

The tables below have a summary of the used notation and abbreviations.

Symbol	Meaning
θ	Parameter vector.
$\boldsymbol{\theta}$	Random parameter vector.
M	Matrix.
$[N \times n]$	Dimensionality of a matrix with N rows and n columns.
M^T	Transpose of the matrix M.
$\text{diag}[m_1, \ldots, m_N]$	Diagonal matrix with diagonal $[m_1, \ldots, m_N]$
$\mathbf{M}$	Random matrix.
$\hat{\theta}$	Estimate of θ.
$\hat{\boldsymbol{\theta}}$	Estimator of θ.
τ	Index in an iterative algorithm.

Table B.1: General Notation.

Symbol	Meaning
Ω	Set of possible outcomes.
ω	Outcome or elementary event.
$\mathcal{E}$	Set of possible events.
$\mathcal{E}$	Event.
Prob$\{\mathcal{E}\}$	Probability of the event $\mathcal{E}$.
$(\Omega, \mathcal{E}, Prob\{\cdot\})$	Probabilistic model of an simulation.
$\mathcal{Z}$	Domain of the random variable $\mathbf{z}$.
$P(z)$	Probability distribution of a discrete random variable $\mathbf{z}$. Also $P_{\mathbf{z}}(z)$.
$F(z) = \text{Prob}\mathbf{z} \leq z$	Distribution function of a continuous random variable $\mathbf{z}$. Also $F_{\mathbf{z}}(z)$.
$p(z)$	Probability density of a continuous r.v.. Also $p_{\mathbf{z}}(z)$.
$E[\mathbf{z}]$	Expected value of the random variable $\mathbf{z}$.
$E_{\mathbf{x}}[\mathbf{z}] = \int_{\mathcal{X}} z(x,y)p(x)dx$	Expected value of the random variable $\mathbf{z}$ averaged over $\mathbf{x}$.
Var$[\mathbf{z}]$	Variance of the random variable $\mathbf{z}$.
$L(\theta)$	Likelihood of a parameter θ.
$l(\theta)$	Log-Likelihood of a parameter θ.
$l_{emp}(\theta)$	Empirical Log-likelihood of a parameter θ.

Table B.2: Probability Theory notation.

Symbol	Meaning
$\mathbf{x}$	Multidimensional input variable.
$\mathcal{X} \subset \Re^n$	Input space.
$\mathbf{y}$	Multidimensional output variable.
$\mathcal{Y} \subset \Re$	Output space.
x_i	i^{th} realization of the random vector $\mathbf{x}$.
$f(x)$	Target regression function.
$\mathbf{w}$	Random noise variable.
$z_i = \langle x_i, y_i \rangle$	Input-output sample: i^{th} case in training set.
N	Number of available samples.
$D_N = z_1, z_2, \ldots, z_N$	Training set.
Λ	Class of hypothesis.
α	Hypothesis parameter vector.
$h(x, \alpha)$	Hypothesis function.
Λ_s	Hypothesis class of complexity s.
$L(y, f(x, \alpha))$	Loss function.
$R(\alpha)$	Functional risk.
α_0	$\arg\min_{\alpha \in \Lambda} R(\alpha)$.
$R_{\text{emp}}(\alpha)$	Empirical functional risk.
α_N	$\arg\min R_{emp}(\alpha)$.
G_N	Mean integrated squared error (MISE).
l	Number of folds in cross-validation.
$\hat{G}_{\text{cv}}$	Cross-validation estimate of G_N.
$\hat{G}_{\text{loo}}$	Leave-one-out estimate of G_N.
N_{tr}	Number of samples used for training in cross-validation.
N_{ts}	Number of samples used for test in cross-validation.
$\alpha^i_{N_{tr}} \quad i = 1, \ldots, l$	Parameter which minimizes the empirical risk of $D_{N_{tr}}$.
$D_{(i)}$	Training set with the i^{th} sample set aside.
$\alpha_{N(i)}$	Parameter which minimizes the empirical risk of $D_{(i)}$.
$\hat{G}_{\text{boot}}$	Bootstrap estimate of G_N.
D^*_N	Bootstrap training set of size N generated by D_N with replacement.
N_b	Number of bootstrap samples.

Table B.3: Learning Theory notation.

Symbol	Meaning
x_{ij}	j^{th} element of vector x_i.
X_{ij}	ij^{th} element of matrix X.
q	Query point (point in the input space where a prediction is required).
$\hat{y}_q$	Prediction in the query point.
$\hat{y}_i^{-j}$	Leave-one-out prediction in x_i with the j^{th} sample set aside.
$e_j^{\text{loo}} = y_j - \hat{y}_j^{-j}$	Leave-one-out error with the j^{th} sample set aside.
$K(\cdot)$	Kernel function.
B	Bandwidth.
β	Linear coefficients vector.
$\hat{\beta}$	Least-squares parameters vector.
$\hat{\beta}^{-j}$	Least-squares parameters vector with the j^{th} sample set aside.
$h_j(x, \alpha)$	j^{th}, $j = 1, \ldots, m$, local model in a modular architecture.
ρ_j	Activation or basis function.
η_j	S et of parameters of the activation function.

Table B.4: Data analysis notation.

Abbreviation	Meaning
DT	Decision Tree
KNN	K-Nearest Neighbour
LDA	Latent Discriminant Analysis
MAP	Maximum a posteriori
MLE	Maximum likelihood estimate
NB	Naïve bayes
QDA	Quantitative Discriminant Analysis
RF	Random Forest
ROC	Receiving operating characteristic
SOM	Self-Organising Map
SVM	Support Vector Machine

Table B.5: List of abbreviations.

References

[1] Richard and Braganza, "Cross-site scripting an alternative view," *Network Security*, vol. 2006, no. 9, pp. 17 – 20, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1353485806704251 12, 29, 30

[2] F. Callegati and M. Ramilli, "Frightened by links," *Security Privacy, IEEE*, vol. 7, no. 6, pp. 72 –76, nov.-dec. 2009. 12, 32, 35

[3] P. Flach, *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. New York, NY, USA: Cambridge University Press, 2012. 13, 14, 49, 53

[4] M. Kuhn and K. Johnson, *Applied predictive modeling.* New York, NY: Springer, 2013. 13, 50

[5] K. P. Murphy, *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012. 14, 15, 52, 54, 58

[6] S. Marsland, *Machine Learning: An Algorithmic Perspective*, 1st ed. Chapman & Hall/CRC, 2009. 15, 60, 61

[7] G. C. Extension. (2010, Jan.) Google chrome extension. [Online]. Available: https://developer.chrome.com/extensions 18, 101, 102

[8] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 1245–1254. [Online]. Available: http://doi.acm.org/10.1145/1557019.1557153 20, 63, 86, 103

[9] Google. (2010, Jan.) Google. [Online]. Available: http://www.google.com 23, 115, 125

[10] A. K. Sood and R. J. Enbody, "Frametrapping the framebusting defence,"

Network Security, vol. 2011, no. 10, pp. 8 – 12, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1353485811701052 24

[11] S. Lekies, B. Stock, and M. Johns, "25 million flows later: Large-scale detection of dom-based xss," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 1193–1204. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516703 26, 27

[12] G. Di Lucca, A. Fasolino, M. Mastoianni, and P. Tramontana, "Identifying cross site scripting vulnerabilities in web applications," in *Web Site Evolution, 2004. WSE 2004. Proceedings. Sixth IEEE International Workshop on*, sept. 2004, pp. 71 – 80. 26, 34

[13] L. K. Shar and H. B. K. Tan, "Automated removal of cross site scripting vulnerabilities in web applications," *Information and Software Technology*, vol. 54, no. 5, pp. 467 – 478, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584911002503 28

[14] E. Kirda, N. Jovanovic, C. Kruegel, and G. Vigna, "Client-side cross-site scripting protection," *Computers and Security*, vol. 28, no. 7, pp. 592

– 604, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404809000479 30

[15] H. Shahriar, V. K. Devendran, and H. Haddad, "Proclick: A framework for testing clickjacking attacks in web applications," in *Proceedings of the 6th International Conference on Security of Information and Networks*, ser. SIN '13. New York, NY, USA: ACM, 2013, pp. 144–151. [Online]. Available: http://doi.acm.org/10.1145/2523514.2523538 31

[16] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda, "Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks," in *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 88–106. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02918-9_6 35

[17] J. Narvaez, B. Endicott-Popovsky, C. Seifert, C. Aval, and D. A. Frincke, "Drive-by-downloads," in *Proceedings of the 2010 43rd Hawaii International Conference on System Sciences*, ser. HICSS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/HICSS.2010.160 35, 36

[18] D. Harley and P.-M. Bureau, "Drive-by downloads from the trenches," in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, oct. 2008, pp. 98 –103. 35

[19] V. Okanovic and T. Mateljan, "Designing a new web application framework," in *MIPRO, 2011 Proceedings of the 34th International Convention*, may 2011, pp. 1315 –1318. 36, 41

[20] R. Li, M. Dong, B. Liu, J. Lu, X. Ma, and K. Li, "Sectag: A multi-policy supported secure web tag framework," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 633–635. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866379 36

[21] D. Tarditi, S. Puri, and J. Oglesby, "Accelerator: Using data parallelism to program gpus for general-purpose uses," Microsoft Research, Tech. Rep. MSR-TR-2005-184, October 2006. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=70250 41

[22] H. Martín, A. M. Bernardos, J. Iglesias, and J. R. Casar, "Activity logging using lightweight classification techniques in mobile devices," *Personal*

Ubiquitous Comput., vol. 17, no. 4, pp. 675–695, Apr. 2013. [Online]. Available: http://dx.doi.org/10.1007/s00779-012-0515-4 42

[23] J.-S. Park, S.-H. Yoon, and M.-S. Kim, "Software architecture for a lightweight payload signature-based traffic classification system," in *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*, ser. TMA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 136–149. [Online]. Available: http://dl.acm.org/citation.cfm?id=1986282.1986298 42

[24] H. Shahriar and M. Zulkernine, "Client-side detection of cross-site request forgery attacks," in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ser. ISSRE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 358–367. [Online]. Available: http://dx.doi.org/10.1109/ISSRE.2010.12 42

[25] Y. Cao, Z. Li, V. Rastogi, Y. Chen, and X. Wen, "Virtual browser: A virtualized browser to sandbox third-party javascripts with enhanced security," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS

'12. New York, NY, USA: ACM, 2012, pp. 8–9. [Online]. Available: http://doi.acm.org/10.1145/2414456.2414460 42

[26] M. Kan and H. Thi, "Fast webpage classification using url features," in *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 2005, pp. 325–326. 43

[27] S. Garera, N. Provos, M. Chew, and A. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the 2007 ACM workshop on Recurring malcode*. ACM, 2007, pp. 1–8. 43

[28] E. Spertus, "Smokey: Automatic recognition of hostile messages," in *Proceedings of the National Conference on Artificial Intelligence*. Citeseer, 1997, pp. 1058–1065. 43, 68

[29] W. Cohen, "Learning Trees an ules with Set-val ed Features." 43, 68, 71

[30] S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," in *Proceedings of the seventh international conference on Information and knowledge management*. ACM, 1998, pp. 148–155. 43, 69

[31] D. Guan, C. Chen, and J. Lin, "Anomaly based malicious url detection in

instant messaging," in *Proceedings of the Joint Workshop on Information Security (JWIS)*, 2009. 43

[32] D. K. McGrath and M. Gupta, "Behind phishing: An examination of phisher modi operandi," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, ser. LEET'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 4:1–4:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387709.1387713 44

[33] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, "All your iframes point to us," in *Proceedings of the 17th conference on Security symposium*. USENIX Association, 2008, pp. 1–15. 44, 63

[34] S. N. Bannur, L. K. Saul, and S. Savage, "Judging a site by its content: learning the textual, structural, and visual features of malicious web pages," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, ser. AISec '11. New York, NY, USA: ACM, 2011, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/2046684.2046686 45

[35] E. Charniak, *Statistical language learning*. The MIT Press, 1996. 46

[36] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine Learning*, 1995, pp. 273–297. 47

[37] C.-C. Chang and C.-J. Lin. (2012, Mar.) Libsvm: A library for sup- port vector machines. [Online]. Available: http://www.csie.ntu.edu.tw/cjlin/libsvm/ 49

[38] T. A. Almeida, A. Yamakami, and J. Almeida, "Evaluation of approaches for dimensionality reduction applied with naive bayes anti-spam filters," in *Proceedings of the 2009 International Conference on Machine Learning and Applications*, ser. ICMLA '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 517–522. [Online]. Available: http://dx.doi.org/10.1109/ICMLA.2009.22 49

[39] C. Bishop and S. S. en ligne), *Pattern recognition and machine learning*. springer New York, 2006, vol. 4. 49, 51

[40] J. R. Quinlan, "Learning decision tree classifiers," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 71–72, Mar. 1996. [Online]. Available: http://doi.acm.org/10.1145/234313.234346 51

[41] S. B. Kotsiantis, "Decision trees: A recent overview," *Artif. Intell.*

Rev., vol. 39, no. 4, pp. 261–283, Apr. 2013. [Online]. Available: http://dx.doi.org/10.1007/s10462-011-9272-4 51

[42] L. B. Statistics and L. Breiman, "Random forests," in *Machine Learning*, 2001, pp. 5–32. 52

[43] T. K. Ho, "Random decision forests," in *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1, Aug 1995, pp. 278–282 vol.1. 53

[44] T. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *Electronic Computers, IEEE Transactions on*, vol. EC-14, no. 3, pp. 326–334, 1965. 53

[45] R. A. FISHER, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936. [Online]. Available: http://dx.doi.org/10.1111/j.1469-1809.1936.tb02137.x 54

[46] H. Abdi and D. Valentin, "Discriminant correspondence analysis," in *In N.J. Salkind (Ed): Encyclopedia of measurement and statistics. Thousand Oaks: Sage. (pp.270275.* 54

[47] G. Perriere and J. Thioulouse, "Use of correspondence discriminant analysis to predict the subcellular location of bacterial proteins," 2003. 54

[48] N. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992. 55

[49] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, pp. 972–976, 2007. [Online]. Available: www.psi.toronto.edu/affinitypropagation 56

[50] T. Kohonen, "Essentials of the self-organizing map," *Neural Netw.*, vol. 37, pp. 52–65, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2012.09.018 59, 112

[51] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, Eds., vol. 1. University of California Press, 1967, pp. 281–297. 60

[52] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10.

New York, NY, USA: ACM, 2010, pp. 1177–1178. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772862 61

[53] H. Yang, M. R. Lyu, and I. King, "Efficient online learning for multitask feature selection," *ACM Trans. Knowl. Discov. Data*, vol. 7, no. 2, pp. 6:1–6:27, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2499907.2499909 62

[54] G. Salton, A. Wong, and C. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975. 66

[55] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval* 1," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988. 66

[56] T. Mitchell, "Machine learning." *Burr Ridge, IL: McGraw Hill*, 1977. 67

[57] M. Porter, "An algorithm for suffix stripping," *Program: electronic library and information systems*, vol. 14, no. 3, pp. 130–137, 1993. 67

[58] W. Frakes and R. Baeza-Yates, "Information retrieval: data structures and algorithms," 1992. 68

[59] W. Cohen and Y. Singer, "Context-sensitive learning methods for text categorization," *ACM Transactions on Information Systems (TOIS)*, vol. 17, no. 2, pp. 141–173, 1999. 68

[60] J. Furnkranz, T. Mitchell, and E. Riloff, "A case study in using linguistic phrases for text categorization on the www," in *Proceedings from the AAAI/ICML Workshop on Learning for Text Categorization*, 1998, pp. 5–12. 68

[61] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A bayesian approach to filtering junk e-mail," in *Learning for Text Categorization: Papers from the 1998 workshop*, vol. 62, 1998. 68

[62] J. Bao, J. Shen, X. Liu, and H. Liu, "The heavy frequency vector-based text clustering," *Int. J. Bus. Intell. Data Min.*, vol. 1, no. 1, pp. 42–53, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1504/IJBIDM.2005.007317 69

[63] S. Robertson and K. Jones, "Relevance weighting of search terms," *Journal of the American society for Information Science*, vol. 27, no. 3, pp. 129–146, 1976. 69

[64] D. Cutting, D. Karger, J. Pedersen, and J. Tukey, "Scatter/gather: A cluster-based approach to browsing large document collections," in *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1992, pp. 318–329. 69

[65] S. Robertson and S. Walker, "Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval," in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc., 1994, pp. 232–241. 69

[66] T. Joachims and C.-M. U. P. P. D. O. C. SCIENCE., "A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization," in *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*. Citeseer, 1997, pp. 143–151. 70

[67] J. Zeng, C. Wu, and W. Wang, "Multi-grain hierarchical topic extraction algorithm for text mining," *Expert Syst. Appl.*, vol. 37, no. 4, pp. 3202–3208, Apr. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2009.09.061 71

[68] Y. Yang and C. Chute, "An example-based mapping method for text catego-

rization and retrieval," *ACM Transactions on Information Systems (TOIS)*, vol. 12, no. 3, pp. 252–277, 1994. 71

[69] C., F. Damerau, and S. Weiss, "Automated learning of decision rules for text categorization," *ACM Transactions on Information Systems (TOIS)*, vol. 12, no. 3, pp. 233–251, 1994. 71

[70] R. Jurgelenaite and T. Heskes, "Learning symmetric causal independence models," *Mach. Learn.*, vol. 71, no. 2-3, pp. 133–153, Jun. 2008. [Online]. Available: http://dx.doi.org/10.1007/s10994-007-5041-7 71

[71] L. R. Biggers, "The effects of identifier retention and stop word removal on a latent dirichlet allocation based feature location technique," in *Proceedings of the 50th Annual Southeast Regional Conference*, ser. ACM-SE '12. New York, NY, USA: ACM, 2012, pp. 164–169. [Online]. Available: http://doi.acm.org/10.1145/2184512.2184551 72

[72] C. Van Rijsbergen, "Information retrieval. 1979," 2004. 72

[73] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004. 73

[74] Y. Ke and R. Sukthankar, "Pca-sift: A more distinctive representation for

local image descriptors," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, ser. CVPR'04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 506–513. [Online]. Available: http://dl.acm.org/citation.cfm?id=1896300.1896374 73

[75] K. Elmeleegy, A. Chanda, A. L. Cox, and W. Zwaenepoel, "Lazy asynchronous i/o for event-driven servers," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247415.1247436 77

[76] PhishTank. (2012, Mar.) Phishtank. [Online]. Available: http://www.phishtank.com/ 79

[77] A. K. Uysal and S. Gunal, "The impact of preprocessing on text classification," *Inf. Process. Manage.*, vol. 50, no. 1, pp. 104–112, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2013.08.006 79

[78] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2005.10.010 80

[79] A. Blum, "On-line algorithms in machine learning," in *Developments from a June 1996 Seminar on Online Algorithms: The State of the Art*. London, UK, UK: Springer-Verlag, 1998, pp. 306–325. [Online]. Available: http://dl.acm.org/citation.cfm?id=647371.723908 91

[80] N. Carlini, A. P. Felt, and D. Wagner, "An evaluation of the google chrome extension security architecture," in *Proceedings of the 21st USENIX Conference on Security Symposium*, ser. Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2362793.2362800 100

[81] L. Liu, X. Zhang, G. Yan, and S. Chen, "Chrome extensions: Threat analysis and countermeasures." in *NDSS*, 2012. 100

[82] P. H. Chia, Y. Yamamoto, and N. Asokan, "Is this app safe?: A large scale study on application permissions and risk signals," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 311–320. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187879 100

[83] T. Kohonen, M. R. Schroeder, and T. S. Huang, Eds., *Self-Organizing Maps*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001. 110

[84] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008. 110, 116, 121, 125, 137

[85] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas, "Engineering applications of the self-organizing map," *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1358–1384, 1996. 110

[86] J. Moehrmann, A. Burkovski, E. Baranovskiy, G.-A. Heinze, A. Rapoport, and G. Heidemann, "A discussion on visual interactive data exploration using self-organizing maps," in *Proceedings of the 8th International Conference on Advances in Self-organizing Maps*, ser. WSOM'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 178–187. [Online]. Available: http://dl.acm.org/citation.cfm?id=2026666.2026688 110

[87] K. A. Smith and A. Ng, "Web page clustering using a self-organizing map of user navigation patterns," *Decis. Support Syst.*, vol. 35, no. 2, pp. 245–256, 2003. 110

[88] T. Kohonen, S. Kaski, K. Lagus, J. Salojarvi, V. Paatero, and A. Saarela, "Self organization of a massive document collection," *IEEE Transactions on Neural Networks*, vol. 11, pp. 574–585. 110, 123

[89] G. Arroyave, O. O. Lobo, and A. Marn, "A parallel implementation of the som algorithm for visualizing textual documents in a 2d plane," 2002. 111

[90] (2012, Mar.) Dmoz open directory project. [Online]. Available: http://www.dmoz.org 114

[91] Yahoo. (2010, Jan.) Yahoo. [Online]. Available: http://random.yahoo.com/bin/ryl 114

[92] OpenDNS. (2010, Jan.) Internet security or dns service for your business or home - opendns. [Online]. Available: http://www.opendns.com 114

[93] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker, "Spamscatter: Characterizing internet scam hosting infrastructure," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 10:1–10:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1362903.1362913 114, 119

[94] Apache. (2010, Jan.) Apache hadoop. [Online]. Available: http://hadoop.apache.org/ 119

[95] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg,

H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, "Apache hadoop goes realtime at facebook," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11. New York, NY, USA: ACM, 2011, pp. 1071–1080. [Online]. Available: http://doi.acm.org/10.1145/1989323.1989438 119

[96] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 147–156. [Online]. Available: http://doi.acm.org/10.1145/2588555.2595641 119

[97] *Beowulf Cluster Computing with Linux*. Cambridge, MA, USA: MIT Press, 2002. 119

[98] R. Lämmel, "Google's mapreduce programming model — revisited," *Sci. Comput. Program.*, vol. 68, no. 3, pp. 208–237, 2007. 121, 137

[99] P. Frenger, "Hard java," *SIGPLAN Not.*, vol. 43, no. 5, pp. 5–9, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1402227.1402231 125

[100] E.-o. Baek and H.-J. Lee, "Facilitating deep learning in a learning community," *Int. J. Technol. Hum. Interact.*, vol. 8, no. 1, pp. 1–13, Jan. 2012. [Online]. Available: http://dx.doi.org/10.4018/jthi.2012010101 138

[101] G. Schmiedl, M. Seidl, and K. Temper, "Mobile phone web browsing: A study on usage and usability of the mobile web," in *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '09. New York, NY, USA: ACM, 2009, pp. 70:1–70:2. [Online]. Available: http://doi.acm.org/10.1145/1613858.1613942 139

[102] W. Ahn, J. Choi, T. Shull, M. J. Garzarán, and J. Torrellas, "Improving javascript performance by deconstructing the type system," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '14. New York, NY, USA: ACM, 2014, pp. 496–507. [Online]. Available: http://doi.acm.org/10.1145/2594291.2594332 139