

PHISHTOR CHROME EXTENSION FOR PHISHING WEBSITE DETECTION

by

RAMKUMAR B 2015103042

ASWIN SUNDAR 2015103006

A project report submitted to the

**FACULTY OF INFORMATION AND
COMMUNICATION ENGINEERING**

in partial fulfilment of the requirements for the

award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**

ANNA UNIVERSITY, CHENNAI – 25

NOVEMBER 2018

BONAFIDE CERTIFICATE

Certified that this project report titled **PHISHTOR - CHROME EXTENSION FOR PHISHING WEBSITE DETECTION** is the bonafide work of **RAMKUMAR B (2015103042)** and **ASWIN SUNDAR (2015103006)** who carried out the project work for **Creative and Innovative Lab** under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place:
Chennai

Date:

Bhuvaneshwari R

Teaching Fellow

Department of Computer Science and
Engineering

Anna University, Chennai – 25

ABSTRACT

This document proposes a phishing¹ detection plugin for chrome browser that can detect and warn the user about phishing web sites in real-time using random forest classifier. Based on the IEEE paper, Intelligent phishing website detection using random forest classifier², the random forest classifier seems to outperform other techniques in detecting phishing websites.

One common approach is to make the classification in a server and then let the plugin to request the server for result. Unlike the old approach, this project aims to run the classification in the browser itself. The advantage of classifying in the client side browser has advantages like, better privacy (the user's browsing data need not leave his machine), detection is independent of network latency.

This project is mainly of implementing the above mentioned paper in Javascript for it to run as a browser plugin. Since javascript doesn't have much ML libraries support and considering the processing power of the client machines, the approach needs to be made lightweight. The random forest classifier needs to be trained on the phishing websites dataset³ using python scikit-learn and then the learned model parameters need to be exported into a portable format for using in javascript.

¹ Phishing is mimicking a creditable company's website to take private information of a user.

² <https://ieeexplore.ieee.org/abstract/document/8252051/>

³ <https://archive.ics.uci.edu/ml/datasets/phishing+websites>

TABLE OF CONTENTS

ABSTRACT	i
LIST OF FIGURES	iv
LIST OF TABLES	v
LIST OF ABBREVIATIONS	vi
INTRODUCTION	1
1.1 PROBLEM DOMAIN	1
1.2 PROBLEM DESCRIPTION	2
1.3 SCOPE	2
1.4 CONTRIBUTION	2
1.5 SWOT ANALYSIS	3
1.6 PESTLE ANALYSIS	4
1.7 ORGANISATION OF THESIS	5
RELATED WORKS	6
2.1 DIRECTORY BASED APPROACHES	7
2.2 RULE BASED APPROACHES	7
2.3 ML BASED APPROACHES	8
2.4 DRAWBACKS	9
REQUIREMENTS ANALYSIS	10
3.1 FUNCTIONAL REQUIREMENTS	10
3.2 NON FUNCTIONAL REQUIREMENTS	10
3.3 CONSTRAINTS AND ASSUMPTIONS	11
3.4 SYSTEM MODELS	12
SYSTEM DESIGN	14
4.1 SYSTEM ARCHITECTURE	14
4.2 UI DESIGN	16
4.3 CLASS DIAGRAM	16

4.4 MODULE DESIGN	18
4.5 COMPLEXITY ANALYSIS	21
SYSTEM DEVELOPMENT	23
5.1 PROTOTYPE ACROSS THE MODULES.....	24
5.2 EXPORTING ALGORITHM.....	24
5.3.DEPLOYMENT DETAILS.....	25
RESULTS AND DISCUSSION	26
6.1.DATASET FOR TESTING	26
6.2.OUTPUT OBTAINED IN VARIOUS STAGES	26
6.3.SAMPLE SCREENSHOTS DURING TESTING	29
6.4 PERFORMANCE EVALUATION	29
CONCLUSIONS.....	32
7.1 SUMMARY.....	32
7.2.CRITICISMS.....	33
7.3.FUTURE WORK	33
REFERENCES	34

LIST OF FIGURES

Figure 2.1 Approaches to phishing detection	6
Figure 3.1 Use case diagram of the system	12
Figure 3.2 System Sequence diagram	13
Figure 4.1 System Architecture	15
Figure 4.2 UI Design	17
Figure 4.3 Class Diagram	17
Figure 4.4 Random Forest JSON structure	19
Figure 5.1 Code Overview	23
Figure 6.1 Preprocessing output	26
Figure 6.2 Training output	27
Figure 6.3 Model JSON	27
Figure 6.4 Webpage features	28
Figure 6.5 Classification Output	28
Figure 6.6 Test Output	29
Figure 6.7 Cross Validation Output	30
Figure 6.8 Performance measure	31

LIST OF TABLES

Table 1.1 SWOT analysis	3
Table 4.1 Webpage Features	18
Table 4.2 Time Complexity of various modules	21

LIST OF ABBREVIATIONS

URL	Uniform Resource Locator
API	Application Programming Interface
HTTP	Hyper Text Transfer Protocol
SSL	Secured Socket Layer
DNS	Domain Name System
GDPR	General Data Privacy Regulation
JSON	JavaScript Object Notation
DOM	Document Object Model
UI	User Interface
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DOMAIN

Phishing is the fraudulent attempt to obtain sensitive information such as usernames, passwords, and credit card details (and money), often for malicious reason. It is typically carried out by email spoofing or instant messaging, and it often directs users to enter personal information at a fake website, the look and feel of which are identical to the legitimate site, the only difference being the URL of the website in concern. Communications purporting to be from social web sites, auction sites, banks, online payment processors are often used to lure victims. Phishing emails may contain links to websites that distribute malware.

Detecting phishing websites often include lookup in a directory of malicious sites. Since most of the phishing websites are short lived, the directory cannot always keep track of all, including new phishing websites. So the problem of detecting phishing websites can be solved in a better way by machine learning techniques. Based on a comparison of different ML techniques, the random forest classifier seems to perform better.

Only way for an end user to benefit from this is to implement detection in a browser plugin. So that the user can be warned in real time as he browses a phishing site. However, browser extensions have restrictions such as they can be written only in javascript and they have limited access to page URLs and resources.

Existing plugins send the URL to a server, so that the classification can be done in the server and the result is returned to the plugin. With this approach, user privacy is questioned and also the detection may be delayed due to network latency and the plugin may fail to warn the user in right time. As it is an important security problem and also considering the privacy aspects, we decided to implement this on a chrome browser plugin which can do the classification without an external server.

1.2 PROBLEM DESCRIPTION

To develop a browser plugin which once installed, should warn the user on the event of he/she visiting a phishing website. The plugin should not contact any external web service for this which may leak the user's browsing data. The detection should be instant so that the user will be warned before entering any sensitive information on the phishing website.

1.3 SCOPE

According to wikipedia, In 2017, 76% of organisations experienced phishing attacks. Nearly half of information security professionals surveyed said that the rate of attacks increased from 2016. In the first half of 2017 businesses and residents of Qatar were hit with more than 93,570 phishing events in a three-month span. With increasing number of internet users, there is a prominent need for security solutions against attacks such as phishing. Hence this plugin would be a good contribution for the chrome users.

1.4 CONTRIBUTION

This is the first implementation of phishing website detection in browser plugin without use of an external web service. This makes use

of existing works done on phishing detection and implements them in a manner that it will benefit end users. This involves porting the existing python classifier (random forest) to javascript. The plugin with an one time download of the learned model, will be able to classify websites in real time. This involves developing such a model (random forest) in javascript, as browser plugin supports only javascript. Thus this project contributes to better privacy and rapid detection of phishing.

1.5 SWOT ANALYSIS

STRENGTHS	WEAKNESSES
<ul style="list-style-type: none"> • Enables user privacy. • Rapid detection of phishing. • Can detect new phishing sites too. • Can interrupt the user incase of phishing. 	<ul style="list-style-type: none"> • Javascript limits functionality. • Cannot use features that needs a external service such as SSL, DNS, page ranks. • No library support.
OPPORTUNITIES	THREATS
<ul style="list-style-type: none"> • Everyone conscious of privacy and security can use this plugin. • Non technical people who do business transactions are vulnerable to phishing and they are potential end users for this. 	<ul style="list-style-type: none"> • Server side classification plugins may perform better than this and users without privacy concerns may opt of those. • Chrome Plugin API will be continuously changed.

Table 1.1 SWOT analysis

1.6 PESTLE ANALYSIS

1.6.1 Political

This project is rarely controlled by political factors. One such scenario is that the government may take any measure to prevent phishing and in such cases the plugin may lose its potential.

1.6.2 Economical

The plugin is completely based on public dataset and open chrome plugin API. Thus it has no Economical factors controlling it.

1.6.3 Social

The social factor that will have effect on this plugin is awareness of the user. Phishing detection systems aims to aid an user in finding a phishing sites. At least the users need to be aware about phishing to install this plugin.

1.6.4 Technological

Technological advancements or improved techniques for phishing detection can make this plugin become outdated and are a major threat to this plugin.

1.6.5 Legal

Legal policies those in concern of user privacy such as GDPR will enhance the potential of this plugin as user will be moving to more privacy based products.

1.6.6 Environmental

This plugin has no environmental factors affecting it.

1.7 ORGANISATION OF THESIS

Chapter 2 discusses the existing approaches to phishing detection in greater detail. Chapter 3 gives the requirements analysis of the system. It explains the functional and non-functional requirements, constraints and assumptions made in the implementation of the system and the various UML diagrams. Chapter 4 explains the overall system architecture and the design of various modules along with their complexity. Chapter 5 gives the implementation details of each module. Chapter 6 elaborates on the results of the implementation. Chapter 7 concludes the thesis and gives an overview of its criticisms.

CHAPTER 2

RELATED WORKS

This chapter gives a survey of the possible approaches to phishing website detection. This survey helps to identify various existing approaches and to find the drawbacks in them. The difficulty in most of the approaches is that they are not implemented in real time so that an end user will benefit from it.

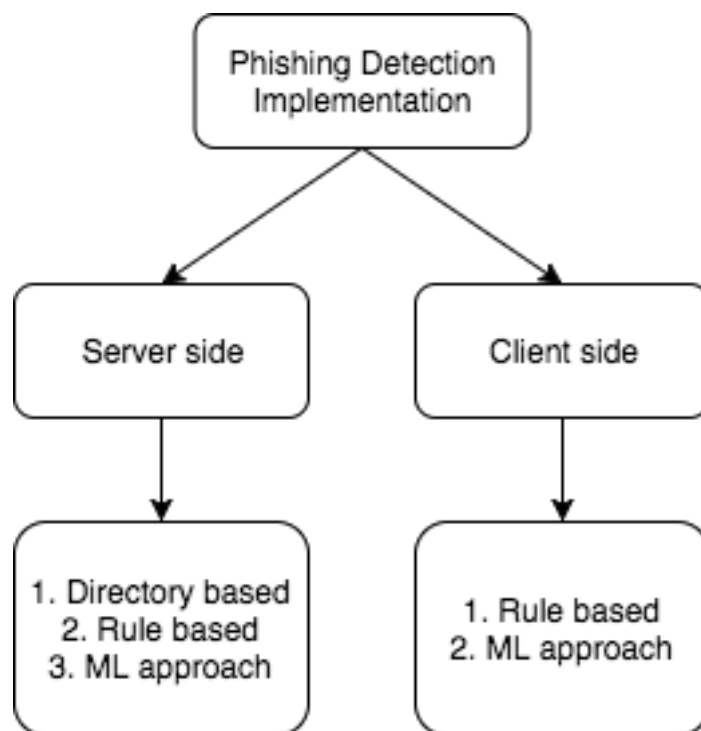


Figure 2.1 Approaches to phishing detection

2.1 DIRECTORY BASED APPROACHES

Most popular one of this kind is PhishTank. According to PhishTank⁴, it is a collaborative clearing house for data and information about phishing on the Internet. Also, PhishTank provides an open API for developers and researchers to integrate anti-phishing data into their applications at no charge. Thus PhishTank is a directory of all phishing websites that are found and reported by people across the web so that developers can use their API for detecting phishing websites.

Google has a API called Google Safe Browsing API which also follows directory based approach and also provides open API similar to PhishTank.

This kind of approach clearly can't be effective as new phishing web sites are continuously developed and the directory can't be kept up to date always. This also leaks users browsing behaviour as the URLs are sent to the PhishTank API.

2.2 RULE BASED APPROACHES

An existing chrome plugin named PhishDetector⁵ uses a rule based approach so that it can detect phishing without external web service. Although rule based approaches support easier implementation on client side, they can't be accurate compared to Machine Learning based approaches. Similar work by Shreeram.V on detection of phishing attacks using genetic algorithm⁶ uses a rule that is generated by a genetic algorithm for detection.

⁴ <http://phishtank.com/>

⁵ <https://chrome.google.com/webstore/detail/phishdetector-true-phishi/kgceldbalfgmgelepbblood-foogmjdgmj>

⁶ <https://ieeexplore.ieee.org/document/5670593/>

PhishNet is one such Predictive blacklisting approach. It used rules that can match with TLD, directory structure, IP address, HTTP header response and some other.

SpoofGuard⁷ by Stanford is a chrome plugin which used similar rule based approach by considering DNS, URL, images and links.

Phishwish: A Stateless Phishing Filter Using Minimal Rules by Debra L. Cook, Vijay K. Gurbani, Michael Daniluk worked on a phishing filter that offers advantages over existing filters: It does not need any training and does not consult centralized white or black lists. They used only 11 rules to determine the veracity of a website.

2.3 ML BASED APPROACHES

Intelligent phishing website detection using random forest classifier (IEEE-2017) by Abdulhamit Subasi, Esraa Molah, Fatin Almkallawi and Touseef J. Chaudhery discusses the use of the random forest classifier for phishing detection. Random Forest has performed the best among the classification methods by achieving the highest accuracy 97.36%.

PhishBox: An Approach for Phishing Validation and Detection (IEEE-2017) by Jhen-Hao Li, and Sheng-De Wang discusses ensemble models for phishing detection. As a result, The false-positive rate of phishing detection is dropped by 43.7% in average.

Real time detection of phishing websites (IEEE-2016) by Abdulghani Ali Ahmed, and Nurul Amirah Abdullah discusses an approach based on features from only the URL of the website. They were able to come up with a detection mechanism that is capable of detecting various types of phishing attacks maintaining a low rate of false alarms.

Using Domain Top-page Similarity Feature in Machine Learning-Based Web Phishing Detection by Nuttapon Sanglerdsinlapachai,

⁷ <https://crypto.stanford.edu/SpoofGuard/>

Arnon Rungsawang presents a study on using a concept feature to detect web phishing problem. They applied additional domain top-page similarity feature to a machine learning based phishing detection system. The evaluation result in terms of f-measure was up to 0.9250, with 7.50% of error rate.

Netcraft⁸ is one popular phishing detection plugin for chrome that uses server side prediction.

2.4 DRAWBACKS

Based on the above mentioned related works, It can be seen that the plugins either use rule based approach or server side ML based approach. Rule based approach doesn't seem to perform well compared to ML based approaches and on the other side ML based approaches need libraries support and so they are not implemented in client side plugin. All the existing plugins send the target URL to an external web server for classification. This project aims to implement the same in browser plugin removing the need of external web service and improving user privacy.

⁸ <https://toolbar.netcraft.com>

CHAPTER 3

REQUIREMENTS ANALYSIS

3.1 FUNCTIONAL REQUIREMENTS

The plugin warns the user when he/she visits a phishing website. The plugin should adhere to the following requirements:

- The plugin should be fast enough to prevent the user from submitting any sensitive information to the phishing website.
- The plugin should not use any external web service or API which can leak user's browsing pattern.
- The plugin should be able to detect newly created phishing websites.
- The plugin should have a mechanism of updating itself to emerging phishing techniques.

3.2 NON FUNCTIONAL REQUIREMENTS

3.2.1 User Interface

There must be a simple and easy to use user interface where the user should be able to quickly identify the phishing website. The input should be automatically taken from the webpage in the current tab and the output should be clearly identifiable. Further the user should be interrupted on the event of phishing.

3.2.2 Hardware

No special hardware interface is required for the successful implementation of the system.

3.2.3 Software

- Python for training the model
- Chrome browser

3.2.4 Performance

The plugin should be always available and should make fast detection with low false negatives.

3.3 CONSTRAINTS AND ASSUMPTIONS

3.3.1 Constraints

- Certain techniques use features such as SSL, page rank etc. Such information cannot be obtained from client side plugin without external API. Thus those features can't be used for prediction.
- Heavy techniques can't be used considering the processing power of client machines and the page load time of the website.
- Only Javascript can be used to develop chrome plugins. Machine learning libraries support for javascript is far less compared to python and R.

3.3.2 Assumptions

- The plugin is provided with the needed permissions in the chrome environment.
- The user has a basic knowledge about phishing and extensions.

3.4 SYSTEM MODELS

3.4.1 Use Case Diagram

The overall use case diagram of the entire system is shown in figure 3.1. The user can install the plugin and then can continue his normal browsing behaviour. This plugin will automatically check the browsing pages for phishing and warns the user of the same.

Pre condition: The user visits a website and have plugin installed.

Post condition: The user is warned incase it's a phishing website.

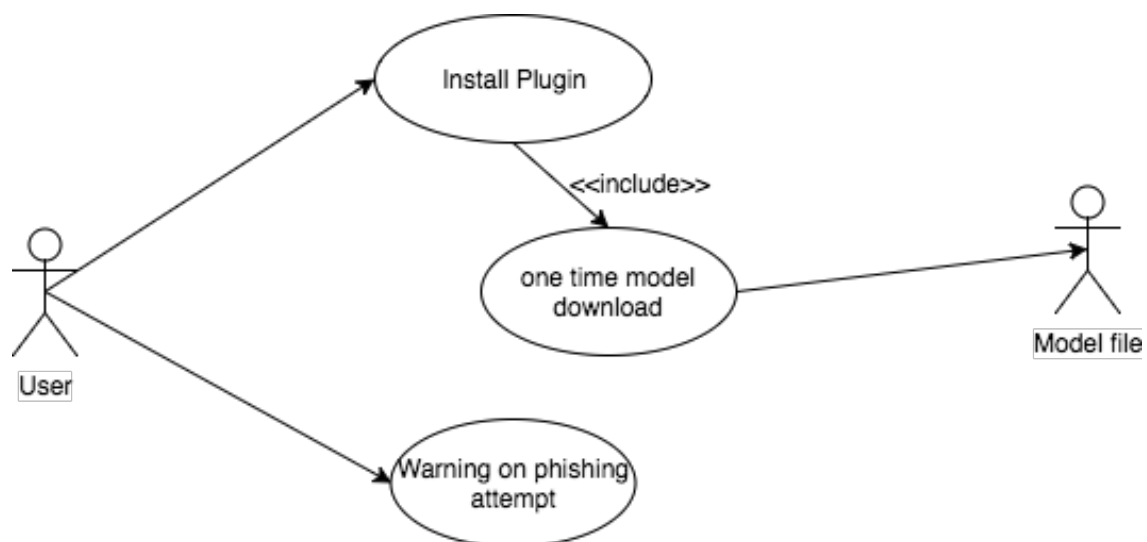


Figure 3.1 Use case diagram of the system

3.4.2 Sequence diagram

The sequence of interactions between the user and the plugin are shown in the figure 3.2

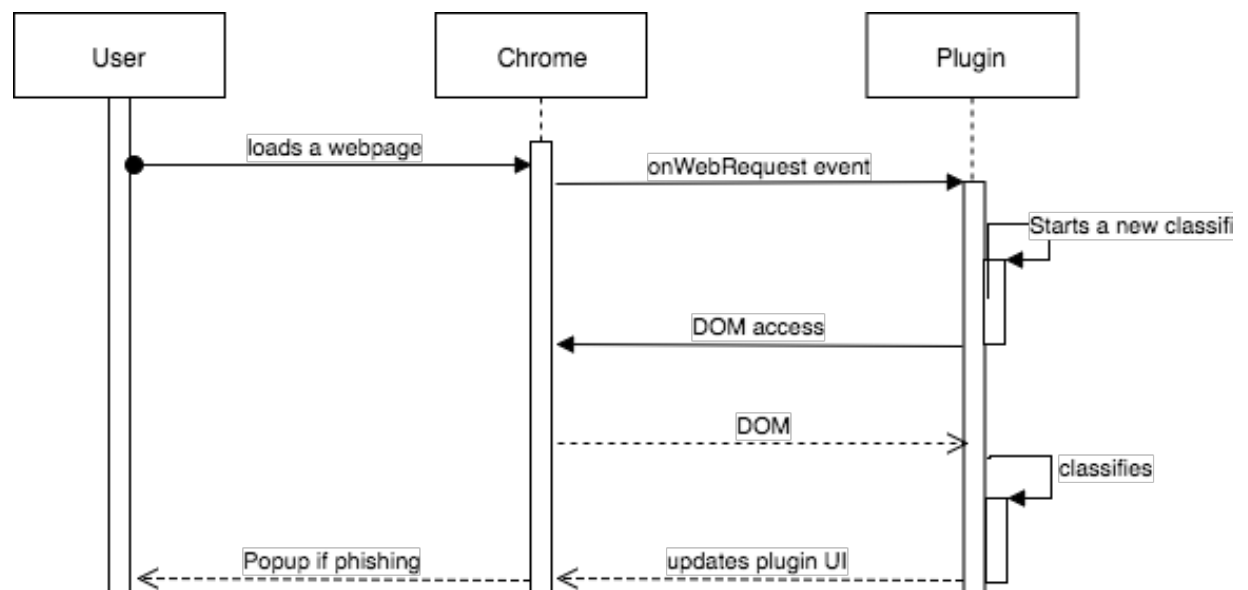


Figure 3.2 System Sequence diagram

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The block diagram of the entire system is shown in the figure 4.1. A Random Forest classifier is trained on phishing sites dataset using python scikit-learn. A JSON format to represent the random forest classifier has been devised and the learned classifier is exported to the same. A browser script has been implemented which uses the exported model JSON to classify the website being loaded in the active browser tab.

The system aims at warning the user in the event of phishing. Random Forest classifier on 17 features of a website is used to classify whether the site is phishing or legitimate. The dataset arff file is loaded using python arff library and 17 features are chosen from the existing 30 features. Features are selected on basis that they can be extracted completely offline on the client side without being dependent on a web service or third party. The dataset with chosen features are then separated for training and testing. Then the Random Forest is trained on the training data and exported to the above mentioned JSON format. The JSON file is hosted on a URL.

The client side chrome plugin is made to execute a script on each page load and it starts to extract and encode the above selected features. Once the features are encoded, the plugin then checks for the exported model JSON in cache and downloads it again incase it is not there in cache.

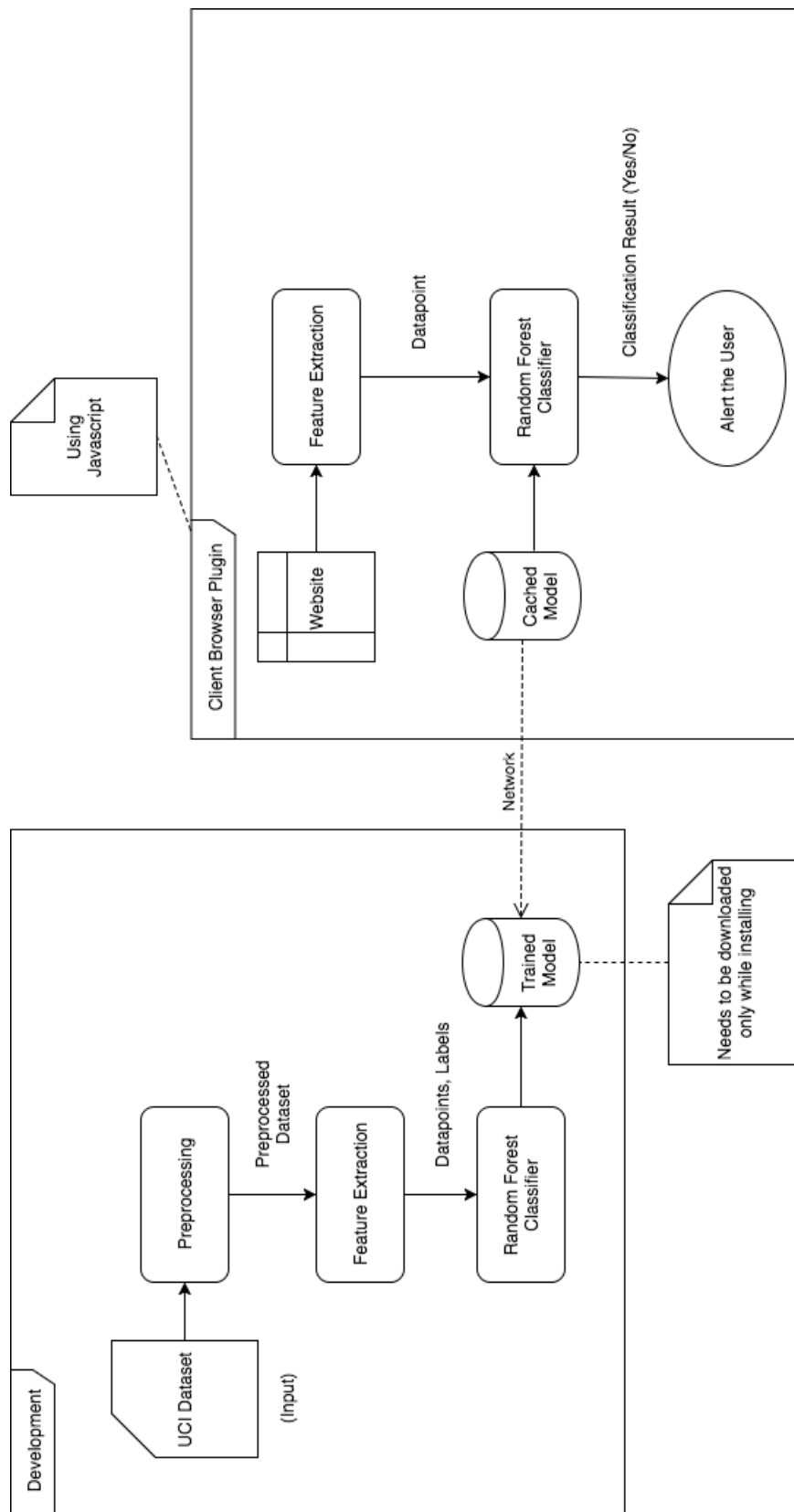


Figure 4.1 System Architecture

With the encoded feature vector and model JSON, the script can run the classification. Then a warning is displayed to the user, incase the website is classified as phishing. The entire system is designed lightweight so that the detection will be rapid.

4.2 UI DESIGN

A simple and easy to use User Interface has been designed for the plugin using HTML and CSS. The UI contains a large circle indicating the percentage of the legitimacy of the website in active tab. The circle also changes its colour with respect to the classification output (Green for legitimate website and Light Red for phishing). Below the circle, the analysis results containing the extracted features are displayed in the following colour code.

Green	-	Legitimate
Yellow	-	Suspicious
Light Red	-	Phishing

The plugin also displays a alert warning incase of phishing to prevent the user from entering any sensitive information on the website. The test results such as precision, recall and accuracy are displayed in a separate screen. The UI is shown in figure 4.2

4.3 CLASS DIAGRAM

The class diagram of the entire Machine Translation system is shown in figure 4.3. This diagram depicts the functions of various modules in the system clearly. It also shows the interaction between the modules of the system thereby providing a clear idea for implementation.

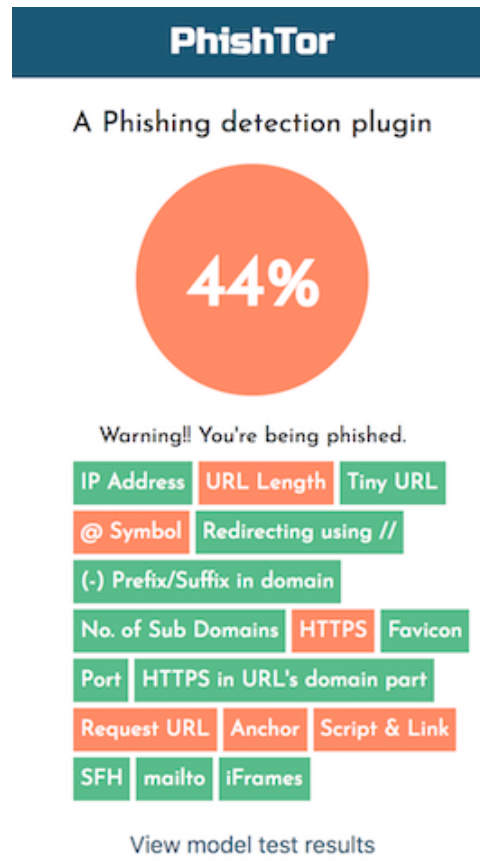


Figure 4.2 UI Design

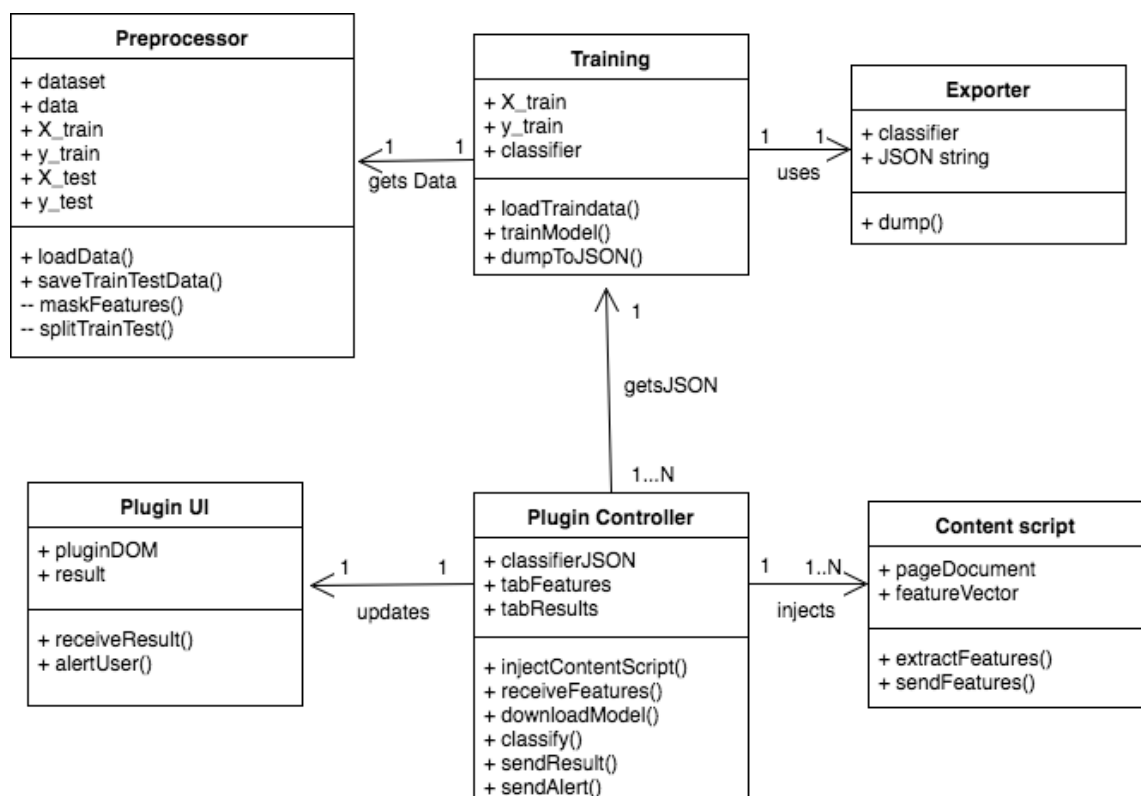


Figure 4.3 Class Diagram

4.4 MODULE DESIGN

4.4.1 Preprocessing

The dataset is downloaded from UCI repository and loaded into a numpy array. The dataset consists of 30 features, which needs to be reduced so that they can be extracted on the browser. Each feature is experimented on the browser so that it will be feasible to extract it without using any external web service or third party. Based on the experiments, 17 features have been chosen out of 30 without much loss in the accuracy on the test data. More number of features increases the accuracy and on the other hand, reduces the ability to detect rapidly considering the feature extraction time. Thus a subset of features is chosen in a way that the tradeoff is balanced.

IP address	Degree of subdomain	Anchor tag href domains
URL length	HTTPS	Script & link tag domains
URL shortener	Favicon domain	Empty server form handler
@' in URL	TCP Port	Use of mailto
Redirection with '/'	HTTPS in domain name	Use of iFrame
-' in domain	Cross domain requests	

Table 4.1 Webpage Features

Then the dataset is split into training and testing set with 30% for testing. Both the training and testing data are saved to disk.

4.4.2 Training

The training data from the preprocessing module is loaded from the disk. A random forest classifier is trained on the data using scikit-learn library. Random Forest is an ensemble learning technique and thus

an ensemble of 10 decision tree estimators is used. Each decision tree follows CART algorithm and tries to reduce the gini impurity.

$$Gini(E) = 1 - \sum_{j=1}^c p_j^2$$

The cross validation score is also calculated on the training data. The F1 score is calculated on the testing data. Then the trained model is exported to JSON using the next module.

4.4.3 Exporting Model

Every machine learning algorithm learns its parameter values during the training phase. In Random Forest, each decision tree is an independent learner and each decision tree learns node threshold values and the leaf nodes learn class probabilities. Thus a format needs to be devised to represent the Random Forest in JSON.

The overall JSON structure consists of keys such as number of estimators, number of classes and etc. Further it contains an array in

```
{
  "n_features": 17,
  "n_classes": 2,
  "classes": [-1, 1],
  "n_outputs": 1,
  "n_estimators": 10,
  "estimators": [{
    "type": "split",
    "threshold": "<float>",
    "left": {},
    "right": {}
  },
  {
    "type": "leaf",
    "value": ["<float>", "<float>"]
  }
  ]
}
```

Figure 4.4 Random Forest JSON structure

which each value is an estimator represented in JSON. Each decision tree is encoded as a JSON tree with nested objects containing threshold for that node and left and right node objects recursively.

4.4.4 Plugin Feature Extraction

The above mentioned 17 features needs to be extracted and encoded for each webpage in realtime while the page is being loaded. A content script is used so that it can access the DOM of the webpage. The content script is automatically injected into each page while it loads. The content script is responsible to collect the features and then send them to the plugin. The main objective of this work is not to use any external web service and the features needs to be independent of network latency and the extraction should be rapid. All these are made sure while developing techniques for extraction of features.

Once a feature is extracted it is encoded into values $\{-1, 0, 1\}$ based on the following notation.

-1	-	Legitimate
0	-	Suspicious
1	-	Phishing

The feature vector containing 17 encoded values is passed on to the plugin from the content script.

4.4.5 Classification

The feature vector obtained from the content script is ran through the Random Forest for classification. The Random Forest parameters JSON is downloaded and cached in disk. The script tries to load the JSON from disk and incase of cache miss, the JSON is downloaded again.

A javascript library has been developed to mimic the Random Forest behaviour using the JSON by comparing feature vector against the threshold of the nodes. The output of the binary classification is based on the leaf node values and the user is warned if the webpage is classified as phishing.

4.5 COMPLEXITY ANALYSIS

4.5.1 Time Complexity

The time complexity of each module of the system is shown in Table 4.2

S.No	Module	Complexity
1	Preprocessing	$O(n)$
2	Training	$O(E * v * n \log(n))$
3	Exporting model	$O(E * n \log(n))$
4	Plugin feature extraction	$O(v)$
5	Classification	$O(E * n \log(n))$

Table 4.2 Time Complexity of various modules

- 'n' denotes number of data points.
- 'E' denotes number of ensembles (decision trees).
- 'v' denotes number of features.

4.5.2 Complexity of the project

- The complexity of the project lies in balancing the tradeoff between accuracy and rapid detection. Choosing a subset of features that will make the detection fast and at the same time without much drop in accuracy.

- Porting of scikit-learn python object to javascript compatible format. For example, JSON.
- Reproducing the Random Forest behaviour in javascript reduced the accuracy by a small margin.
- Many features are not feasible to extract without using a external web service. Use of an external web service will again affect the detection time.
- Maintaining rapid detection is important as the system should detect the phishing before the user submit any sensitive information.

CHAPTER 5

SYSTEM DEVELOPMENT

The system is overall split into backend and plugin. The backend consists of dataset preprocessing and training modules. The frontend which is the plugin consists of javascript files for content script and background script including the Random Forest script. The plugin also consists of HTML and CSS files for the user interface. The overall code overview showing the organisation of these various modules can be seen in figure 5.1

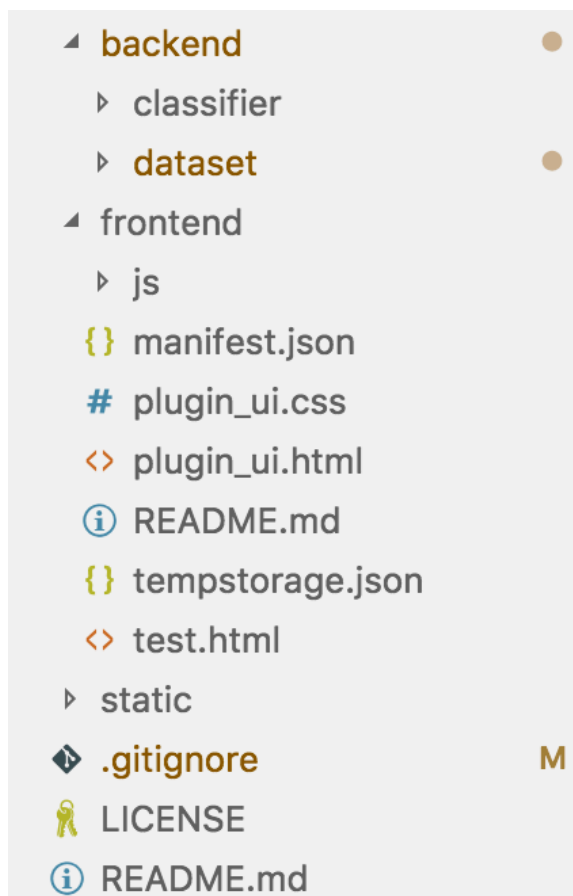


Figure 5.1 Code Overview

5.1 PROTOTYPE ACROSS THE MODULES

The input and output to each module of the system is described in this section.

- **Preprocessing:** This module takes the downloaded dataset in arff format and creates four new files listed as training features, training class labels, testing features, testing class labels.
- **Training:** This module takes the four output files from pre-processor and gives a trained Random Forest object along with the cross validation score on the training set.
- **Exporting model:** This module takes the learned Random Forest classifier object and recursively generates its JSON representation which is written to file in disk.
- **Plugin Feature Extraction:** This module takes a webpage as input and generates a feature vector with 17 encoded features.
- **Classification:** This module takes the feature vector from feature extraction module and the JSON format from the Exporting model module and then gives a boolean output which denotes whether the webpage is legitimate or phishing.

5.2 EXPORTING ALGORITHM

The algorithm used to export Random Forest model as JSON is as follows.

TREE_TO_JSON(NODE):

1. $tree_json \leftarrow \{\}$
2. **if** (node has threshold) **then**
3. $tree_json["type"] \leftarrow "split"$
4. $tree_json["threshold"] \leftarrow node.threshold$
5. $tree_json["left"] \leftarrow TREE_TO_JSON(node.left)$


```

6.   tree_json["right"] ← TREE_TO_JSON(node.right)
7.   else
8.     tree_json["type"] ← "leaf"
9.     tree_json["values"] ← node.values
10.  return tree_json

```

RANDOM_FOREST_TO_JSON(RF):

```

1.  forest_json ← {}
2.  forest_json['n_features'] ← rf.n_features_
3.  forest_json['n_classes'] ← rf.n_classes_
4.  forest_json['classes'] ← rf.classes_
5.  forest_json['n_outputs'] ← rf.n_outputs_
6.  forest_json['n_estimators'] ← rf.n_estimators
7.  forest_json['estimators'] ← []
8.  e ← rf.estimators
9.  for (i ← 0 to rf.n_estimators)
10.   forest_json['estimators'][i] ← TREE_TO_JSON(e[i])
11. return forest_json

```

5.3 DEPLOYMENT DETAILS

The backend requires Python 3 and the Classifier JSON and Test set are served over HTTP using Github. The plugin is distributed as single file and requires Chrome browser to run. The plugin (frontend) is packed into a crx file for distribution.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 DATASET FOR TESTING

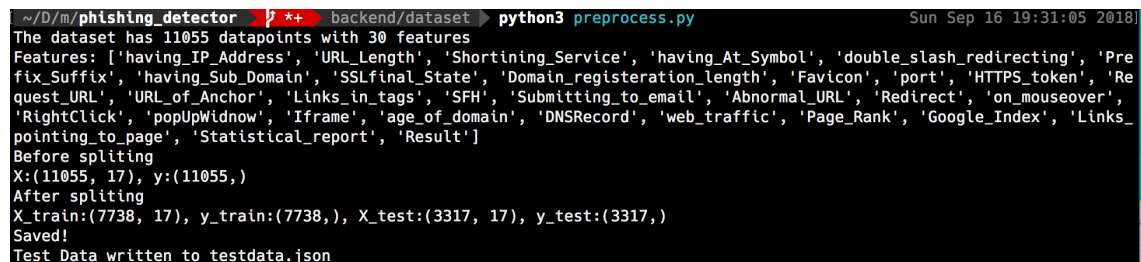
The test set consists of data points separated from the dataset by ratio 70:30. Also the plugin is tested with websites that are listed in phishTank. New phishing sites are also added to PhishTank as soon as they are found. It should be noted that the plugin is able detect new phishing sites too. The results of this module testing as well as the testing of the entire system are summarised below.

6.2 OUTPUT OBTAINED IN VARIOUS STAGES

This section shows the results obtained during module testing.

6.2.1 Preprocessing

The output the preprocessing module is shown in figure 6.1.

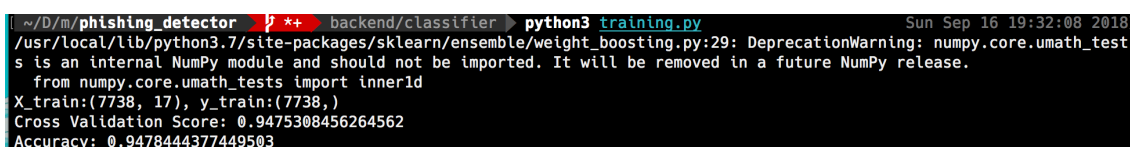
A terminal window showing the execution of a Python script named 'preprocess.py'. The script processes a dataset with 11055 datapoints and 30 features. It lists the features: 'having_IP_Address', 'URL_Length', 'Shortining_Service', 'having_At_Symbol', 'double_slash_redirecting', 'Prefix_Suffix', 'having_Sub_Domain', 'SSLfinal_State', 'Domain_registration_length', 'Favicon', 'port', 'HTTPS_token', 'Request_URL', 'URL_of_Anchor', 'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL', 'Redirect', 'on_mouseover', 'RightClick', 'popUpWidnow', 'Iframe', 'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank', 'Google_Index', 'Links_pointing_to_page', 'Statistical_report', and 'Result'. It then shows the dataset being split into training and testing sets. The training set has 7738 datapoints and the test set has 3317 datapoints. The output is saved to a file named 'testdata.json'.

```
~/D/m/phishing_detector ➤ backend/dataset ➤ python3 preprocess.py
The dataset has 11055 datapoints with 30 features
Features: ['having_IP_Address', 'URL_Length', 'Shortining_Service', 'having_At_Symbol', 'double_slash_redirecting', 'Prefix_Suffix', 'having_Sub_Domain', 'SSLfinal_State', 'Domain_registration_length', 'Favicon', 'port', 'HTTPS_token', 'Request_URL', 'URL_of_Anchor', 'Links_in_tags', 'SFH', 'Submitting_to_email', 'Abnormal_URL', 'Redirect', 'on_mouseover', 'RightClick', 'popUpWidnow', 'Iframe', 'age_of_domain', 'DNSRecord', 'web_traffic', 'Page_Rank', 'Google_Index', 'Links_pointing_to_page', 'Statistical_report', 'Result']
Before splitting
X:(11055, 17), y:(11055,)
After splitting
X_train:(7738, 17), y_train:(7738,) X_test:(3317, 17), y_test:(3317,)
Saved!
Test Data written to testdata.json
```

Figure 6.1 Preprocessing output

6.2.2 Training

The output the training module is shown in figure 6.2.



```

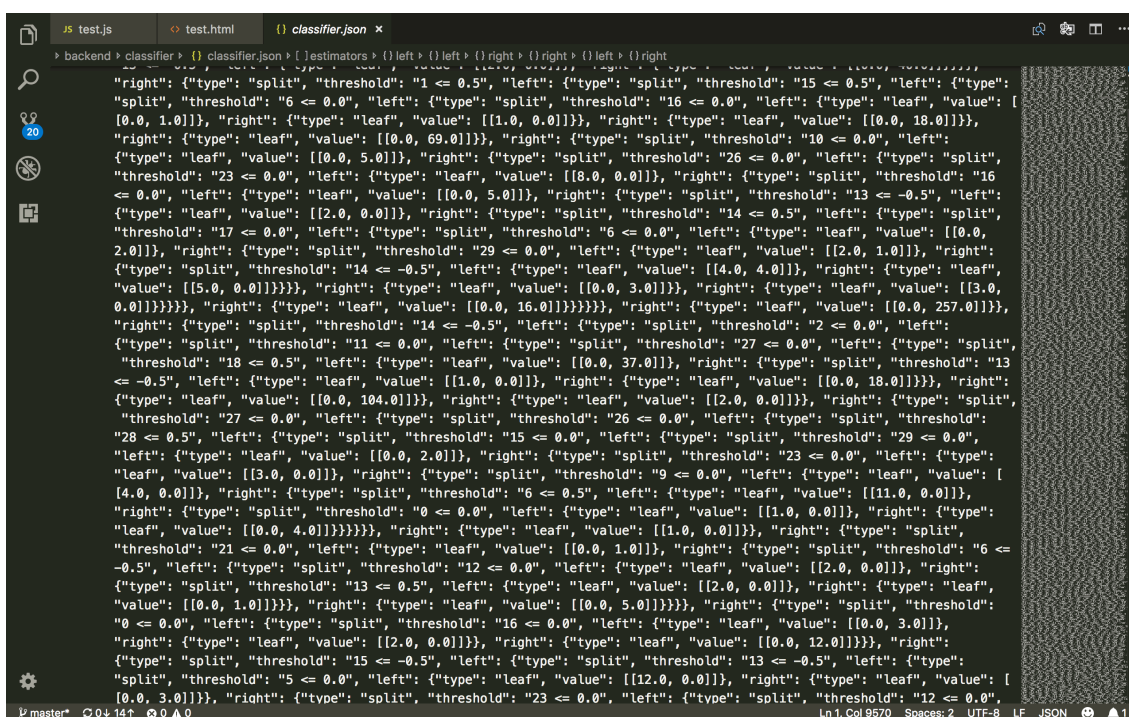
~/D/m/phishing_detector ➤ python3 backend/classifier/training.py
/usr/local/lib/python3.7/site-packages/sklearn/ensemble/weight_boosting.py:29: DeprecationWarning: numpy.core.umath_tests
s is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.
  from numpy.core.umath_tests import inner1d
X_train:(7738, 17), y_train:(7738,)
Cross Validation Score: 0.9475308456264562
Accuracy: 0.9478444377449503

```

Figure 6.2 Training output

6.2.3 Exporting model

The output the export module is shown in figure 6.3. It outputs a JSON file representing the Random Forest parameters.



```


{
  "type": "split", "threshold": "1 <= 0.5", "left": {
    "type": "split", "threshold": "15 <= 0.5", "left": {
      "type": "split", "threshold": "6 <= 0.0", "left": {
        "type": "leaf", "value": [0.0, 1.0]
      },
      "right": {
        "type": "leaf", "value": [1.0, 0.0]
      }
    },
    "right": {
      "type": "leaf", "value": [0.0, 69.0]
    }
  },
  "right": {
    "type": "split", "threshold": "26 <= 0.0", "left": {
      "type": "split", "threshold": "23 <= 0.0", "left": {
        "type": "leaf", "value": [0.0, 5.0]
      },
      "right": {
        "type": "split", "threshold": "16 <= 0.0", "left": {
          "type": "leaf", "value": [0.0, 5.0]
        },
        "right": {
          "type": "split", "threshold": "13 <= -0.5", "left": {
            "type": "leaf", "value": [2.0, 0.0]
          },
          "right": {
            "type": "split", "threshold": "14 <= 0.5", "left": {
              "type": "split", "threshold": "17 <= 0.0", "left": {
                "type": "split", "threshold": "6 <= 0.0", "left": {
                  "type": "leaf", "value": [0.0, 2.0]
                },
                "right": {
                  "type": "split", "threshold": "29 <= 0.0", "left": {
                    "type": "leaf", "value": [2.0, 1.0]
                  },
                  "right": {
                    "type": "split", "threshold": "14 <= -0.5", "left": {
                      "type": "leaf", "value": [4.0, 4.0]
                    },
                    "right": {
                      "type": "leaf", "value": [5.0, 0.0]
                    }
                  }
                },
                "right": {
                  "type": "leaf", "value": [0.0, 3.0]
                }
              }
            },
            "right": {
              "type": "leaf", "value": [0.0, 16.0]
            }
          }
        },
        "right": {
          "type": "split", "threshold": "2 <= 0.0", "left": {
            "type": "split", "threshold": "11 <= 0.0", "left": {
              "type": "split", "threshold": "27 <= 0.0", "left": {
                "type": "split", "threshold": "18 <= 0.5", "left": {
                  "type": "leaf", "value": [0.0, 37.0]
                },
                "right": {
                  "type": "split", "threshold": "13 <= -0.5", "left": {
                    "type": "leaf", "value": [1.0, 0.0]
                  },
                  "right": {
                    "type": "leaf", "value": [0.0, 18.0]
                  }
                }
              },
              "right": {
                "type": "leaf", "value": [0.0, 104.0]
              }
            },
            "right": {
              "type": "leaf", "value": [2.0, 0.0]
            }
          }
        },
        "right": {
          "type": "split", "threshold": "22 <= 0.0", "left": {
            "type": "split", "threshold": "26 <= 0.0", "left": {
              "type": "split", "threshold": "28 <= 0.5", "left": {
                "type": "split", "threshold": "15 <= 0.0", "left": {
                  "type": "split", "threshold": "29 <= 0.0", "left": {
                    "type": "leaf", "value": [0.0, 2.0]
                  },
                  "right": {
                    "type": "split", "threshold": "23 <= 0.0", "left": {
                      "type": "leaf", "value": [3.0, 0.0]
                    },
                    "right": {
                      "type": "split", "threshold": "9 <= 0.0", "left": {
                        "type": "leaf", "value": [4.0, 0.0]
                      },
                      "right": {
                        "type": "split", "threshold": "6 <= 0.5", "left": {
                          "type": "leaf", "value": [11.0, 0.0]
                        },
                        "right": {
                          "type": "split", "threshold": "0 <= 0.0", "left": {
                            "type": "leaf", "value": [1.0, 0.0]
                          },
                            "right": {
                              "type": "leaf", "value": [0.0, 0.0]
                            }
                        }
                      }
                    },
                    "right": {
                      "type": "leaf", "value": [1.0, 0.0]
                    }
                  }
                },
                "right": {
                  "type": "split", "threshold": "21 <= 0.0", "left": {
                    "type": "leaf", "value": [0.0, 1.0]
                  },
                  "right": {
                    "type": "split", "threshold": "6 <= -0.5", "left": {
                      "type": "split", "threshold": "12 <= 0.0", "left": {
                        "type": "leaf", "value": [2.0, 0.0]
                      },
                      "right": {
                        "type": "split", "threshold": "13 <= 0.5", "left": {
                          "type": "leaf", "value": [2.0, 0.0]
                        },
                          "right": {
                            "type": "leaf", "value": [0.0, 1.0]
                          }
                        }
                      },
                      "right": {
                        "type": "leaf", "value": [0.0, 5.0]
                      }
                    }
                  },
                  "right": {
                    "type": "split", "threshold": "0 <= 0.0", "left": {
                      "type": "split", "threshold": "16 <= 0.0", "left": {
                        "type": "leaf", "value": [0.0, 3.0]
                      },
                      "right": {
                        "type": "leaf", "value": [2.0, 0.0]
                      }
                    },
                    "right": {
                      "type": "leaf", "value": [0.0, 12.0]
                    }
                  }
                }
              }
            },
            "right": {
              "type": "split", "threshold": "15 <= -0.5", "left": {
                "type": "split", "threshold": "13 <= -0.5", "left": {
                  "type": "split", "threshold": "5 <= 0.0", "left": {
                    "type": "leaf", "value": [12.0, 0.0]
                  },
                  "right": {
                    "type": "leaf", "value": [0.0, 3.0]
                  }
                },
                "right": {
                  "type": "split", "threshold": "23 <= 0.0", "left": {
                    "type": "split", "threshold": "12 <= 0.0", "left": {
                      "type": "leaf", "value": [0.0, 0.0]
                    },
                    "right": {
                      "type": "leaf", "value": [0.0, 0.0]
                    }
                  },
                  "right": {
                    "type": "leaf", "value": [0.0, 0.0]
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}

```

Figure 6.3 Model JSON

6.2.4 Plugin Feature Extraction

The 17 features extracted for the webpage at thetechcache.science are logged in to the console which is shown in figure 6.4. The features are stored as key value pairs and the values are encoded from -1 to 1 as discussed above.

▼ Object 

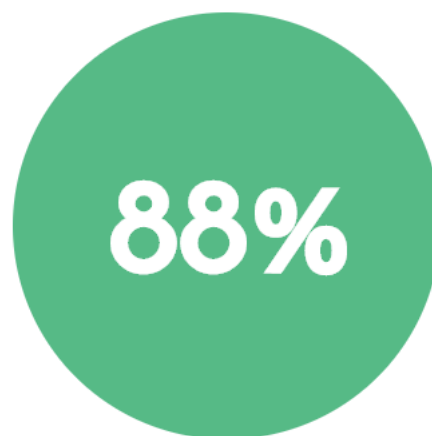
```
(-) Prefix/Suffix in domain: "-1"
@ Symbol: "-1"
Anchor: "-1"
Favicon: "-1"
HTTPS: "-1"
HTTPS in URL's domain part: "-1"
IP Address: "-1"
No. of Sub Domains: "-1"
Port: "-1"
Redirecting using //: "-1"
Request URL: "0"
SFH: "-1"
Script & Link: "0"
Tiny URL: "-1"
URL Length: "-1"
iFrames: "-1"
mailto: "-1"
```

Figure 6.4 Webpage features

6.2.5 Classification

The output of the classification is shown right in the Plugin UI. Green circle indicates legitimate site and Light red indicates phishing.

A Phishing detection plugin



This website is safe to use :)

Figure 6.5 Classification Output

6.3 SAMPLE SCREENSHOTS DURING TESTING

The output of the plugin while visiting a phishing site taken from PhishTank. This site has a low trust value and also the light red circle indicates phishing.

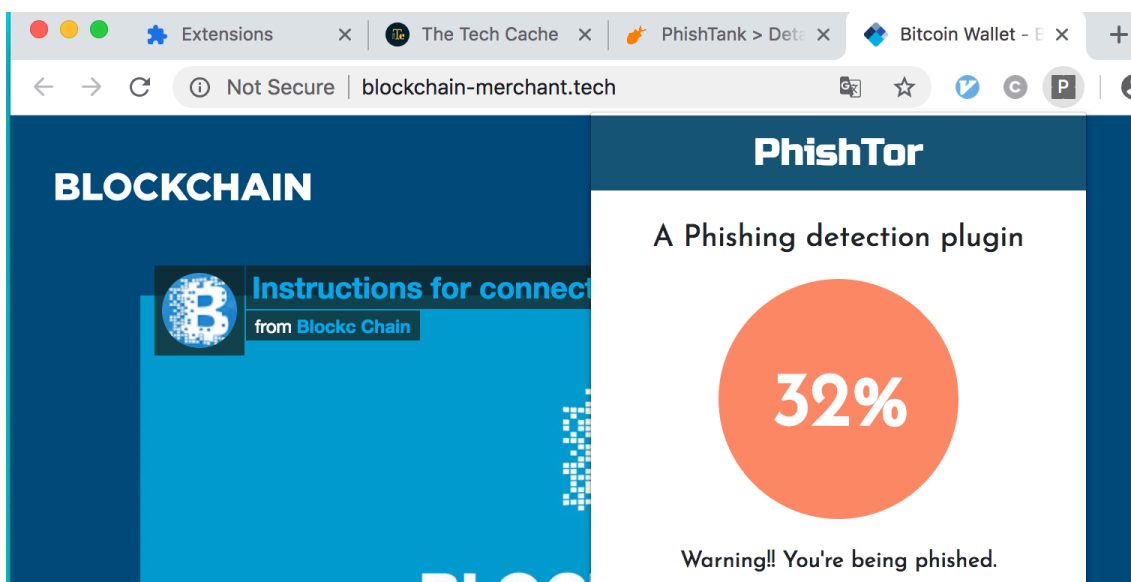


Figure 6.6 Test Output

6.4 PERFORMANCE EVALUATION

The performance of the entire system is evaluated using the standard parameters described below.

6.4.1 Cross Validation score

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called **overfitting**.

To solve this problem, yet another part of the dataset can be held out as a so-called “**validation set**”: training proceeds on the training set,

after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set. However, by partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k “folds”:

A model is trained using k of the folds as training data; the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The score on 10 Fold cross validation is as below

```
print('Cross Validation Score: {0}'.format(np.mean(cross_val_score)))
Cross Validation Score: 0.9476602117325363
```

Figure 6.7 Cross Validation Output

(calculated with cross_val_score of scikit-learn)

6.4.2 F1 score

F1 score is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score: precision is the number of correct positive results divided by the number of all positive results returned by the classifier, and recall is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmon-

ic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$\mathbf{F1 = 2 * (precision * recall) / (precision + recall)}$$

The precision, recall and F1 score of the phishing classifier is calculated manually using javascript on the test data set. The results are shown in the figure 6.8

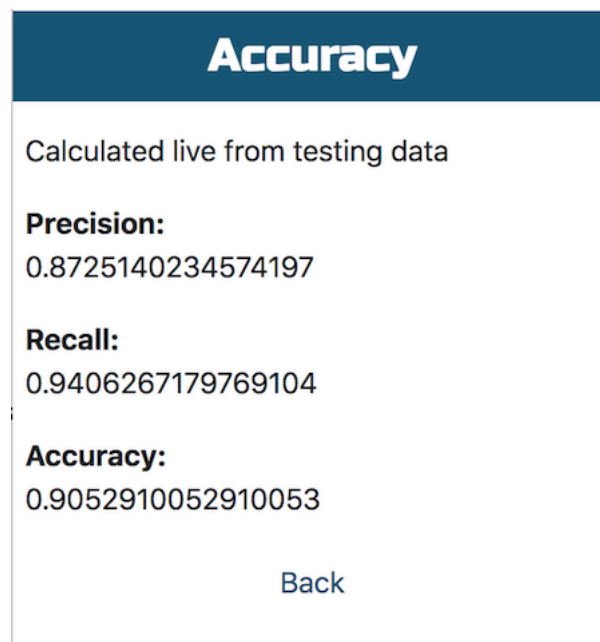


Figure 6.8 Performance measure

CHAPTER 7

CONCLUSIONS

7.1 SUMMARY

This is a phishing website detection system that focuses on client side implementation with rapid detection so that the users will be warned before getting phished. The main implementation is porting of Random Forest classifier to javascript. Similar works often use webpage features that are not feasible to extract on the client side and this results in the detection being dependent on the network. On the other side, this system uses only features that are possible to extract on the client side and thus it is able to provide rapid detection and better privacy. Although using lesser features results in mild drop in accuracy, it increases the usability of the system. This work has identified a subset of webpage feature that can be implemented on the client side without much effect in accuracy.

The port from python to javascript and own implementation of Random Forest in javascript further helped in rapid detection as the JSON representation of the model and the classification script is designed with time complexity in mind. The plugin is able to detect the phishing even before the page loads completely. The F1 score calculated on the test set on the client side is **0.886**.

7.2 CRITICISMS

The system has a lower accuracy than the state-of-the art but it is more usable and the trade-off between accuracy and rapid detection is handled well enough. The chrome extension API restrictions has a small effect on the plugin. Since the features are extracted in content script which is injected on page load, this plugin can't prevent a malicious javascript code from executing. Further the accuracy reduces from 0.94 to 0.886 while porting from python to javascript and this needs to be investigated.

Javascript doesn't support multithreading and browser execute only javascript. Thus the classification can't be made faster by using parallel threads. Currently the results are not cached on the plugin and it's computed repeatedly even for frequently visited sites.

7.3 FUTURE WORK

The classifier is currently trained on 17 features which can be increased provided that, they don't make the detection slower or result in loss of privacy. The extension can made to cache results of frequently visited sites and hence reducing computation. But this may result in pharming attack being undetected. A solution needs to be devised for caching of results without losing the ability to detect pharming. The classification in javascript can be done using WorkerThreads which may result in better classification time. Thus a lot of improvements and enhancements are possible this system offers a more usable solution in the field of phishing detection.

REFERENCES

- [1] A. Subasi, E. Molah, F. Almkallawi, and T. J. Chaudhery, “Intelligent phishing website detection using random forest classifier,” *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Nov. 2017.
- [2] “UCI Machine Learning Repository: Phishing Websites Data Set,” [Online]. Available: [https://archive.ics.uci.edu/ml/datasets/phishing websites](https://archive.ics.uci.edu/ml/datasets/phishing%20websites).
- [3] J.-H. Li and S.-D. Wang, “PhishBox: An Approach for Phishing Validation and Detection,” *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2017.
- [4] A. A. Ahmed and N. A. Abdullah, “Real time detection of phishing websites,” *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2016.
- [5] R. Aravindhan, R. Shanmugalakshmi, K. Ramya, and S. C., “Certain investigation on web application security: Phishing detection and phishing target discovery,” *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2016.

- [6] S. B. K.p, “Phishing Detection in Websites Using Neural Networks and Firefly,” *International Journal Of Engineering And Computer Science*, 2016.
- [7] “An Efficient Approaches For Website Phishing Detection Using Supervised Machine Learning Technique,” *International Journal of Advance Engineering and Research Development*, vol. 2, no. 05, 2015.
- [8] S. Gupta and A. Singhal, “Phishing URL detection by using artificial neural network with PSO,” *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, 2017.
- [9] Ammar ALmomani, G. B. B, Tat-Chee Wan, Altyeb Altaher, and Selvakumar Manickam, “Phishing Dynamic Evolving Neural Fuzzy Framework for ...,” Jan-2013. [Online]. Available: <https://arxiv.org/pdf/1302.0629>.