AFNN-R: An Effective and Accurate Phishing Websites Detection using Optimal Feature Selection Neural Network and Random Forest Classifier

Abstract

Phishing attacks are now a big threat to people's daily life and networking environment. Through disguising illegal URLs as legitimate ones, attackers can induce users to visit the phishing URLs to get private information and other benefits. Effective methods of detecting phishing websites are urgently needed to alleviate the threats posed by phishing attacks. The objective of our proposed approach is to develop a model for effective and accurate phishing website prediction and to avoid some useless or small impact features and falling into the problem of overfitting. FVV, Feature Validity Value is firstly introduced to evaluate the impact of sensitive features. Optimal feature selection algorithm, this algorithm calculates the FFV values of all features of the input URLs and their relevant websites at first. Then, a threshold is set to select sensitive features to construct an optimal feature vector. Through this algorithm, many useless and small influence features are pruned. Due to no disturbance from these redundant features, the overfitting problem of the underlying neural network is alleviated. Meanwhile, this algorithm is also able to reduce the time cost of the process of phishing websites detection. The selected optimal features are used to train the underlying neural network and, finally, an optimal classifier (Random Forest Classifier) is constructed to detect the phishing websites.

CHAPTER 1

INTRODUCTION

1.1 Phishing URL Detection with ML

Phishing is a form of fraud in which the attacker tries to learn sensitive information such as login credentials or account information by sending as a reputable entity or person in email or other communication channels.

Typically a victim receives a message that appears to have been sent by a known contact or organization. The message contains malicious software targeting the user's computer or has links to direct victims to malicious websites in order to trick them into divulging personal and financial information, such as passwords, account IDs or credit card details.

Phishing is popular among attackers, since it is easier to trick someone into clicking a malicious link which seems legitimate than trying to break through a computer's defence systems. The malicious links within the body of the message are designed to make it appear that they go to the spoofed organization using that organization's logos and other legitimate contents.

Phishing domain (or Fraudulent Domain) characteristics, the features that distinguish them from legitimate domains, why it is important to detect these domains, and how they can be detected using machine learning and natural language processing techniques.

1.2 Characteristics of Phishing Domains

Let's check the URL structure for the clear understanding of how attackers think when they create a phishing domain.

Uniform Resource Locator (URL) is created to address web pages. The figure below shows relevant parts in the structure of a typical URL.

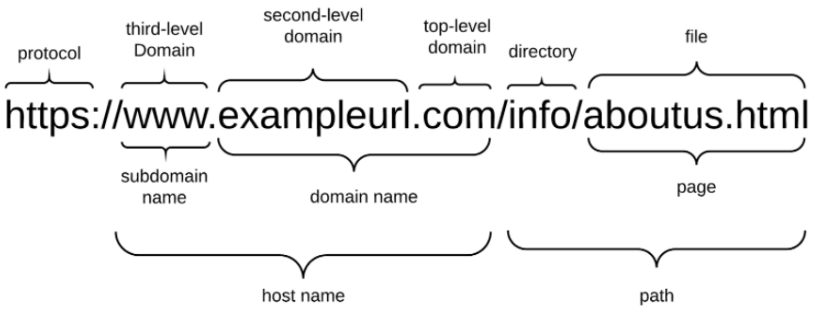


Fig 1.1 URL structure

It begins with a protocol used to access the page. The fully qualified domain name identifies the server who hosts the web page. It consists of a registered domain name (second-level domain) and suffix which we refer to as top-level domain (TLD). The domain name portion is constrained since it has to be registered with a domain name Registrar. A Host name consists of a subdomain name and a domain name.

A phisher has full control over the subdomain portions and can set any value to it. The URL may also have a path and file components which, too, can be changed by the phisher at will. The subdomain name and path are fully controllable by the phisher. We use the term FreeURL to refer to those parts of the URL in the rest of the article.

The attacker can register any domain name that has not been registered before. This part of URL can be set only once. The phisher can change FreeURL at any time to create a new URL. The reason security defenders struggle to detect phishing domains is because of the unique part of the website domain (the FreeURL). When a domain detected as a fraudulent, it is easy to prevent this domain before a user access to it.

Some threat intelligence companies detect and publish fraudulent web pages or IPs as blacklists, thus preventing these harmful assets by others is getting easier. The attacker must intelligently choose the domain names because the aim should be convincing the users, and then setting the FreeURL to make detection difficult. Let’s analyse an example given below.

http://paypal.com-webappsuserid29348325limited.active-userid.com/webapps/89980/

protocol	http://
Domain name	active-userid.com
path	/webapps/89980/
Subdomain item1	com-webappsuserid29348325limited
Subdomain item2	paypal

Fig 1.2URL Domains

Although the real domain name is active-userid.com, the attacker tried to make the domain look like paypal.com by adding FreeURL. When users see paypal.com at the beginning of the URL, they can trust the site and connect it, then can share their sensitive information to the fraudulent site. This is a frequently used method by attackers. Other methods that are often used by attackers are Cybersquatting and Typo squatting.

Cybersquatting (also known as domain squatting), is registering, trafficking in, or using a domain name with bad faith intent to profit from the goodwill of a trademark belonging to someone else. The cyber squatter may offer selling the domain to a person or company who owns a trademark contained within the name at an inflated price or may use it for fraudulent purposes such as phishing. For example, the name of your company is "abcompany" and you register as abcompany.com. Then phishers can register abcompany.net, abcompany.org, abcompany.biz and they can use it for fraudulent purpose.

Typosquatting, also called URL hijacking, is a form of cybersquatting which relies on mistakes such as typographical errors made by Internet users when inputting a website address into a web browser or based on typographical errors that are hard to notice while quick reading. URLs which are created with Typosquatting looks like a trusted domain.

A user may accidentally enter an incorrect website address or click a link which looks like a trusted domain, and in this way, they may visit an alternative website owned by a phisher.A famous example of Typosquatting is **goggle.com**, an extremely dangerous website.

Another similar thing is **yutube.com**, which is similar to **goggle.com** except it targets **YouTube users**. Similarly, **www.airfrance.com** has been typo squatted as **www.arifrance.com**, diverting users to a website peddling discount travel. Some other examples; **paywpal.com**, **microroft.com**, **applle.com**, **appie.com**.

1.3 Features Used for Phishing Domain Detection

There are a lot of algorithms and a wide variety of data types for phishing detection in the academic literature and commercial products. A phishing URL and the corresponding page have several features which can be differentiated from a malicious URL. For example; an attacker can register long and confusing domain to hide the actual domain name **(**Cybersquatting, Typosquatting).

In some cases attackers can use direct IP addresses instead of using the domain name. This type of event is out of our scope, but it can be used for the same purpose. Attackers can also use short domain names which are irrelevant to legitimate brand names and don't have any FreeUrl addition. But these type of web sites are also out of our scope, because they are more relevant to fraudulent domains instead of phishing domains.

Beside URL-Based Features, different kinds of features which are used in machine learning algorithms in the detection process of academic studies are used. Features collected from academic studies for the phishing domain detection with machine learning techniques are grouped as given below.

1. URL-Based Features
2. Domain-Based Features
3. Page-Based Features
4. Content-Based Features

1.3.1 URL-Based Features

URL is the first thing to analyse a website to decide whether it is a phishing or not. As we mentioned before, URLs of phishing domains have some distinctive points. Features which are related to these points are obtained when the URL is processed. Some of URL-Based Features are given below.

- Digit count in the URL
- Total length of URL
- Checking whether the URL is Typo squatted or not. (google.com → goggle.com)
- Checking whether it includes a legitimate brand name or not (apple-icloud-login.com)
- Number of subdomains in URL
- Is Top Level Domain (TLD) one of the commonly used one?

1.3.2 Domain-Based Features

The purpose of Phishing Domain Detection is detecting phishing domain names. Therefore, passive queries related to the domain name, which we want to classify as phishing or not, provide useful information to us. Some useful Domain-Based Features are given below.

- Its domain name or its IP address in blacklists of well-known reputation services?
- How many days passed since the domain was registered?
- Is the registrant name hidden?

1.3.3 Page-Based Features

Page-Based Features are using information about pages which are calculated reputation ranking services. Some of these features give information about how much reliable a web site is. Some of Page-Based Features are given below.

- Global Pagerank
- Country Pagerank
- Position at the Alexa Top 1 Million Site

Some Page-Based Features give us information about user activity on target site. Some of these features are given below. Obtaining these types of features is not easy. There are some paid services for obtaining these types of features.

- Estimated Number of Visits for the domain on a daily, weekly, or monthly basis
- Average Pageviews per visit
- Average Visit Duration
- Web traffic share per country
- Count of reference from Social Networks to the given domain
- Category of the domain
- Similar websites etc.

1.3.4 Content-Based Features

Obtaining these types of features requires active scan to target domain. Page contents are processed for us to detect whether target domain is used for phishing or not. Some processed information about pages are given below.

- Page Titles
- Meta Tags
- Hidden Text
- Text in the Body

- Images etc.

By analysing these information, we can gather information such as;

- Is it required to login to website
- Website category
- Information about audience profile etc.

All of features explained above are useful for phishing domain detection. In some cases, it may not be useful to use some of these, so there are some limitations for using these features. For example, it may not be logical to use some of the features such as Content-Based Features for the developing fast detection mechanism which is able to analyze the number of domains between 100.000 and 200.000. Another example would be, if we want to analyze new registered domains Page-Based Features is not very useful. Therefore, the features that will be used by the detection mechanism depends on the purpose of the detection mechanism. Which features to use in the detection mechanism should be selected carefully.

1.4 Detection Process

Detecting Phishing Domains is a classification problem, so it means we need labeled data which has samples as phish domains and legitimate domains in the training phase. The dataset which will be used in the training phase is a very important point to build successful detection mechanism. We have to use samples whose classes are precisely known. So it means, the samples which are labelled as phishing must be absolutely detected as phishing. Likewise the samples which are labelled as legitimate must be absolutely detected as legitimate. Otherwise, the system will not work correctly if we use samples that we are not sure about.

For this purpose, some public datasets are created for phishing. Some of the well-known one is Phish Tank. These data sources are used commonly in academic studies.Collecting legitimate domains is another problem. For this purpose, site reputation services are commonly used. These services analyse and rank available websites. This

ranking may be global or may be country-based. Ranking mechanism depends on a wide variety of features.

The websites which have high rank scores are legitimate sites which are used very frequently. One of the well-known reputation ranking service is Alexa. Researchers are using top lists of Alexa for legitimate sites.

When we have raw data for phishing and legitimate sites, the next step should be processing these data and extract meaningful information from it to detect fraudulent domains. The dataset to be used for machine learning must actually consist these features. So, we must process the raw data which is collected from Alexa, Phish tank or other data resources, and create a new dataset to train our system with machine learning algorithms. The feature values should be selected according to our needs and purposes and should be calculated for every one of them.

There so many machine learning algorithms and each algorithm has its own working mechanism. In this article, we have explained ***Decision Tree Algorithm***, because I think, this algorithm is a simple and powerful one.

Initially, as we mentioned above, phishing domain is one of the classification problem. So, this means we need labelled instances to build detection mechanism. In this problem we have two classes: **(1)** phishing and **(2)** legitimate.

When we calculate the features that we've selected our needs and purposes, our dataset looks like in figure below. In our examples, we selected 12 features, and we calculated them. Thus we generated a dataset which will be used in training phase of machine learning algorithm.

No.	1: domain	2: tld	3: brandName	4: editDbrandName	5: digitCount	6: length	7: isKnownTld	8: www	9: keywords	10: punnyCode	11: randomDomain	12:
	String	String	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	Numeric	
...	ayanasalon	com	0.0	1.0	0.0	10.0	0.0	0.0	0.0	0.0	0.0	
...	esteticabrasilbeauty	com	0.0	1.0	0.0	20.0	0.0	0.0	0.0	0.0	0.0	
...	erate365	com	0.0	1.0	3.0	8.0	0.0	0.0	0.0	0.0	0.0	
...	upstatescbusiness	com	1.0	1.0	0.0	17.0	0.0	0.0	1.0	0.0	0.0	
...	6-4c	com	0.0	0.0	2.0	4.0	0.0	0.0	0.0	0.0	0.0	
...	services-confirmatio...	com	1.0	1.0	0.0	23.0	0.0	0.0	1.0	0.0	0.0	
...	hmsinformatica	com	1.0	1.0	0.0	14.0	0.0	0.0	1.0	0.0	0.0	

Fig 1.3Dataset Features

A Decision Tree can be considered as an improved nested-if-else structure. Each features will be checked one by one. An example tree model is given below.

Generating a tree is the main structure of detection mechanism. Yellow and elliptical shaped ones represent features and these are called nodes. Green and angular ones represent classes and these are called leaves. The *length* is checked when an example arrives and then the other features are checked according to the result. When the journey of the samples is completed, the class that a sample belongs to will become clear.

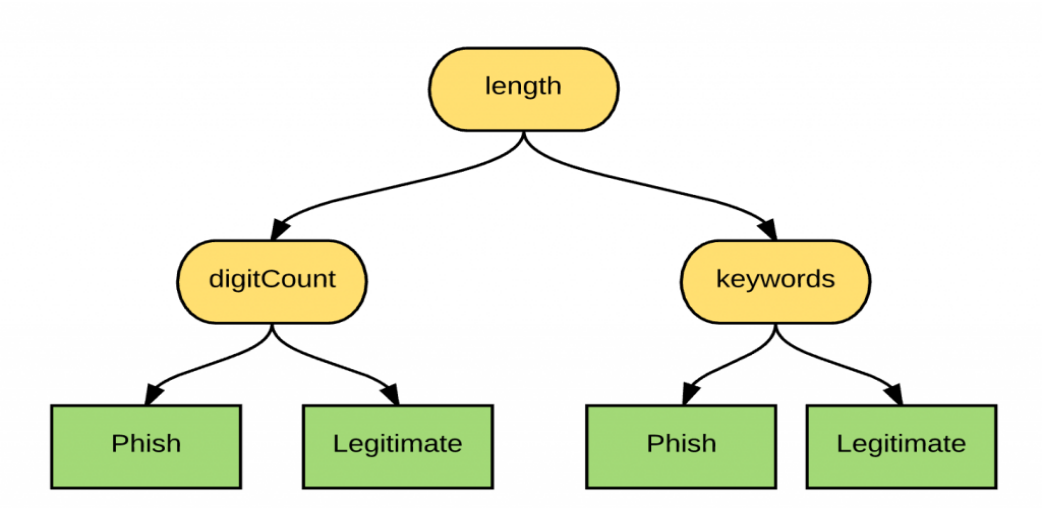


Fig 1.4Decision Tree

Now, the most important question about Decision Trees is not answered yet. The question is that which feature will be located as the root? And which ones must come after

the root? Choosing features intelligently effects efficiency and success rate of algorithms directly.

So, how does decision tree algorithm select features?

Decision Tree uses an information gain measure which indicates how well a given feature separates the training examples according to their target classification. The name of the method is Information Gain. The mathematical equation of information gain method is given below.

$$Gain(S, A) = \underbrace{Entropy(S)}_{\text{original entropy of S}} - \underbrace{\sum_{v \in values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)}_{\text{relative entropy of S}}$$

High Gain score means that the feature has a high distinguishing ability. Because of this, the feature which has maximum gain score is selected as the root. **Entropy** is a statistical measure from information theory that characterizes (im-) purity of an arbitrary collection S of examples. The mathematical equation of Entropy is given below.

$$H(S) \equiv \sum_{i=1}^{n} -p_i \log_2 p_i$$

Original Entropy is a constant value, Relative Entropy is changeable. Low Relative Entropy Score means high purity, likewise high Relative Entropy Score means low purity. As we move down the tree, we want to increase the purity, because high purity on the leaf implies high success rate.

In the training phase, dataset is divided into two parts by comparing the feature values. In our example we have 14 samples. “+” sign representing phishing class, and “-”

sign representing legitimate class. We divided these samples into two parts according to the *length* feature. Seven of them settle right, the other seven of them settle left. As shown in the figure below, right part of tree has high purity, so it means low Entropy Score (E), likewise left part of tree has low purity and high Entropy Score (E). All calculations were done according to the equations given above. Information Gain Score about the *length* feature is 0,151.

The Decision Tree Algorithm calculates this information for every feature and selects features with maximum Gain scores. To growth the tree, leaves are changed as a node which represents a feature. As the tree grows downwards, all leaves will have high purity. When the tree is big enough, the training process is completed.

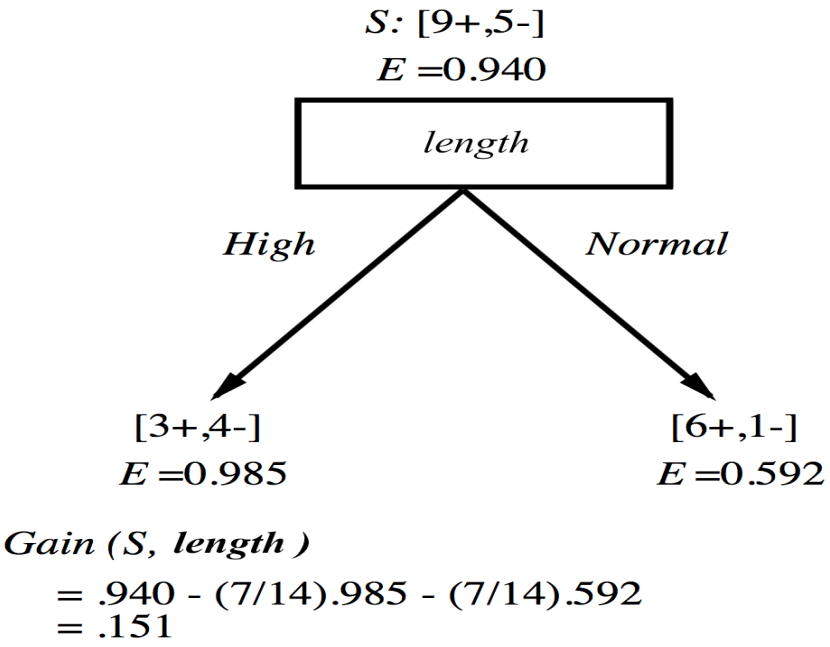


Fig 1.5Decision Tree Algorithm for Gain scores

The Tree created by selecting the most distinguishing features represents model structure for our detection mechanism. Creating mechanism which has high success rate depends on training dataset. For the generalization of system success, the training set must be consisted of a wide variety of samples taken from a wide variety of data sources. Otherwise, our system may working with high success rate on our dataset, but it cannot work successfully on real world data.

Phishing is a form of cyber-attack typically performed by sending false correspondences that seem to be originated from a legitimate source. The objective of such attack is to gain access to sensitive information such as credit card numbers, credential data, or even to download and activate malware applications and viruses on the target machines.

One can say, it is almost essential to have an online presence to perform the necessary transactions like banking, e-commerce, social networking. On the other hand, the significance of the World Wide Web has consistently been expanding. The web is not only imperative for individual clients, but also for organizations to function effectively.

CHAPTER 2

LITERATURE REVIEW

2.1 Detecting Phishing Websites through Deep Reinforcement Learning

Phishing is the simplest form of cybercrime with the objective of baiting people into giving away delicate information such as individually recognizable data, banking and credit card details, or even credentials and passwords. This type of simple yet most effective cyber-attack is usually launched through emails, phone calls, or instant messages. The credential or private data stolen are then used to get access to critical records of the victims and can result in extensive fraud and monetary loss. Hence, sending malicious messages to victims is a stepping stone of the phishing procedure.

Aphisherusually setups a deceptive website, where the victims are conned into entering credentials and sensitive information. It is therefore important to detect these types of malicious websites before causing any harmful damages to victims. Inspired by the evolving nature of the phishing websites, this paper introduces a novel approach based on deep reinforcement learning to model and detect malicious URLs. The proposed model is capable of adapting to the dynamic behavior of the phishing websites and thus learn the features associated with phishing website detection.

In recent years, the application of various kinds of machine learning algorithms to the classical classification problem and in particular to security and malware detection has received tremendous attention and interest from research community. Furthermore, with the advancement of computational power, deep learning algorithms have created a new chapter in pattern recognition and artificial intelligence. As a result, many classification, decision, and automation problems are now can be formulated through these sophisticated learning algorithms. Deep learning-based approaches are particularly effective when the number of features involved in the computation is large.The proposed approach is robust, dynamic, and self-adaptive since reinforcement learning-based algorithms can estimate a solution (i.e., action) based on the stochastic state conversions and the rewards for choosing an action for that state.This paper presents a deep reinforcement learning-based model for detecting phishing website by analyzing the given URLs. The model itself is self-adaptive to the changes in theURL structure. The problem of detecting phishing websites isan instance of the

classical classification problem. Therefore,we have developed a reinforcement learning model usingdeep neural network, to solve this classification problem.

We have used our model on a balanced and labelled dataset of legitimate and malicious URLs in which 14 lexical features were extracted from the given URLs to train the model. The performance is measured using precision, recall, accuracy and F-measure. The key contributions of this paper are as follows:

1) Model the identification of phishing websites through Reinforcement Learning (RL), where an agent learns the value function from the given input URL in order to perform the classification task.

2) Map the sequential decision making process for classification using a deep neural network based implementation of Reinforcement Learning.

3) Evaluate the performance of the deep reinforcement learning-based phishing URL classifier and compare its performance with the existing phishing URL classifiers.

The reinforcement learning approach has been utilized to gain proficiency for optimal behavior. This adaptive learning paradigm is defined as the problem of an “agent” to perform an action based on a “trial and error” basis through communications with an unknown “environment” which provides feedback in the form of numerical “rewards”.

A typical URL has two principle parts: (1) Protocol: Specifies the protocol to be used for communication between user and web server, (2) Resource identifier: indicating the IP address or the domain space where the resource is located. A colon and two forward slashes separate the protocol from resource identifier.

There are a certain characteristics of websites that helps in distinguishing between phishing sites from the legitimate ones. Examples of such characteristics include: long URLs, IP address in URLs, and request access to additional URLs in which these characteristics are the indications of being phishing websites. The website features in four groups: (1) Anomaly-based, (2) Address bar-based, (3) HTML and JavaScript-based and (4) Domain-based. We followed the proposed work in to build our set of 14 features.

There are some other deep learning based algorithms that should be examined for the problem stated in this paper such as LSTM. Moreover this classifier can be extended for other binary classification problems like Web spam detection and presence of malicious bots in the network. RL based approach being more adaptive, the classifier can be extended for mitigating various privacy and security concerns in wearable devices.

2.2 Effective phishing website detection based on improved BP neural network and dual feature evaluation

Nowadays, phishing poses a big threat to people's daily network environment. By phishing, attackers obtain the network users private information by inducing them to open illegal websites. Due to the active learning ability and preferable classifying ability for many datasets, BP neural network is an important heuristic machine learning method in phishing websites detection and prevention. However, improper selection of initial parameters, such as the initial weight and threshold, will induce the BP neural network into local minimum and slow learning convergence.

Aiming at these problems, this paper proposes DF.GWO-BPNN, an effective phishing website detection model based on the improved BP neural network and dual feature evaluation mechanism. Under this model, the grey wolf algorithm is firstly used to optimize the BP neural network to reasonably select initial parameters.

Then, the dual feature mechanism is used to evaluate the results of the improved BP neural network. By the dual feature evaluation mechanism, the accuracy of phishing website recognition is improved. Meanwhile, the black and white list is used to improve the efficiency of the proposed model. The DF.GWO-BPNN model is compared with some existing phishing website detection models.

Phishing is a kind of cybercrime behavior in which attackers obtain the network users private information by inducing them to open illegal websites. On available of the stolen private information, phishing attackers can get money and other benefits from network users. With the progress of network technology, phishing attackers can use a variety of techniques to makethe phishing websites look legitimate. Phishing websites are becoming more and more capable of avoiding detection.Nowadays, phishing websites are widely flooded in the daily PC and mobile environments. Meanwhile, the number of phishing websites is growing rapidly, which poses a big threat to the people's online life. The distribution and harm of phishing websites are crossing the national borders and becoming a global problem. It is urgently needed effective techniques to prevent and detect phishing websites.

Meanwhile, for the purpose of efficiency, the black and white list is used to cache websites that have already been processed. Experimental results on testing many commonly

used samples have demonstrated thatour proposed DF.GWO-BPNN model is accurate and strong adaptability.

This framework, we propose DF.GWO-BPNN, an effective phishing website detection model based on the improved BP neural network and dual feature evaluation mechanism. In the proposed model, the GWO (Grey Wolf Optimizer) algorithm is used to overcome the shortages of traditional BP neural network.

By optimizing with the GWO, the local minimum and slow coverage problems of BP neural network are avoided. The new the dual feature mechanism is used to evaluate the results of the improved BP neural network. By the dual feature evaluation mechanism, the accuracy of phishing website recognition is improved.

DF.GWO-BPNN is composed of four modules, the Dynamic Hash Library, the Feature Extraction and Classification module, theDF.GWOBPNN Classifier (the BP neural network optimized by GWO) and the ComprehensiveEvaluation module. Specifically, the Dynamic Hash Library caches the black and white lists. For the purpose of improving efficiency, this module is used to cache websites that have already been processed. The Feature Extraction and Classification module extracts and classifies features from the URL composition.

By this module, URL features are divided into two categories, dominant features and the recessive features. URLs with recessive features are feeded to the DF.GWO-BPNN Classifier for further procession. As the core component of the proposed, theDF.GWO-BPNN Classifiers used to classify the phishing websites. Due to the global search ability, the GWO is used to optimize the BP neural network to reasonably select initial parameters.

On constructing this model, we used the GWO algorithm to overcome the shortages of local minimum and slow learning convergence of the BP neural network. In order to elevate the phishing websites recognition rate, we used the dual feature evaluation mechanism to comprehensively evaluate the extracted features from URLs. Meanwhile, the black and white list was used to improve the efficiency of the proposed model. The DF.GWO-BPNN model was compared with some existing phishing website detection models.

The experimental results demonstrated that our proposed model was accurate and strong adaptability for detecting phishing websites. Although the GWO optimization algorithm in this model can get the global optimal, but the convergence speed is reduced in

the later stages. We have already noticed that extracting more explicit features from URLs and enriching the black and white list can alleviate this problem. So, in the later of our work, more features will be extracted from URLs and more reasonable black and white list will be construct.

2.3 URL2Vec: URL Modeling with Character Embedding's for Fast and Accurate Phishing Website Detection

A deep learning-based approach to phishing detection is proposed. Specifically, websites' URLs and the characters in these URLs are mapped to documents and words, respectively, in the context of word2vec-based word embedding learning. Consequently, character embedding can be achieved from a corpus of URLs in an unsupervised manner. Furthermore, we combine character embedding with the structures of URLs to obtain the vector representations of the URLs. In particular, an URL is partitioned into the following five sections:

URL protocol, sub-domain name, domain name, domain suffix, and URL path. To identify the phishing URLs, existing classification algorithms can be used smoothly on the vector representations of the URLs, avoiding laborious work on designing effective features manually and empirically. For evaluations, we collect a large-scale dataset, i.e., 1 Million Phishing Detection Dataset (1M-PD), which has been released for public use.

Phishing detection is a challenging task. The traditionaltechnologies for phishing detection need to access webpages, while phishing webpages usually live for quite a short time. Therefore, it is difficult to collect valid webpages, and extract effective features related to their content. In addition, heuristic based anti-phishing approaches rely on laborious analysis of URLs and webpage content to extract hand-extracted features. However, The rules of designing such features, once mastered by phishers, are easily circumvented. Consequently, these techniques may not be able to deal with emerging phishing URLs, which makes anti-phishing researchers quite passive.

To deal with these challenges, anti-phishing researchers have developed quite a number of solutions. Detected malicious webpages by extracting the lexical and host-based features from URLs. However, it relies on the third-party tools such as WHOIS, DNS, etc., which may increase the time cost and be susceptible to the performance of these tools. The success of machine learning in recent years inspires researchers to apply

them on phishing detection tasks. Heuristic-based phishing methods have designed a large number of features from several aspects, such as URL, Hypertext Mark-up Language (HTML) codes, and webpage content. Ideally, automatic learning features that are effective for phishing detection are highly expected, which is the aim of our work.

The main contributions in this work are summarized as follows.

1. We propose a novel framework by making use of embedding representation of characters in URLs to detect phishing webpages. The character embedding achieved by the word2vec model does not rely on any external knowledge, hand-crafted information, or network load, overthrowing the conventional methods.

2. We devise a structure-wise strategy to obtain unified representation for the URLs based on the achieved character embedding. In particular, an URL is divided into five parts, including URL protocol, sub-domain name, domain name, domain suffix, URL path, etc., which contributes to the performance significantly.

3. We release a large-scale and real-world phishing detection dataset, i.e.,1M-PD consisting of 1 million URLs, which can hopefully be used to promote the research on phishing detection applications.

Our phishing detection system consists of three modules as shown in bellow Fig. More specifically, the character embedding learning module achieves the vector representation of characters in URLs. Given the character embedding's, the URL vector representation module obtains the vector representation of URLs by concatenating the average embedding's of the characters in the five parts of the URLs. Finally, the detectormodule trains machine learning algorithms on the vector representations of URLs to classify them into phishing ones or legitimate ones. Subsequently, we introduce the modules in each subsections respectively.

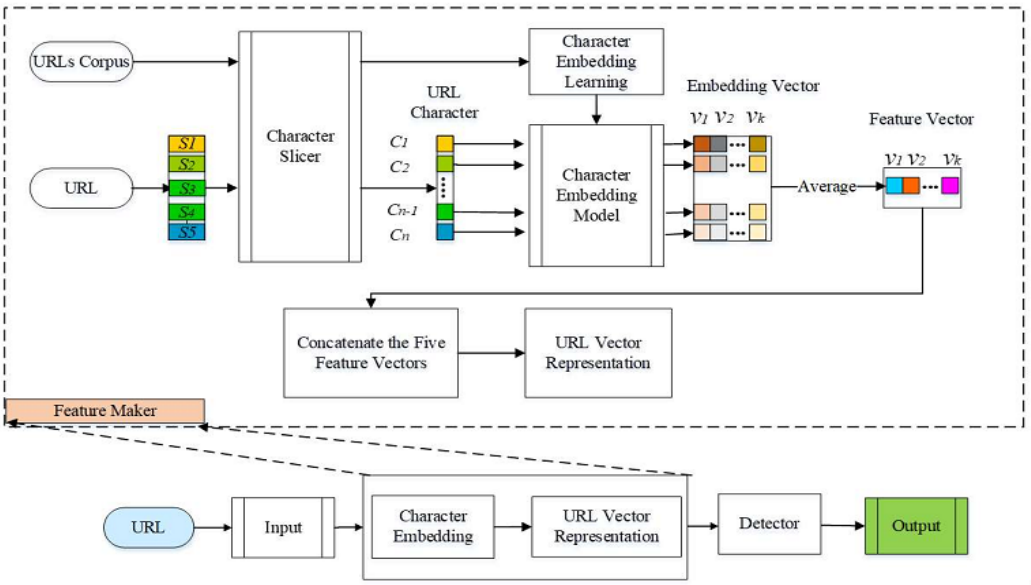


Fig 2.1 Overview of the Phishing framework

2.4 Detecting Phishing Websites and Targets Based on URLs and Webpage Links

This work, we propose to extract features from URLs and webpage links to detect phishing websites and their targets. In addition to the basic features of a given URL, such as length, suspicious characters, number of dots, a feature matrix is also constructed from these basic features of the links in the given URL's webpage. Furthermore, certain statistical features are extracted from each column of the feature matrix, such as mean, median, and variance. Lexical features are also extracted from the given URL, the links and content in its webpage, such as title and textual content. A number of machine learning models have been investigated for phishing detection, among which Deep Forest model shows competitive performance, achieving a true positive rate of 98.3% and a false alarm rate of 2.6%. In particular, we design an effective strategy based on search operator via search engines to find the phishing targets, which achieves an accuracy of 93.98%.

Phishing detection is a challenging task, which has attracted a lot of research attention from anti-phishing researchers to seek for effective solutions. The list-based anti-phishing approaches (blacklist or whitelist) store URLs in the database, which is used to match the stored URLs with the URLs entered by users in browsers. These approaches perform quickly but fail to detect newly created phishing URLs as they have not been included in the database. Heuristic-based methods usually extract textual features to detect phishing websites, which can detect newly created URLs. However, certain textual features extracted

from the textual content of webpages cannot be used to detect phishing websites in other languages.

Some researchers propose that similarity-based approaches should be used to compare with the similarity between the given suspicious legitimate webpages under attack, i.e., phishing targets, which should be known in advance. Lately, phishing targets can be detected automatically, unfortunately, its approach is very slow since it needs to obtain and analyze a large number of webpages to form a parasitic community.

A machine learning based method for phishing detection, which extracts statistical features and lexical features from URLs and the links inside the webpages. Given the URL representation in vector form, Deep Forest and a number of existing machine learning models, such as GBDT and XGBoost, can be applied seamlessly for phishing detection. The proposed approach works regardless of webpages in different languages. The method is efficient and effective, as shown in our experiments. We also propose an effective method based on search operator to detect phishing target, i.e., the legitimate websites under attack.

The main contributions of this paper are summarized as follows:

1. We propose to extract URL features and the statistical features of the links in the webpages to detect phishing webpages, which are effective for phishing detection.

2. We propose a method based on search operator to find the phishing targets of the detected phishing websites, allowing specified matching between query keywords and corresponding sections of the webpages of the phishing target candidates.

3. We investigate and evaluate quite a few machine learning based classification algorithms for phishing detection, among which Deep Forest achieves the highest performance.

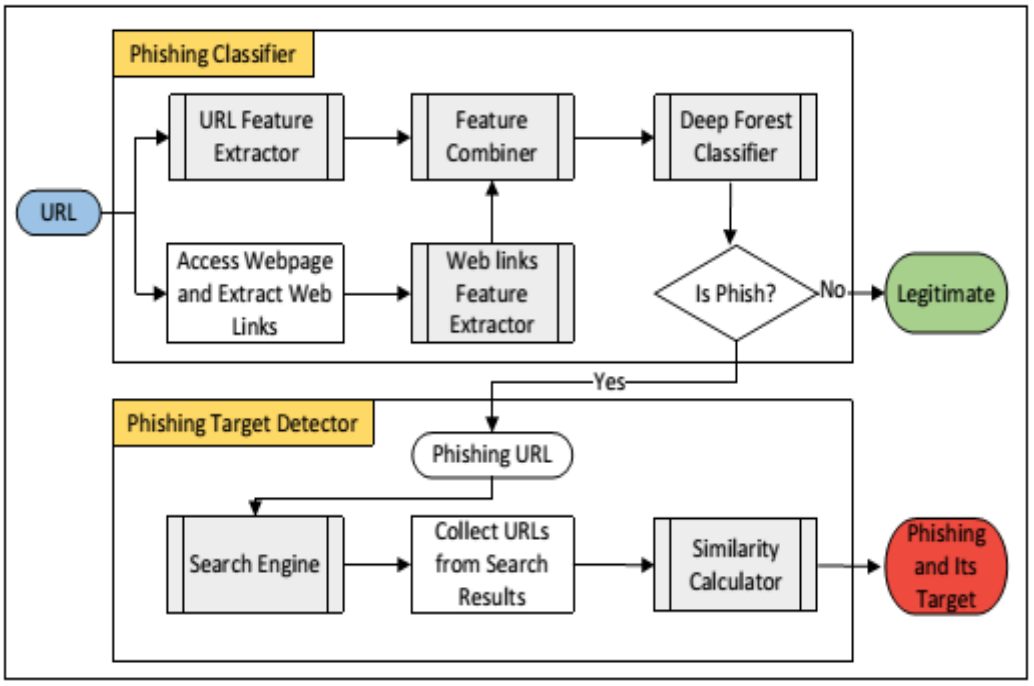


Fig 2.2 Overview of Phishing Websites and Targets Based on URLs and Webpage Links

To combine URL and webpage link features for phishing website detection. The achieved features can be used by various classification algorithms, among which DF shows competitive performance. In particular, we extract features from URLs and the links in the first-level webpages, and do not access the content of the second-level webpages. Therefore, the proposed approach performs quickly in practice and achieves a high accuracy. In addition, we proposed a search operator based method for phishing target detection, which has also achieved a relatively high accuracy.

2.5 An Effective Neural Network Phishing Detection Model Based on Optimal Feature Selection

As a common means to obtain user privacy information, phishing poses a big threat to people's daily network environment. The detection and prevention the threats of phishing websites are of importance. Due to the active learning ability from large-scale datasets, neural network is an important heuristic machine learning method in phishing websites detection and prevention. However, during the process of data training, some useless features may cause the machine learning method to over-fitting which will result in the training model not being able to precisely predict and detect the phishing websites.

Aiming at this problem, based on the optimal feature selection method, this paper proposes an effective neural network detection model (OFS-NN) to detect the pushing websites. Under this model, an optimal feature selection algorithm that adapts to the sensitive features of phishing URLs (Uniform Resource Locators) is firstly proposed. Based on the calculation of the effective value of each feature, this algorithm sets a threshold to eliminate some useless features and selects an optimal feature set suitable for detecting phishing websites. Then, the selected optimal feature set is trained by the neural network to construct an optimal classifier to classify and predict the pushing websites.

Phishing is a kind of cybercrime behavior in which attackers send malicious links through spam and social networks to lure users to obtain private information (user name, account, password, bank card information, etc.). Criminals use the stolen information to get money and other benefits. According to the report of Anti-Phishing Alliance of China, up to the fifth month of 2018, there are 421,070 phishing websites have been identified.

Generally, phishing websites mainly involve in three major industries: the payment transactions, the financial securities and the Ecommerce. Nowadays, phishing websites are widely flooded in the daily PC and mobile environments. Meanwhile, thenumber of phishing websites is growing rapidly, which poses a big threat to the people's online life. Hence, it is urgently needed effective techniques to prevent and detect phishing websites.

Aiming, this paper proposes OFSNN, an effective neural network phishing website detection model. Under this model, two algorithms, phishing websites feature extraction and optimal feature selection, are firstly employed to obtain the optimal feature set. Then, by feeding the selected optimal feature set, the neural network is trained to obtain the optimal classifier for phishing website prediction and selection.

1) By the feature extraction algorithm, four categories phishing website sensitive features, URL features, script features, security features and statistical features, are extracted from the input URLs. By doing this, we can ensure the high coverage of the feature extraction.

2) By calculating effective values of each feature of the dataset, the optimal feature selection algorithm selects an optimal feature set. During the process of optimal feature selection, much useless and small impact features are cut off. Since there is no interference

from useless features and small impact, the problem of over-fitting is resolved. At the same time, the optimal features selection algorithm brings a certain degree of improvement in performance.

3) OFS-NN takes the neural network as the classifier. By training the selected optimal features, the neural network can effectively predict and select the phishing websites. Due to the powerful learning and fitting abilities of the deep neural network, OFS-NN has the strong ability of generalization.

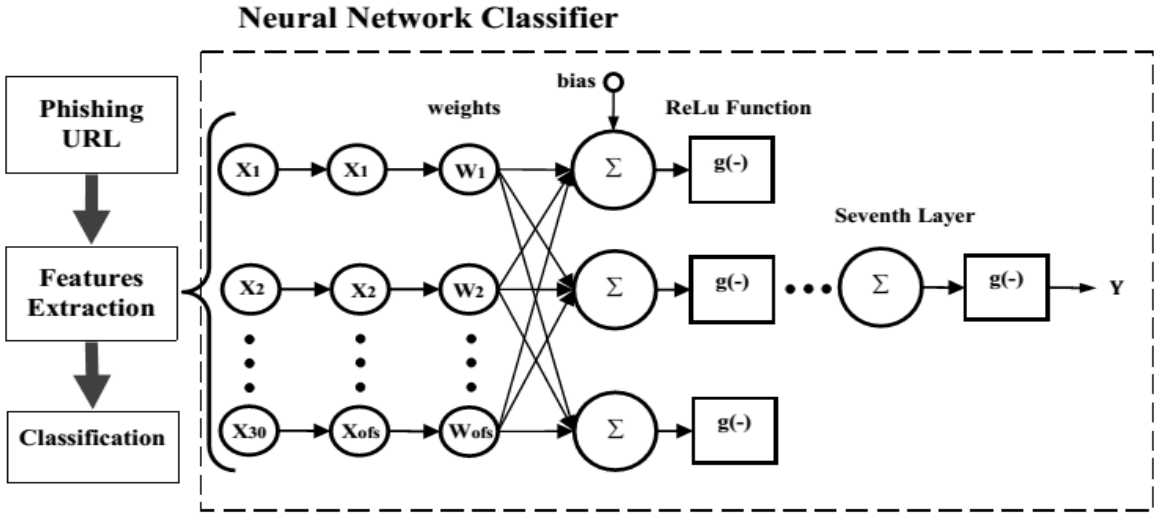


Fig 2.3 The detailed structure of the proposed phishing detection model OFS-NN.

Meanwhile, by repeating experimental analysis, we trained the best neural network structure suitable for phishing website detection. Since the OFS-NN model uses the neural network algorithm, it has strong ability of independent learning. The optimal feature selection algorithm can properly deal with problems of big number of phishing sensitivity features and the continuous change of features. This algorithm can reduce the over-fitting problem of the neural network classifier to some extent. Since the sensitivity feature phishing websites are continuous changing, in the future, it is necessary to collect more features to perform optimal feature selection.

2.6 Nudges for Privacy and Security: Understanding and Assisting Users' Choices Online

Advancements in information technology often task users with complex and consequential privacy and security decisions. A growing body of research has investigated individuals' choices in the presence of privacy and information security trade-offs, the decision-making hurdles affecting those choices, and ways to mitigate such hurdles.

This article provides a multi-disciplinary assessment of the literature pertaining to privacy and security decision making. It focuses on research on assisting individuals' privacy and security choices with soft paternalistic interventions that nudge users toward more beneficial choices. The article discusses potential benefits of those interventions, highlights their shortcomings, and identifies key ethical, design, and research challenges.

As they go about their online activities, individuals are faced with an increasing number of privacy and security decisions. Those decisions range from configuring visibility on social networking sites to deciding whether to download a mobile app based on the access to sensitive data it requests; from determining whether to trust a website to clicking on—or ignoring—a link in an email.

Such decisions arise in both personal and work contexts: workers who used to rely on workplace system administrators to manage enterprise security often find that they have to configure some security settings on their own, be it in the context of "Bring Your Own Device" workplaces or while interacting with an ever more diverse set of services where system administrators are not available to help.

First, we review research in relevant fields to gain insights into the impact of cognitive and behavioral biases on online security and privacy decision making. Then, we review interventions developed in various fields aimed at helping users make "better" online security and privacy decisions—that is, decisions that minimize adverse outcomes or are less likely to be regretted. We show how this work shares similarities with mechanisms developed to nudge people in a variety of other domains, such as health and retirement planning.

We broadly refer to these efforts as "nudging research," regardless of the originating field of study. We posit that all these efforts can be largely viewed as implementations of soft paternalistic concepts, whereby interventions are intended to gently guide users toward safer practices rather than imposing particular decisions. We suggest that prior work on the design of user interface technologies for security and privacy can be examined from a nudging perspective: every design decision potentially nudges users in one direction or another.

Our analysis of both the behavioral literature in general and the privacy and security literature in particular has highlighted a vast array of factors that may affect and impair end-users' privacy and security choices. As, the effects of various biases and heuristics on privacy and security choices have already started to be analyzed in a growing body of empirical research. The effect of other biases and heuristics—well

known to behavioral researchers, less so to privacy and security specialists—is currently only conjectural, but the hypotheses we present in that section may help drive future research efforts. Furthermore, examples of soft-paternalistic interventions in the field of privacy and security have started to arise both in research and in actual commercial products. As highlighted, we can find growing evidence both of tools aimed at making people reflect on their disclosure or security actions before they take them, and of tools and interface designs that nudge individuals toward more (or more open) disclosures.

In this survey article, our goal has been to document ongoing efforts in this area, discuss some of their limitations, and highlight their potential. We view this new emerging area as one that could lead to the design of a variety of tools and systems that effectively assist humans with online security and privacy decisions without imposing overly prescriptive models of what the "right" decisions might be. The studies we covered in this review have attempted to highlight the human processes that drive privacy and security behaviours, and how those processes can be (and are being) influenced by tools, interfaces, and choice architectures—even when they remain agnostic regarding the appropriateness of such interventions. As noted judging appropriateness is outside the scope of our review—it is, instead, the domain of society's and individuals' autonomous valuations.

In stating that, however, we have also argued that far from seeing nudging interventions as an invasion on individuals' otherwise pristine and untouched autonomy, we should realize that every design decision behind the construction of every online (e.g., software, online social networks, online blogs, mobile devices and applications, etc.) or offline (e.g., conference rooms, vehicles, food menus, etc.) system or tool we use has the potential to influence users' behaviours, regardless of whether the designer, or the user, is fully aware of those influences and their consequences. In simple terms, there is no such thing as a neutraldesign in privacy, security, or anywhere else. Therefore, we argue for conscious and cautious design of choice architectures and nudges that are inherent to any system, as well as the use of nudging to help users overcome cognitive and behavioral hurdles that may impact their privacy and security decisions.

2.7 Detection of Malicious URLs using Machine Learning Techniques

The primitive usage of URL (Uniform Resource Locator) is to use as a Web Address. However, some URLs can also be used to host unsolicited content that can potentially result in cyber attacks. These URLs are called malicious URLs. The inability of the end user system to detect and remove the malicious URLs can put the

legitimate user in vulnerable condition. Furthermore, usage of malicious URLs may lead to illegitimate access to the user data by adversary.

The main motive for malicious URL detection is that they provide an attack surface to the adversary. It is vital to counter these activities via some new methodology. In literature, there have been many filtering mechanisms to detect the malicious URLs. Some of them are Black-Listing, Heuristic Classification etc. These traditional mechanisms rely on keyword matching and URL syntax matching. Therefore, these conventional mechanisms cannot effectively deal with the ever evolving technologies and web access techniques. Furthermore, these approaches also fall short in detecting the modern URLs such as short URLs, dark web URLs.

In this work, we used a novel classification method to address the challenges faced by the traditional mechanisms in malicious URL detection. The proposed classification model is built on sophisticated machine learning methods that not only takes care about the syntactical nature of the URL, but also the semantic and lexical meaning of these dynamically changing URLs. The proposed approach is expected to outperform the existing techniques.

One of the other collaborative work has been initiated by the top tier Internet companies such as Google, Facebook along with many of the startup companies to build a single platform that works all together for one cause of preventing the naive users from the malicious URLs.

To counter these limitations, we propose a novel approach using sophisticatedmachine learning techniques that could be used as a common platform by the Internet users. In this paper, we propose a technique in order to detect the malicious URLs. Various feature sets for the URL detection have also been proposed that can be used with Support Vector Machines (SVM). The feature set is composed of the 18 features, such as token count, average path token, largest path, largest token, etc.

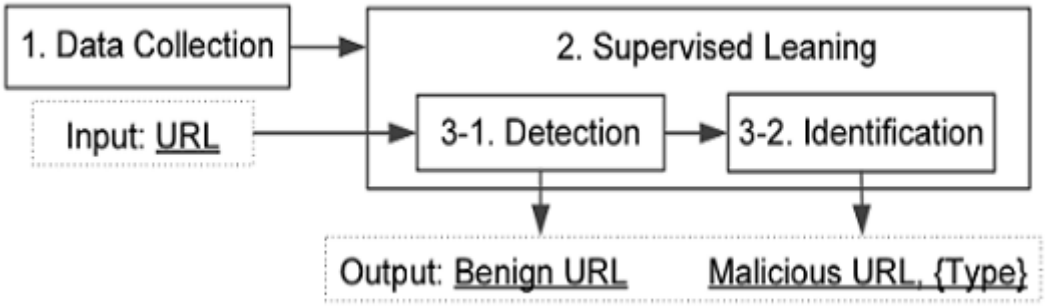


Fig. 2.4 A Framework of Detection of Malicious URLs

The comparison has been made on the various machine learning techniques The detailed view of the results of various techniques has been elaborated in stating that Convolution Neural Networks has shown good performance over the Support Vector Machine algorithm and Logistic Regression algorithm. Compared to the remaining Classification Techniques the Convolution Neural Networks has produced the precision of about 96% over the other two machine learning Techniques.

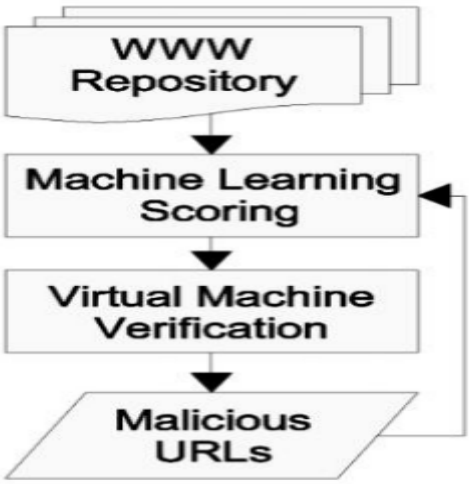


Fig. 2.5URL selection and Verification Workflow

Many methods are been proposed to fabricate the Classification Mechanism, Even though we are currently interested in just machine learning techniques, but out of all Convolutional Neural Networks(CNN) provided the better results this is because of the effective learning rate and quite suitable for the feature extraction. To weight the importance of each token, we used the term frequency and inverse document frequency. The term token is the chunk of the URLs. A token can be any part of the URL including the domain and the path.

Existing System

- Based on the whitelist, Kang proposes a method to detect the phishing websites. This method organizes user access to the site by detecting URL similarity. It deals with local and DNS spoofing attacks by comparing DNS enquiry results.
- Sharifi proposes a blacklist generator method for detecting phishing websites. This method determines whether it is a phishing website by matching the domain name of the website and Google‟s search results.

Disadvantages

- Han sandboxes live phishing toolkits by measuring the impact of blacklisting services on phishing websites at the beginning when these services are installed. In order to mitigate the threats of newly emerged phishing URLs.
- Lee proposes the PhishTrack framework for automatically updating the blacklist of phishing sites. The black and white lists consume smal resources on the underlying systems. However, it cannot properly deal with the newly emerged phishing attacks. In order to mitigate this shortage, as the work that is presented in this paper, the black and white lists are usually working in collaboration with other methods

Proposed System

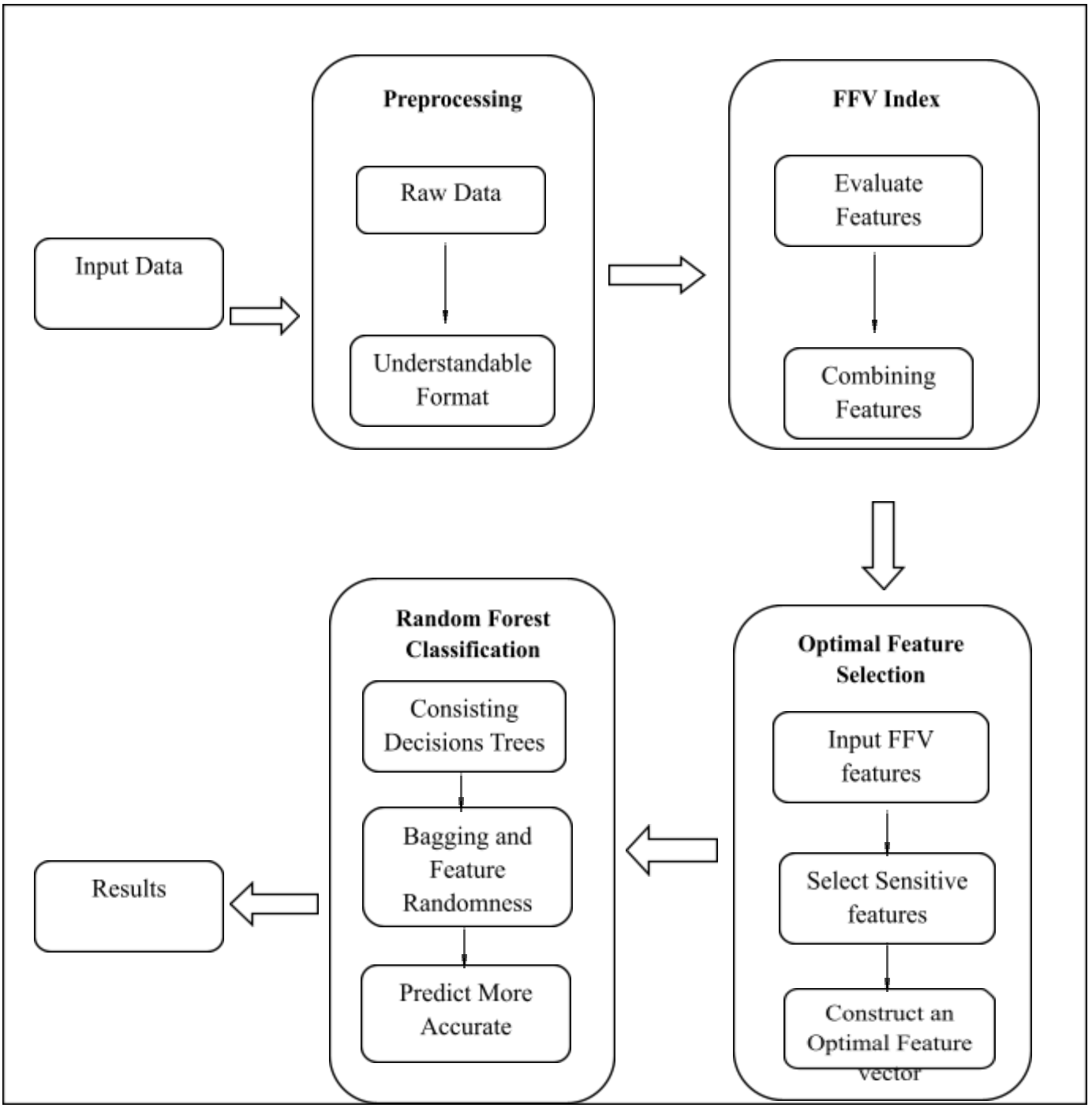
- Objective : To develop a model for effective and accurate phishing website prediction and to avoid some useless or small impact features and falling into the problem of over-fitting.
- FVV, Feature Validity Value is firstly introduced to evaluate the impact of sensitive features.
- Optimal feature selection algorithm, this algorithm calculates the FFV values of all features of the input URLs and their relevant websites at first. Then, a threshold is set to select sensitive features to construct an optimal feature vector.

- The selected optimal features are used to train the underlying neural network and, finally, an optimal classifier (Random Forest Classifier) is constructed to detect the phishing websites.

Advanatages

- This algorithm is able to reduce the time cost of the process of phishing websites detection.
- Through this algorithm, many useless and small influence features are pruned.
- Due to no disturbance from these redundant features, the over-fitting problem of the underlying neural network is alleviated.

System Architecture



SYSTEM SPECIFICATION

Software Used:

Operating System	:	Windows 7 / 8/ 10
Language	:	Python
IDE	:	Anaconda, Notebook

Hardware Used:

Processor	:	Intel core i3
Ram	:	2 GB
Hard Disk	:	120 GB

Module Description

- Preprocessing
- FFV index Calculation
- Optimal Feature Selection
- Training the data
- Classification of data

Preprocessing

- Data preprocessing is a data mining technique that involves transforming raw data into an understandable format.
- Data can have missing values for a number of reasons such as observations that were not recorded and data corruption.
- Handling missing data is important as many machine learning algorithms do not support data with missing values.

FFV index Calculation

- Feature Validity Value is to evaluate the impact of sensitive features on phishing websites detection.
- In order to better evaluate the impact of a selected sensitive feature on detecting phishing attacks, this paper presents the FVV index.
- The new FVV is defined by combining the positive and negative features of URLs.

Optimal Feature Selection

- Feature extraction is the second class of methods for dimension reduction. This function is useful for reducing the dimensionality of high-dimensional data. (ie you get less columns).
- This module uses the FFV values of all features of the input URLs and their relevant websites at first.
- Then, a threshold is set to select sensitive features to construct an optimal feature vector.
- Through this algorithm, many useless and small influence features are pruned.

Training the data

- The selected optimal features are used to train the underlying neural network.
- Neural network is used to train the data, which is resulted as optimal features.
- neural network model is composed of 3 layers: the input, the hidden and the output layers. Algorithms utilized by neural network generally incorporate two phases: the forward propagation and the backward propagation.
- The forward propagation starts to work when it receives a positive propagation signal.
- The backward propagation is invoked when the model detects the occurrence of evident deviation between the calculation result of the output layer and the actual value.

Classification of data

- Random Forest Classifier is used to classify the data of phishing website detection.
- The random forest is a classification algorithm consisting of many decisions trees.
- It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

SOFTWARE AND SYSTEM SPECIFICATION

4.1 PyCharm

PyCharm is the most popular IDE used for Python scripting language. This chapter will give you an introduction to PyCharm and explains its features. PyCharm offers some of the best features to its users and developers in the following aspects:

· Code completion and inspection

· Advanced debugging

· Support for web programming and frameworks such as Django and Flask

Features of PyCharm

Besides, a developer will find PyCharm comfortable to work with because of the features mentioned below:

Code Completion- PyCharm enables smoother code completion whether it is for built in or for an external package.

SQLAlchemy - as Debugger You can set a breakpoint, pause in the debugger and can see the SQL representation of the user expression for SQL Language code.

Git Visualization in Editor - When coding in Python, queries are normal for a developer. You can check the last commit easily in PyCharm as it has the blue sections that can define the difference between the last commit and the current one.

Code Coverage in Editor - You can run .py files outside PyCharm Editor as well marking it as code coverage details elsewhere in the project tree, in the summary section etc.

Package Management - All the installed packages are displayed with proper visual representation. This includes list of installed packages and the ability to search and add new packages.

Local History - Local History is always keeping track of the changes in a way that complements like Git. Local history in PyCharm gives complete details of what is needed to rollback and what is to be added.

4.1.1 Intelligent Coding Assistance

PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactorings and rich navigation capabilities.

Intelligent Code Editor

PyCharm's smart code editor provides first-class support for Python, JavaScript, CoffeeScript, TypeScript, CSS, popular template languages and more. Take advantage of language-aware code completion, error detection, and on-the-fly code fixes!

Smart Code Navigation

Use smart search to jump to any class, file or symbol, or even any IDE action or tool window. It only takes one click to switch to the declaration, super method, test, usages, implementation, and more.

Fast and Safe Refactorings

Refactor your code the intelligent way, with safe Rename and Delete, Extract Method, Introduce Variable, Inline Variable or Method, and other refactorings. Language and framework-specific refactorings help you perform project-wide changes.

Rename and Move

The Rename and Move refactorings work for files, functions, constants, classes, properties, methods, parameters, and local and global variables.

Extract Refactorings

Use Extract Variable/Field/ Constant/Parameter and Inline Local for improving the code structure within a method.

Extract Method

Use Extract Method to break up longer methods, Extract Superclass, Push Up, Pull Down to move the methods and classes.

4.1.2Web Development

In addition to Python, PyCharm provides first-class support for various Python web development frameworks, specific template languages, JavaScript, CoffeeScript, TypeScript, HTML/CSS, AngularJS, Node.js, and more.

Python Web frameworks

PyCharm offers great framework-specific support for modern web development frameworks such as Django, Flask, Google App Engine, Pyramid, and web2py, including Django templates debugger, manage.py and appcfg.py tools, special autocompletion and navigation, just to name a few.

JavaScript & HTML

PyCharm provides first-class support for JavaScript, CoffeeScript, TypeScript, HTML and CSS, as well as their modern successors. The JavaScript debugger is included in PyCharm and is integrated with the Django server run configuration.

Live Edit

Live Editing Preview lets you open a page in the editor and the browser and see the changes being made in code instantly in the browser. PyCharm auto-saves your changes, and the browser smartly updates the page on the fly, showing your edits.

4.1.3Professional Features

PyCharm Professional is a paid version of PyCharm with more out-of-the-box features and integrations. In this section, you'll mainly be presented with overviews of its main features and links to the official documentation, where each feature is discussed in detail. Remember that none of the following features is available in the Community edition.

Django Support

PyCharm has extensive support for Django, one of the most popular and beloved Python web frameworks. To make sure that it's enabled, do the following:

1. Open *Preferences* on Mac or *Settings* on Windows or Linux.
2. Choose *Languages and Frameworks*.
3. Choose *Django*.
4. Check the checkbox *Enable Django support*.

5. Apply changes.

Now that you've enabled Django support, your Django development journey will be a lot easier in PyCharm:

- When creating a project, you'll have a dedicated Django project type. This means that, when you choose this type, you'll have all the necessary files and settings. This is the equivalent of using `django-admin startprojectmysite`.
- You can run `manage.py` commands directly inside PyCharm.
- Django templates are supported, including:
    - Syntax and error highlighting
    - Code completion
    - Navigation
    - Completion for block names
    - Completion for custom tags and filters
    - Quick documentation for tags and filters
    - Capability to debug them
- Code completion in all other Django parts such as views, URLs and models, and code insight support for Django ORM.
- Model dependency diagrams for Django models.

Database Support

Modern database development is a complex task with many supporting systems and workflows. That's why JetBrains, the company behind PyCharm, developed a standalone IDE called DataGrip for that. It's a separate product from PyCharm with a separate license.

Luckily, PyCharm supports all the features that are available in DataGrip through a plugin called *Database tools and SQL*, which is enabled by default. With the help of it, you can query, create and manage databases whether they're working locally, on a server, or in the cloud. The plugin supports MySQL, PostgreSQL, Microsoft SQL Server, SQLite, MariaDB, Oracle, Apache Cassandra, and others. For more information on what you can do with this plugin, check out the comprehensive documentation on the database support.

4.1.4 Conclusion

PyCharm is one of best, if not the best, full-featured, dedicated, and versatile IDEs for Python development. It offers a ton of benefits, saving you a lot of time by helping you with routine tasks. Now you know how to be productive with it!

In this article, you learned about a lot, including:

- Installing PyCharm
- Writing code in PyCharm
- Running your code in PyCharm
- Debugging and testing your code in PyCharm
- Editing an existing project in PyCharm
- Searching and navigating in PyCharm
- Using Version Control in PyCharm
- Using Plugins and External Tools in PyCharm
- Using PyCharm Professional features, such as Django support and Scientific mode

If there's anything you'd like to ask or share, please reach out in the comments below. There's also a lot more information at the PyCharm website for you to explore.

4.2 Anaconda (Python distribution)

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS.

4.2.1 Overview

Anaconda distribution comes with more than 1,500 packages as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example, Google Tensorflow, can find that it stops working

having used pip to install a different package that requires a different version of the dependent numpy library than the one used by Tensorflow. In some cases, the package may appear to work but produce different results in detail.

In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g. the user may wish to have Tensorflow version 2,0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open source packages can be individually installed from the Anaconda repository[8], Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the conda install command. Anaconda Inc compiles and builds all the packages in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed.

Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud,[9] PyPI or other repositories.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

4.2.2 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

JupyterLab

Jupyter Notebook

QtConsole

Spyder

Glue

Orange

RStudio

Visual Studio Code

Conda

Main article: Conda (package manager)

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

4.2.3 Anaconda Cloud

Anaconda Cloud is a package management service by Anaconda where you can find, access, store and share public and private notebooks, environments, and conda and PyPI packages.[20] Cloud hosts useful Python packages, notebooks and environments for a wide variety of applications. You do not need to log in or to have a Cloud account, to search for public packages, download and install them.

You can build new packages using the Anaconda Client command line interface (CLI), then manually or automatically upload the packages to Cloud.

4.3 Jupyter Notebook

This tutorial explains how to install, run, and use Jupyter Notebooks for data science, including tips, best practices, and examples.

As a web application in which you can create and share documents that contain live code, equations, visualizations as well as text, the Jupyter Notebook is one of the ideal tools to help you to gain the data science skills you need.

This tutorial will cover the following topics:

- A basic overview of the Jupyter Notebook App and its components,
- The history of Jupyter Project to show how it's connected to IPython,
- An overview of the three most popular ways to run your notebooks: with the help of a Python distribution, with pip or in a Docker container,
- A practical introduction to the components that were covered in the first section, complete with examples of Pandas DataFrames, an explanation on how to make your

notebook documents magical, and answers to frequently asked questions, such as "How to toggle between Python 2 and 3?", and

- The best practices and tips that will help you to make your notebook an added value to any data science project!

(To practice pandas dataframes in Python, try this course on Pandas foundations.)

What Is A Jupyter Notebook?

In this case, "notebook" or "notebook documents" denote documents that contain both code and rich text elements, such as figures, links, equations, ... Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description, and its results, as well as, they can be executed perform the data analysis in real time.

The Jupyter Notebook App produces these documents.

We'll talk about this in a bit.

For now, you should know that "Jupyter" is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports many other languages.

And there you have it: the Jupyter Notebook.

As you just saw, the main components of the whole environment are, on the one hand, the notebooks themselves and the application. On the other hand, you also have a notebook kernel and a notebook dashboard.

Let's look at these components in more detail.

What Is The Jupyter Notebook App?

As a server-client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser. The application can be executed on a PC without Internet access, or it can be installed on a remote server, where you can access it through the Internet.

Its two main components are the kernels and a dashboard.

A kernel is a program that runs and introspects the user's code. The Jupyter Notebook App has a kernel for Python code, but there are also kernels available for other programming languages.

The dashboard of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels: you can which ones are running and shut them down if necessary.

The History of IPython and Jupyter Notebooks

To fully understand what the Jupyter Notebook is and what functionality it has to offer you need to know how it originated.

Let's back up briefly to the late 1980s. Guido Van Rossum begins to work on Python at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Fast forward two years: the IPython team had kept on working, and in 2007, they formulated another attempt at implementing a notebook-type system. By October 2010, there was a prototype of a web notebook, and in the summer of 2011, this prototype was incorporated, and it was released with 0.12 on December 21, 2011. In subsequent years, the team got awards, such as the Advancement of Free Software for Fernando Pérez on 23 of March 2013 and the Jolt Productivity Award, and funding from the Alfred P. Sloan Foundations, among others.

Lastly, in 2014, Project Jupyter started as a spin-off project from IPython. IPython is now the name of the Python backend, which is also known as the kernel. Recently, the next generation of Jupyter Notebooks has been introduced to the community. It's called JupyterLab.

After all this, you might wonder where this idea of notebooks originated or how it came about to the creators.

A brief research into the history of these notebooks learns that Fernando Pérez and Robert Kern were working on a notebook just at the same time as the Sage notebook was a work in progress. Since the layout of the Sage notebook was based on the layout of Google notebooks, you can also conclude that also Google used to have a notebook feature around that time.

For what concerns the idea of the notebook, it seems that Fernando Pérez, as well as William Stein, one of the creators of the Sage notebook, have confirmed that they were avid users of the Mathematica notebooks and Maple worksheets. The Mathematica notebooks were created as a front end or GUI in 1988 by Theodore Gray.

The concept of a notebook, which contains ordinary text and calculation and/or graphics, was definitely not new.

Also, the developers had close contact with one another and this, together with other failed attempts at GUIs for IPython and the use of "AJAX" = web applications, which didn't require users to refresh the whole page every time you do something, were two other motivations for the team of William Stein to start developing the Sage notebooks.

Codings

```
import pandas as pd

legitimate_urls = pd.read_csv("dataset/legitimate-urls.csv")

phishing_urls = pd.read_csv("dataset/phishing-urls.csv")

legitimate_urls.head(10)
```

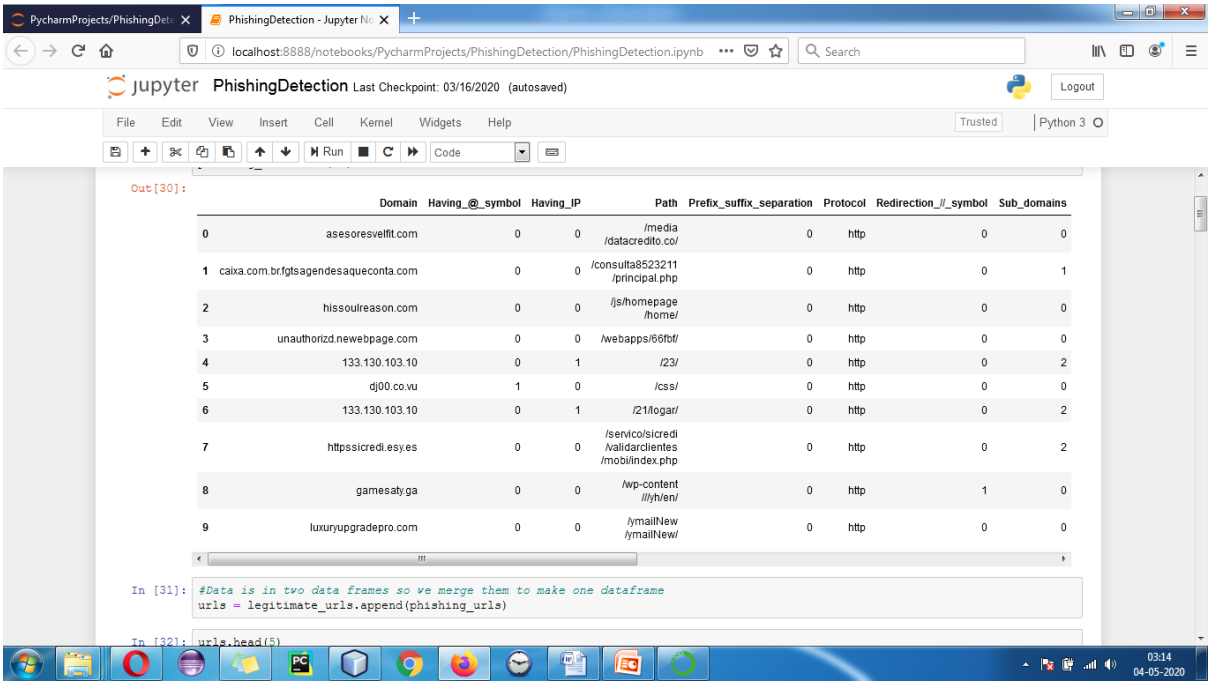
```
phishing_urls.head(10)

#Data is in two data frames so we merge them to make one dataframe

urls = legitimate_urls.append(phishing_urls)

urls.head(5)

urls.shape[1]

urls.columns

#Removing Unnecessary columns

urls = urls.drop(urls.columns[[0,3,5]],axis=1)

print('Number of missing values in dataset')

print(urls.isnull().sum().sum())

# shuffling the rows in the dataset so that when splitting the train and test set are equally distributed

urls = urls.sample(frac=1).reset_index(drop=True)

#Removing class variable from the dataset

urls_without_labels = urls.drop('label',axis=1)

urls_without_labels.columns

labels = urls['label']

#splitting the data into train data and test data

#Dividing the data in the ratio of 70:30 [train_data:test_data]

from sklearn.model_selection import train_test_split

data_train, data_test, labels_train, labels_test = train_test_split(urls_without_labels, labels, test_size=0.30, random_state=110)

print(len(data_train),len(data_test),len(labels_train),len(labels_test))

#initially checking the split of labels_train data

labels_train.value_counts()
```

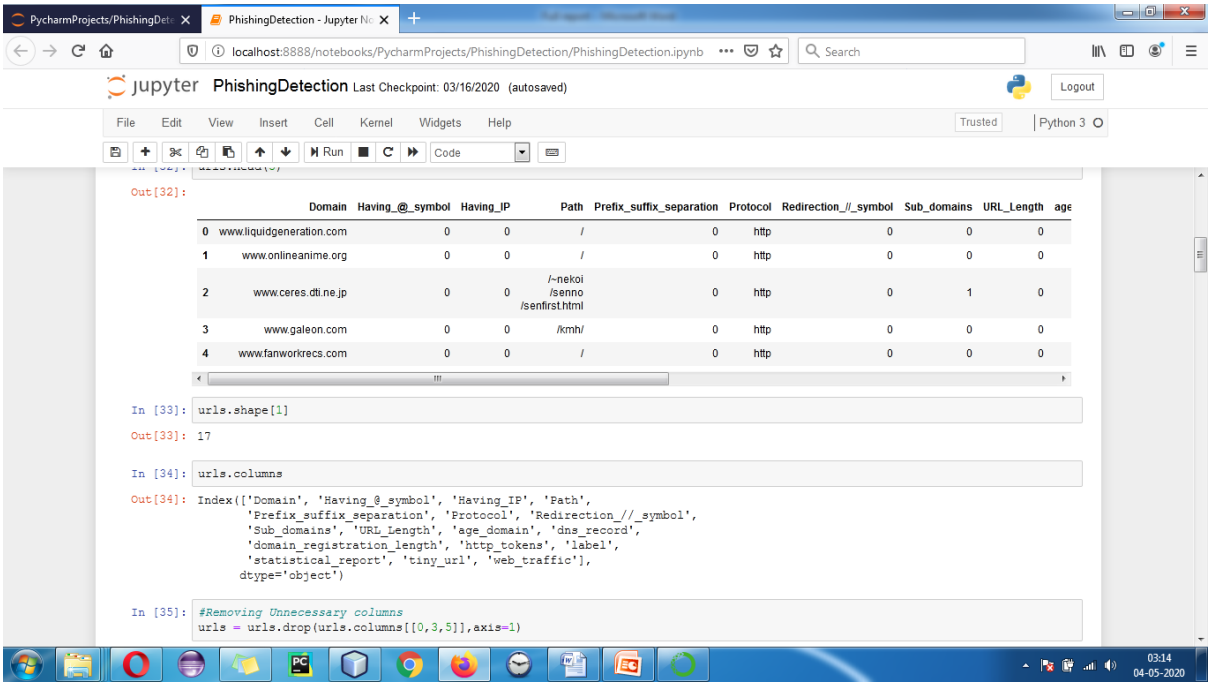
```
#checking the split for labels_test data

labels_test.value_counts()

from sklearn.ensemble import RandomForestClassifier

custom_random_forest_classifier = RandomForestClassifier(n_estimators=500,
max_depth=20, max_leaf_nodes=10000)

custom_random_forest_classifier.fit(data_train,labels_train)

custom_classifier_prediction_label = custom_random_forest_classifier.predict(data_test)

import matplotlib.pyplot as plt

import numpy as np


#feature_importances_ : array of shape = [n_features] ------ The feature importances (the
higher, the more important the feature).


#feature_importances_  -- This method returns the quantified relative importance in the order
the features were fed to the algorithm


importances = custom_random_forest_classifier.feature_importances_


#std = np.std([tree.feature_importances_ for tree in
custom_random_forest_classifier.estimators_],axis=0)   #[[[estimators_ :explaination ---  list
of DecisionTreeClassifier ----- (The collection of fitted sub-estimators.)]]]


#To make the plot pretty, we'll instead sort the features from most to least important.

indices = np.argsort(importances)[::-1]

print(f"indices of columns : {indices}")
```

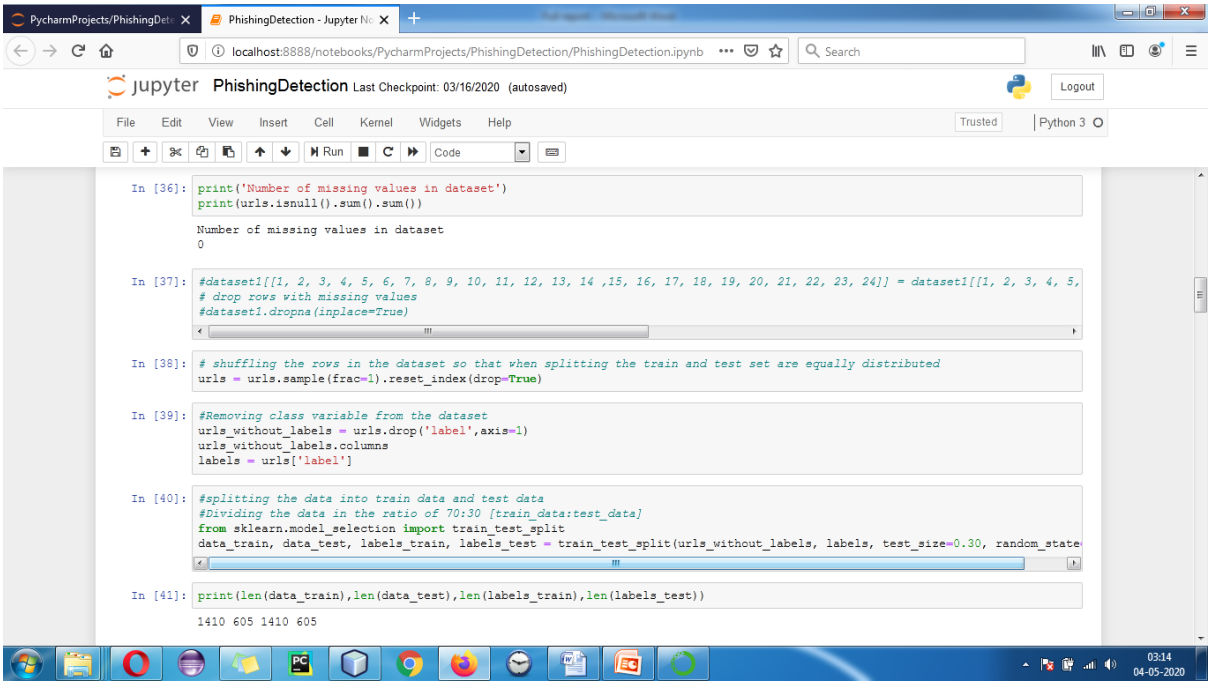
```
# Print the feature ranking

print("\n ***Feature ranking: *** \n")

print("Feature name : Importance")


for f in range(data_train.shape[1]):

    print(f"{f+1} {data_train.columns[indices[f]]}   :  {importances[indices[f]]} \n")


print("**** The blue bars are the feature importances of the randomforest classifier, along with their inter-trees variability*****")


# Plot the feature importances of the forest

plt.figure()

plt.title("Feature importances")

plt.bar(range(data_train.shape[1]), importances[indices],

       color="b", align="center")

#yerr=std[indices] -- this is another parameter that can be included if std is calculated above

#and also it gives error bar that's the reason we calculate std above. but here we are not making it plot.


plt.xticks(range(data_train.shape[1]), data_train.columns[indices])

plt.xlim([-1, data_train.shape[1]])


plt.rcParams['figure.figsize'] = (450,400)  #this will increase the size of the plot

plt.show()

from sklearn.tree import DecisionTreeClassifier
```

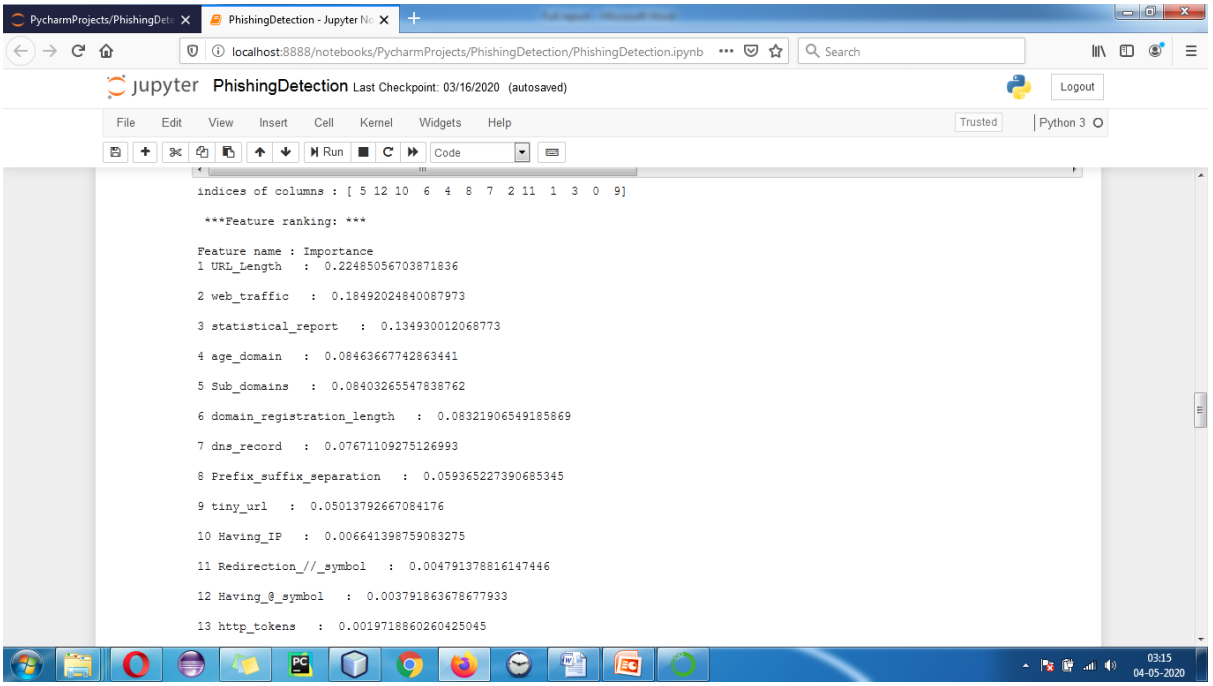
```
model = DecisionTreeClassifier()
model.fit(data_train,labels_train)
pred_label = model.predict(data_test)
print(pred_label),print(list(labels_test))
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(labels_test,pred_label)
print(cm)
accuracy_score(labels_test,pred_label)
#Creating the model and fitting the data into the model
from sklearn.ensemble import RandomForestClassifier
random_forest_classifier = RandomForestClassifier()
random_forest_classifier.fit(data_train,labels_train)
#Predicting the result for test data
prediction_label = random_forest_classifier.predict(data_test)
print(prediction_label),print(list(labels_test))
from sklearn.metrics import confusion_matrix,accuracy_score
cpnfusionMatrix = confusion_matrix(labels_test,prediction_label)
print(cpnfusionMatrix)
accuracy_score(labels_test,prediction_label)
```

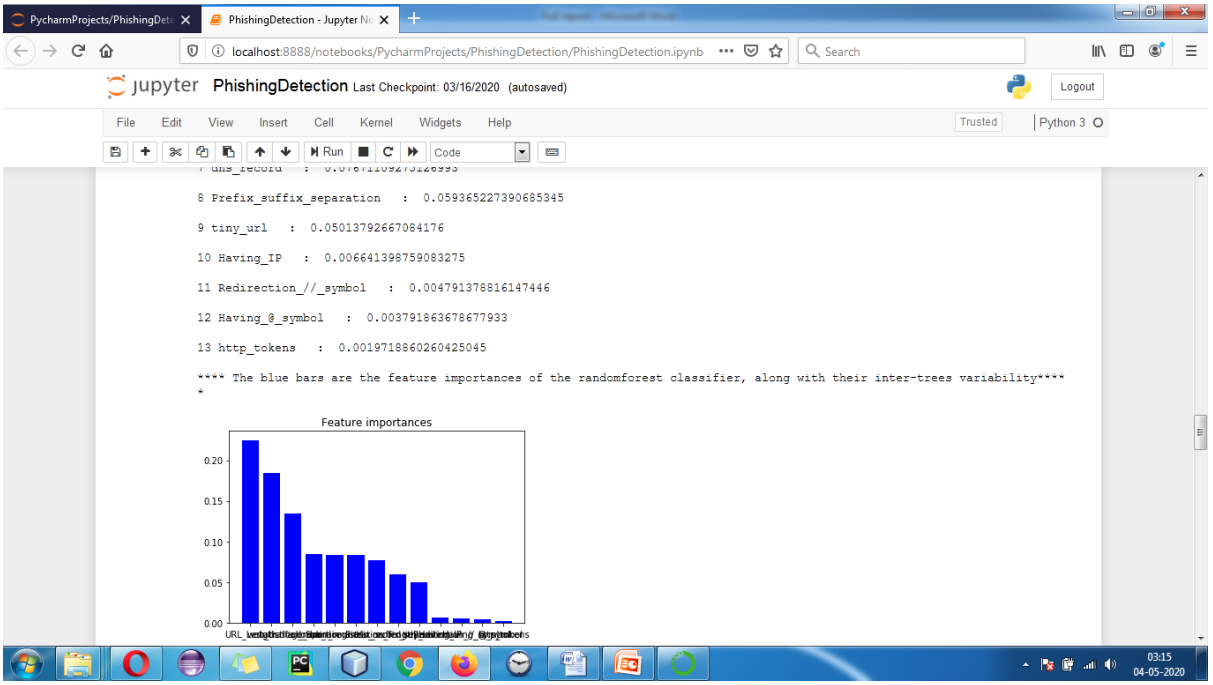
Screenshot

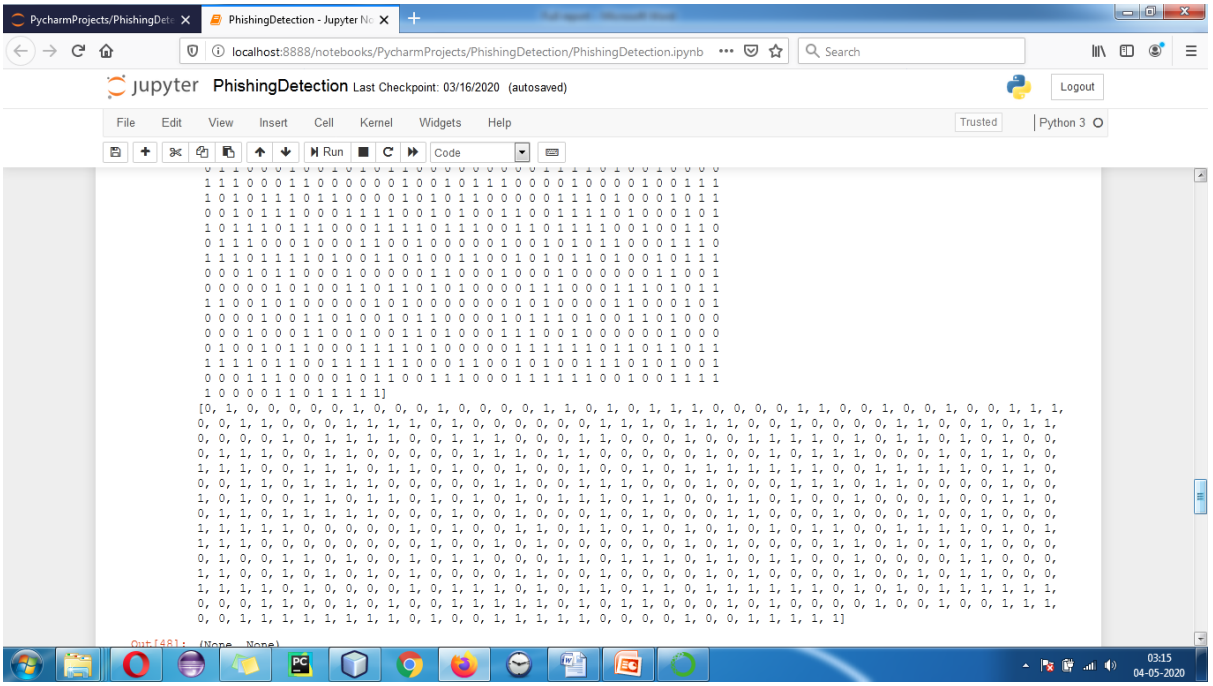

PycharmProjects/PhishingDet
PhishingDetection - Jupyter No
localhost:8888/notebooks/PycharmProjects/PhishingDetection/PhishingDetection.ipynb
Search
jupyter PhishingDetection Last Checkpoint: 03/16/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Out[30]:
Domain Having_@_symbol Having_IP Path Prefix_suffix_separation Protocol Redirection_//_symbol Sub_domains
0 asesoresvelfit.com 0 0 /media /datacredito.co/ 0 http 0 0
1 caixa.com.br.fgtsagendesaqueconta.com 0 0 /consulta8523211 /principal.php 0 http 0 1
2 hissoulreason.com 0 0 /js/homepage /home/ 0 http 0 0
3 unauthorizd.newebpage.com 0 0 /webapps/66fbf/ 0 http 0 0
4 133.130.103.10 0 1 /23/ 0 http 0 2
5 dj00.co.vu 1 0 /css/ 0 http 0 0
6 133.130.103.10 0 1 /21/logar/ 0 http 0 2
7 httpssicredi.esy.es 0 0 /servico/sicredi /validarclientes /mobi/index.php 0 http 0 2
8 gamesaty.ga 0 0 /wp-content ///yh/en/ 0 http 1 0
9 luxuryupgradepro.com 0 0 /ymailNew /ymailNew/ 0 http 0 0
In [31]:
#Data is in two data frames so we merge them to make one dataframe
urls = legitimate_urls.append(phishing_urls)
In [32]:
urls.head(5)
03:14
04-05-2020

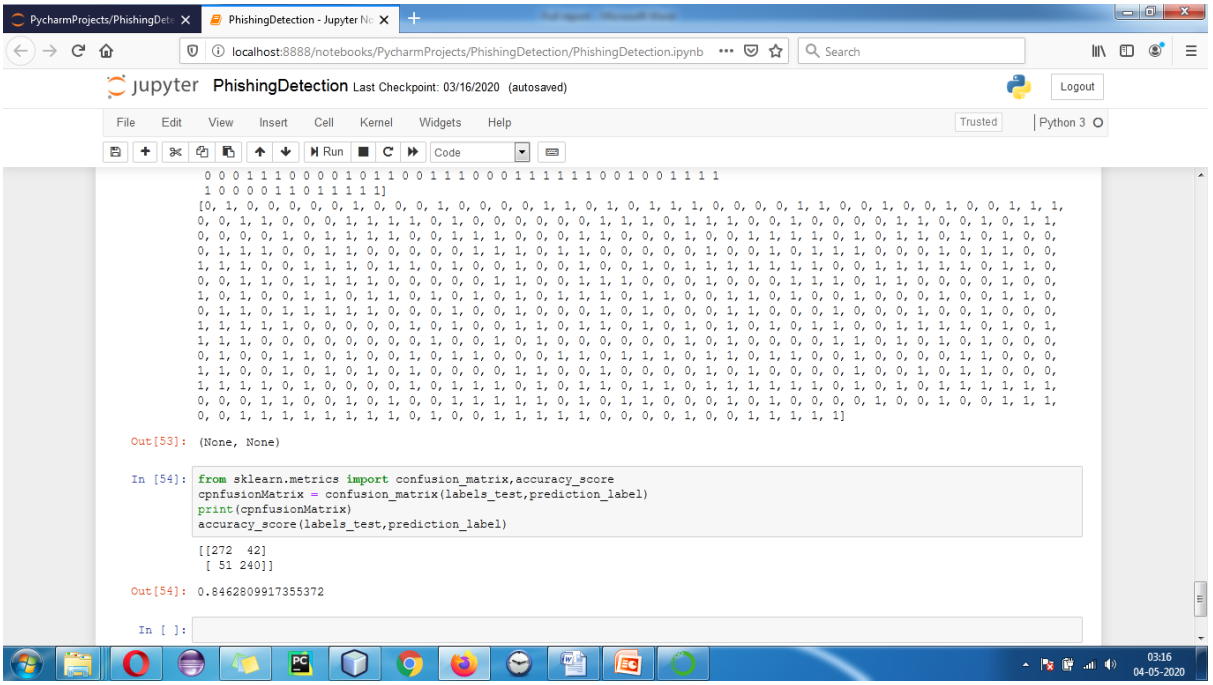

PycharmProjects/PhishingDet
PhishingDetection - Jupyter No
localhost:8888/notebooks/PycharmProjects/PhishingDetection/PhishingDetection.ipynb
Search
jupyter PhishingDetection Last Checkpoint: 03/16/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Out[32]:
Domain Having_@_symbol Having_IP Path Prefix_suffix_separation Protocol Redirection_//_symbol Sub_domains URL_Length age
0 www.liquidgeneration.com 0 0 / 0 http 0 0 0
1 www.onlineanime.org 0 0 / 0 http 0 0 0
2 www.ceres.dti.ne.jp 0 0 /~nekoi /senno /senfirst.html 0 http 0 1 0
3 www.galeon.com 0 0 /kmh/ 0 http 0 0 0
4 www.fanworkrecs.com 0 0 / 0 http 0 0 0
In [33]:
urls.shape[1]
Out[33]:
17
In [34]:
urls.columns
Out[34]:
Index(['Domain', 'Having_@_symbol', 'Having_IP', 'Path',
'Prefix_suffix_separation', 'Protocol', 'Redirection_//_symbol',
'Sub_domains', 'URL_Length', 'age_domain', 'dns_record',
'domain_registration_length', 'http_tokens', 'label',
'statistical_report', 'tiny_url', 'web_traffic'],
dtype='object')
In [35]:
#Removing Unnecessary columns
urls = urls.drop(urls.columns[[0,3,5]],axis=1)
03:14
04-05-2020


PycharmProjects/PhishingDet
PhishingDetection - Jupyter No
localhost:8888/notebooks/PycharmProjects/PhishingDetection/PhishingDetection.ipynb
Search
jupyter PhishingDetection Last Checkpoint: 03/16/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [36]:
print('Number of missing values in dataset')
print(urls.isnull().sum().sum())
Number of missing values in dataset
0
In [37]:
#dataset1[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 ,15, 16, 17, 18, 19, 20, 21, 22, 23, 24]] = dataset1[[1, 2, 3, 4, 5,
drop rows with missing values
#dataset1.dropna(inplace=True)
In [38]:
# shuffling the rows in the dataset so that when splitting the train and test set are equally distributed
urls = urls.sample(frac=1).reset_index(drop=True)
In [39]:
#Removing class variable from the dataset
urls_without_labels = urls.drop('label',axis=1)
urls_without_labels.columns
labels = urls['label']
In [40]:
#splitting the data into train data and test data
#Dividing the data in the ratio of 70:30 [train_data:test_data]
from sklearn.model_selection import train_test_split
data_train, data_test, labels_train, labels_test = train_test_split(urls_without_labels, labels, test_size=0.30, random_state
In [41]:
print(len(data_train),len(data_test),len(labels_train),len(labels_test))
1410 605 1410 605
03:14
04-05-2020


PycharmProjects/PhishingDet
PhishingDetection - Jupyter No
localhost:8888/notebooks/PycharmProjects/PhishingDetection/PhishingDetection.ipynb
Search
jupyter PhishingDetection Last Checkpoint: 03/16/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
indices of columns : [5 12 10 6 4 8 7 2 11 1 3 0 9]
***Feature ranking: ***
Feature name : Importance
1 URL_Length : 0.22485056703871836
2 web_traffic : 0.18492024840087973
3 statistical_report : 0.134930012068773
4 age_domain : 0.08463667742863441
5 Sub_domains : 0.08403265547838762
6 domain_registration_length : 0.08321906549185869
7 dns_record : 0.07671109275126993
8 Prefix_suffix_separation : 0.059365227390685345
9 tiny_url : 0.05013792667084176
10 Having_IP : 0.006641398759083275
11 Redirection_//_symbol : 0.004791378816147446
12 Having_@_symbol : 0.0037918636786779 33
13 http_tokens : 0.0019718860260425045
03:15
04-05-2020


PycharmProjects/PhishingDet
PhishingDetection - Jupyter No
localhost:8888/notebooks/PycharmProjects/PhishingDetection/PhishingDetection.ipynb
Search
jupyter PhishingDetection Last Checkpoint: 03/16/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
8 Prefix_suffix_separation : 0.059365227390685345
9 tiny_url : 0.05013792667084176
10 Having_IP : 0.0066413987590832 75
11 Redirection_//_symbol : 0.004791378816147446
12 Having_@_symbol : 0.003791863678677933
13 http_tokens : 0.0019718860260425045
**** The blue bars are the feature importances of the randomforest classifier, along with their inter-trees variability****
*
Feature importances
0.20
0.15
0.10
0.05
0.00
03:15
04-05-2020


PycharmProjects/PhishingDet
PhishingDetection - Jupyter No
localhost:8888/notebooks/PycharmProjects/PhishingDetection/PhishingDetection.ipynb
Search
jupyter PhishingDetection Last Checkpoint: 03/16/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
[0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1,
0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0,
0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0,
1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1]
Out[48]: (None, None)
03:15
04-05-2020


PycharmProjects/PhishingDet
PhishingDetection - Jupyter No
localhost:8888/notebooks/PycharmProjects/PhishingDetection/PhishingDetection.ipynb
Search
jupyter PhishingDetection Last Checkpoint: 03/16/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
0 0 0 1 1 1 0 0 0 0 1 0 1 1 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1
1 0 0 0 0 1 1 0 1 1 1 1 1]
[0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1,
0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0,
0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1]
Out[53]: (None, None)
In [54]:
from sklearn.metrics import confusion_matrix,accuracy_score
cpnfusionMatrix = confusion_matrix(labels_test,prediction_label)
print(cpnfusionMatrix)
accuracy_score(labels_test,prediction_label)
[[272 42]
[51 240]]
Out[54]: 0.8462809917355372
In [ ]:
03:16
04-05-2020

Conclusion

In this paper, we proposed a feasible phishing website detection model OFS-NN, which has proved to be highly accurate and has low false negative rate and low false positive rate. In addition, the optimal feature selection algorithm is combined with the neural network algorithm to select the optimal feature value set for the input of the neural network. OFS-NN model uses the neural network algorithm, it has strong ability of independent learning. The optimal feature selection algorithm can properly deal with problems of big number of phishing sensitivity features and the continuous change of features. This algorithm can reduce the over-fitting problem of the neural network classifier to some extent.

Future Work

In the future, it is necessary to collect more features to perform optimal feature selection. And choose better accuracy algorithm to beat the random forest algorithm.

References

[1] APAC. Fishing website processing Bulletin in December 2018. [Online]

Available: http://www.apac.cn/gzdt/, Accessed in 2019.

[2] Mahmoud Khonji, Youssef Iraqi, Senior Member, Andrew Jones. Phishing Detection: A Literature Survey. IEEE Communications Surveys and Tutorials, 15(4), 2013, pp.2091-2121.

[3] Alessandro Acquisti, Idris Adjerid, Rebecca Balebako, Laura Brandimarte, Lorrie Faith Cranor, Saranga Komanduri, Pedro Giovanni Leon, Norman Sadeh, Florian Schaub, Manya Sleeper, Yang Wang, Shomir Wilson. Nudges for Privacy and Security: Understanding and Assisting Users‟ Choices Online. ACM Computing Surveys, 50(3), 2017, Article No. 44.

[4] María M. Moreno-Fernández, Fernando Blanco, Pablo Garaizar, Helena Matute. Fishing for phishers. Improving Internet users' sensitivity to visual deception cues to prevent electronic fraud. Computers in Human Behavior, Vol.69, 2017, pp.421-436.

[5] M. Junger, L. Montoya, F.-J. Overink. Priming and warnings are not effective to prevent social engineering attacks. Computers in Human Behavior, Vol.66, 2017, pp.75-87.

[6] El-Sayed M. El-Alfy. Detection of Phishing Websites Based on Probabilistic Neural Networks and K-Medoids Clustering. The Computer Journal, 60(12), 2017, pp.1745-1759.

[7] Cheng Huang, Shuang Hao, Luca Invernizzi, Yong Fang, Christopher Kruegel, Giovanni Vigna. Gossip: Automatically Identifying Malicious Domains from Mailing List Discussions. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (ASIA CCS 2017), Abu Dhabi, United Arab Emirates, April 2-6, 2017, pp.494-505.

[8] Frank Vanhoenshoven, Gonzalo Nápoles, Rafael Falcon, Koen Vanhoof, Mario Köppen. Detecting malicious URLs using machine learning techniques. In: Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI 2016), December 6-9, 2016.

[9] Joshua Saxe, Richard Harang, Cody Wild, Hillary Sanders. A Deep Learning Approach to Fast, Format-Agnostic Detection of Malicious Web Content. In: Proceedings of the 2018 IEEE Symposium on Security and Privacy Workshops (SPW 2018), San Francisco, CA , USA, August 2, 2018, pp.8-14.

[10] Longfei Wu, Xiaojiang Du, Jie Wu. Effective Defense Schemes for Phishing Attacks on Mobile Computing Platforms. IEEE Transactions on Vehicular Technology, 65(8), 2016, pp.6678-6691.

[11] R. Gowtham, Ilango Krishnamurthi. A comprehensive and efficacious architecture for detecting phishing webpages. Computers & Security, Vol.40, 2014, pp.23-37.

[12] Guang Xiang, Jason I. Hong, Carolyn Penstein Rosé, Lorrie Cranor. CANTINA+: A Feature-rich Machine Learning Framework for Detecting Phishing Web Sites. ACM Transactions on Information and System Security, 14(2), 2011, Article No. 21.

[13] Erzhou Zhu, Chengcheng Ye, Dong Liu, Feng Liu, Futian Wang, Xuejun Li. An Effective Neural Network Phishing Detection Model Based on Optimal Feature Selection. In: Proceedings of the 16th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2018), Melbourne, Australia, December 11-13, 2018, pp.781-787.

[14] Steve Sheng, Brad Wardman, Gary Warner, Lorrie Faith Cranor, Jason Hong, Chengshan Zhang. An Empirical Analysis of Phishing Blacklists. In: Proceedings of the 6th Conference on Email and Anti-Spam (CEAS 2009), Mountain View, California, USA, July 16-17, 2009.

[15] GitHub. Implementation for the Usage of Google Safe Browsing APIs (v4). [Online] Available: https://github.com/google/safebrowsing, Accessed in 2019.

[16] Jungmin Kang, Dohoon Lee. Advanced White List Approach for Preventing Access to Phishing Sites. In: Proceedings of the 2007 International Conference on Convergence Information Technology (ICCIT 2007), Hydai Hotel Gyeongui, Korea, November 21-23, 2007, pp.491-496.

[17] Mohsen Sharifi, Seyed Hossein Siadati. A Phishing Sites Blacklist Generator. In: Proceedings of the 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2008), Doha, Qatar, March 31-April 4, 2008, pp.840-843.

[18] Xiao Han, Nizar Kheir, Davide Balzarotti. PhishEye: Live Monitoring of Sandboxed Phishing Kits. In: Proceedings of the 23rd ACM conference on Computer and communications security (CCS 2016), Vienna, Austria, October 24-28, 2016, pp.1402-1413.

[19] Lung-Hao Lee, Kuei-Ching Lee, Hsin-Hsi Chen, Yuen-Hsien Tseng. POSTER: Proactive Blacklist Update for Anti-Phishing. In: Proceedings of the 2014 ACM SIGSAC Conference

on Computer and Communications Security (CCS 2014), Scottsdale, Arizona, USA November 3-7, 2014, pp.1448-1450.

[20] Ahmed Aleroud, Lina Zhou. Phishing environments, techniques, and countermeasures: A survey. Computers & Security, Vol.68, 2017, pp.160-196.

[21] Liang Shi, Derek Lin, Chunsheng Victor Fang, Yan Zhai. A Hybrid Learning from Multi-behavior for Malicious Domain Detection on Enterprise Network. In: Proceedings of the IEEE 15th International Conference on Data Mining Workshops (ICDMW 2015), Atlantic City, NJ, USA, November 14-17, 2015, pp. 987-996.

[22] Yue Zhang, Jason I. Hong, Lorrie F. Cranor. Cantina: a content-based approach to detecting phishing web sites. In: Proceedings of the 16th international conference on World Wide Web (WWW 2007), Banff, Alberta, Canada, May 8-12, 2007, pp.639-648.

[23] Xueni Li, Guanggang Geng, Zhiwei Yan, Yong Chen, Xiaodong Lee. Phishing detection based on newly registered domains. In: Proceedings of the 2016 IEEE International Conference on Big Data (BigData 2016), Washington DC, USA, December 5-8, 2016, pp.3685-3692.

[24] Luong Anh, Tuan Nguyen, Ba Lam To, Huu Khuong Nguyen, Minh Hoang Nguyen. An efficient approach for phishing detection using single-layer neural network. In: Proceedings of the 2014 International Conference on Advanced Technologies for Communications (ATC 2014), Hanoi, Vietnam, October 15-17, 2014, pp.435-440.

[25] Samuel Marchal, Jérôme François, Radu State ,Thomas Engel, PhishStorm: detecting phishing with streaming analytics. IEEE Transactions on Network & Service Management, 11(4), 2014, pp. 458-471.

[26] Routhu Srinivasa Rao, Syed Taqi Ali. PhishShield: A Desktop Application to Detect Phishing Webpages through Heuristic Approach. Procedia Computer Science, Vol.54, 2015, pp.147-156.

[27] Gunikhan Sonowal, K.S. Kuppusamy. PhiDMA - A phishing detection model with multi-filter approach. Journal of King Saud University - Computer and Information Sciences, In Press (DOI: https://doi.org/10.1016/j.jksuci.2017.07.005).

[28] Samuel Marchal, Giovanni Armano, Tommi Grondahl, Kalle Saari, Nidhi Singh, N. Asokan. Off-the-Hook: An Efficient and Usable Client-Side Phishing Prevention Application. IEEE Transactions on Computers, 66(10), 2017, 1717-1733.

[29] UCI Machine Learning Repository. Center for Machine Learning and Intelligent Systems. [Online] Available: http://archive.ics.uci.edu/ml/ index.php, Accessed in 2019.

[30] Phishtank. Out of the Net, into the Tank. [Online] Available: https:// www.phishtank.com/, Accessed in 2019.

[31] Ankit Kumar Jain, B. B. GuptaEmail. A novel approach to protect against phishing attacks at client side using auto-updated white-list. EURASIP Journal on Information Security, 2016(1), 2016, Article No.9.

[32] Guang Xiang, Jason I. Hong. A Hybrid Phish Detection Approach by Identity Discovery and Keywords Retrieval. In: Proceedings of the 18th international conference on World Wide Web (WWW 2009), Madrid, Spain, April 20-24, 2009, pp.571-580.

[33] Gaurav Varshney, Manoj Misra, Pradeep K. Atrey. A phish detector using lightweight search features. Computers & Security, Vol.62, 2016, pp.213-228.

[34] Firdous Kausar, Bushra Al-Otaibi, Asma Al-Qadi, Nwayer Al-Dossari. Hybrid Client Side Phishing Websites Detection Approach. International Journal of Advanced Computer Science and Applications, 5(7), 2014, pp.132-140.