Malicious Website Detection

An application using machine learning

Gagneet Sachdeva & Harpreet Kour

A project presented for the
Cybersecurity & AI Class

Master of Science in Computer Science
University of Washington
Bothell
17th March, 2023

Contents

1 INTRODUCTION

Cybersecurity is a challenging task especially among the rising threats on the internet. Malicious websites are a common and serious threat to cybersecurity. They host unsolicited content and lure unsuspecting users to become victims of scams (monetary loss, theft of private information, and malware installation), and cause losses of billions of dollars every year. The limitations of traditional security management technologies are becoming more and more serious given this exponential growth of new security threats, rapid changes of new IT technologies, and significant shortage of security professionals. Most of the attacking techniques are realized through spreading compromised URLs (or the spreading of such URLs forms a critical part of the attacking operation). The Uniform Resource Locator (URL) is the well-defined structured format unique address for accessing websites over World Wide Web (WWW). Generally, there are three basic components that make up a legitimate URL
i.) Protocol: It is basically an identifier that determines what protocol to use e.g., HTTP, HTTPS, etc.
ii) Hostname: Also known as the resource name. It contains the IP address or the domain name where the actual resource is located.
iii) Path: It specifies the actual path where the resource is located

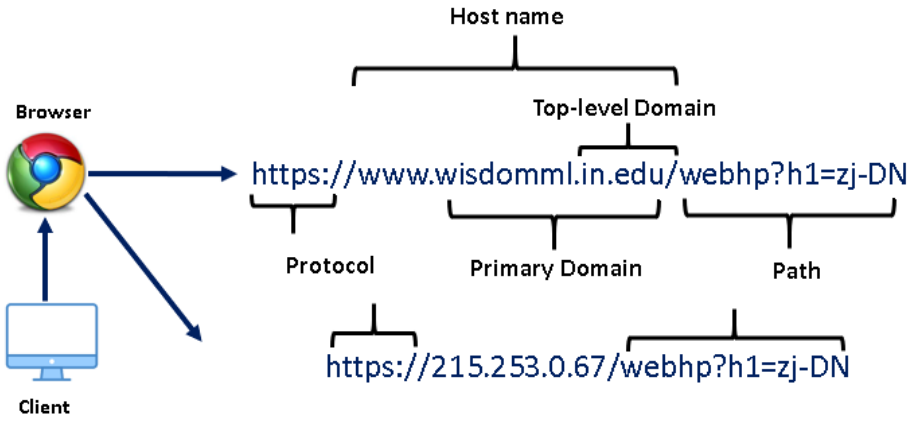


Figure 1: Components of a URL

One common technique in the identification of malicious URLs is the use of a blacklist. While blacklisting a URL has been effective to some extent, the fact that these URLs are rapidly evolving means blacklists are not sufficient defense against this form of attack. Machine learning techniques are proving to be much more useful as they can detect malicious url even if they have not seen it before unlike a blacklist.

1.1 Problem Statement

We formulate malicious URL detection as a binary classification task for two-class prediction: "malicious" versus "benign". Automated malicious URL detection is two-fold:

(1) Feature Representation: Extracting the appropriate feature representation from a d-dimensional feature vector representing the URL; and

(2) Machine Learning: Learning a prediction function which predicts the class assignment for any URL instance x using proper feature representations.

While the first part of feature representation is often based on domain knowledge and heuristics, the second part focuses on training the classification model via a data driven optimization approach. The goal of machine learning for malicious URL detection is to maximize the predictive accuracy, precision and minimize false positive rate.

1.2 Basic Approach

The first step involves collection of raw data and converting it into a .csv file with two features namely, **URL** and **label**. Post that multiple features are extracted out of the URL under three main categories, lexical features, host features and content based features. Once a basic dataset is ready we apply feature reduction techniques to reduce our columns and prepare it for modelling. As we know machine learning algorithms only support numeric inputs hence we are using lexical, host and content numeric features from input URLs. So the input to machine learning algorithms will be the numeric features rather than actual raw URLs. Here, we will be using different types of machine learning models like linear models, tree-based models, unsupervised learning, statistical models and neural networks. This is done to identify the best algorithm which will then be used for the prediction function. The input to this prediction function is the website which the user is browsing. The output is either SAFE or UNSAFE based on our prediction algorithm. The application that we will be integrating our model with is a google chrome based plugin. This plugin will show a pop-up whenever the user visits a website showing whether the website is safe or malicious. Below is the basic architecture of our application.

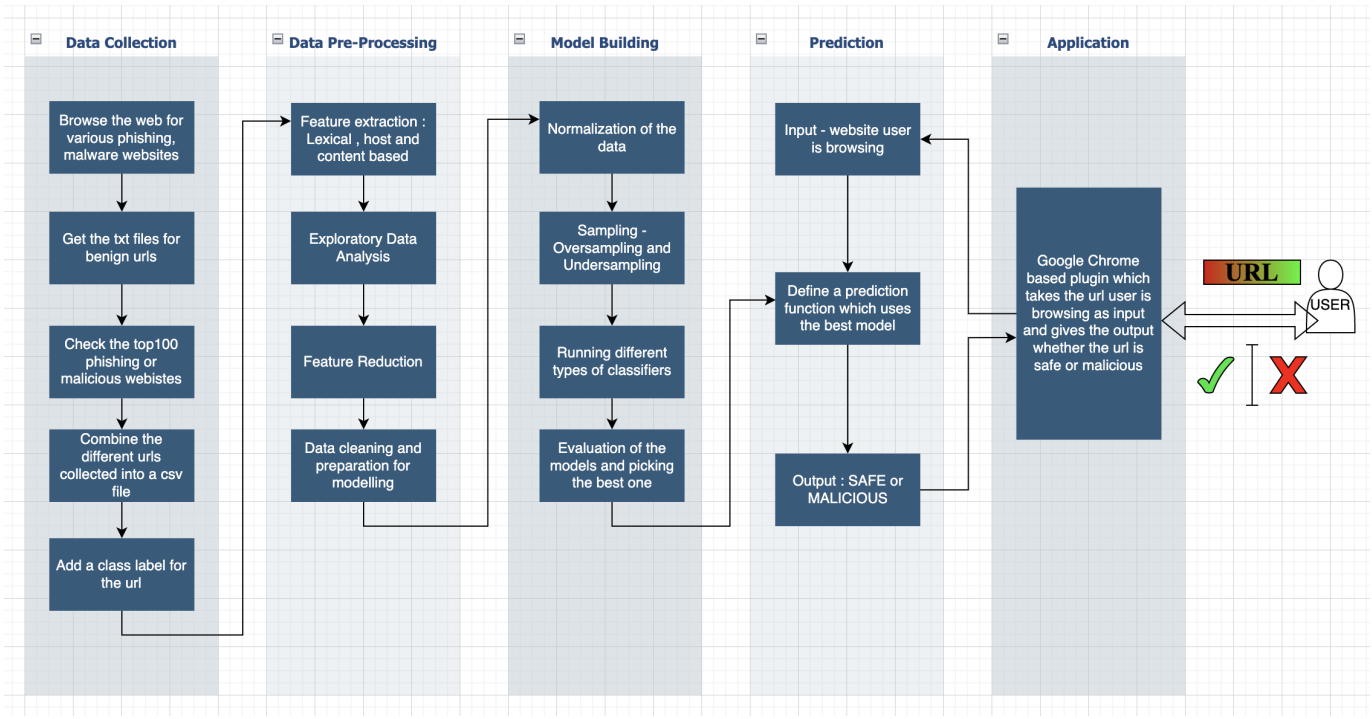


Figure 2: Basic Architecture of the Application

2 DATA COLLECTION

The data collection process mostly involved looking for malicious websites using web based sources. Following are the sources :

1. Norton's dangerous websites
2. PhishStrom Dataset
3. Kaggle Dataset

URLs were collected from these sources and divided into four categories - benign, defacement, phishing and malware. The latter three are combined to form the malicious class. All the URLs are copied from text files or other formats into a single csv file along with their type. The csv file contains only two columns, URL and type. This is how multiple URLs are merged into a single file. This data is good for our application as we are trying to classify

URLs into malicious and benign. The kaggle dataset used here is the most common dataset used for malicious URL detection system as it has been cited in multiple papers. The variety in the dataset makes it a favourable choice for our project.

Although the PhishStorm and the kaggle datasets are both huge we have only used a percentage of their data. This is attributed to the fact that our computing resources and time were limited. After much deliberation, we decided to go ahead with 1000 records in our dataset including both malicious and non-malicious data. In the process of reducing our records for better processing time we have definitely compromised on the performance of our model. The more data the model is trained on the better it is in such applications. Also the datasets used are not new so it does not include the latest malicious websites that may have come up. Another issue with using this data is the class imbalance. We have more malicious data as compared to benign data. This is taken care of using **sampling techniques** provided by sklearn library. We have used an oversampling technique for balancing our classes.

3 DATA PRE-PROCESSING

For deciding our features we have read upon the already existing work on malicious url classification [1],[2],[3] Most of the papers are using three different types of features. Upon looking up at some kaggle workbooks it is clear that the most commonly used features are the lexical features of the url for making prediction. The process of data pre processing here involved two stages namely, feature extraction and then feature reduction. We have also included some exploratory data analysis as none of us are domain experts in this and it would be helpful for us to know the data.

3.1 Feature Extraction

The first step that was taken on the csv file was that of feature extraction. The first two features that are extracted are **DOMAIN and IP_ADDRESS**. They are extracted using **TLD and SOCKET** libraries of python respectively. These are extracted initially and saved along with the csv file containing URL and TYPE.

For DOMAIN following code is used :

```
from tld import get_tld, is_tld
# Function to get the top level domain of the url
def process_tld(url):
    try:
        res = get_tld(url, as_object = True, fail_silently=False,fix_protocol=True)
        pri_domain= res.parsed_url.netloc
    except :
        pri_domain= None
    return pri_domain
```

For IP_ADDRESS following code is used :

```
import socket
# Function to get the ip_address of the url
def ip_address(url):
    try:
        hostname = socket.gethostbyname(url)
        return(hostname)
    except:
        hostname = '0.0.0.0'
        return(hostname)
```

Post that **LABEL** field is generated using the following logic :

```
if type == 'benign' :
    label = 0
else :
    label = 1
```

At this point we have a csv file with 50k datapoints having columns url, type, domain and ip_address. This is what is done in **Data_Collection_Formatting.ipynb** file. Out of this we have randomly selected 1000 records and created a file - **df1000.csv**. This file is used further for feature extraction. We tried to run the feature extraction with different number of rows like 50k, 10k and 3k. For all the runs our processing capabilities were not enough and the content-based features were taking more than 10 hours. Hence the number we settled on is 1000 rows. Post this three types of features were extracted from the feature URL which have been explained below and covered in the file **Feature_Extraction.ipynb**:

- Lexical features
  Lexical features are features obtained from the properties of the URL name (or the URL string)[4]. The motivation is that based on how the URL "looks" it should be possible to identify malicious nature of a URL. For example, many obfuscation methods try to "look" like benign URLs by mimicking their names and adding a minor variation to it. Below are the list of lexical features extracted.

Lexical Feature	Description
url_len	length of the url
count_@	count of '@' characters in the url string
count_?	count of '?' characters in the url string
count_-	count of '-' characters in the url string
count_=	count of '=' characters in the url string
count_.	count of '.' characters in the url string
count_#	count of '#' characters in the url string
count_%	count of '%' characters in the url string
count_+	count of '+' characters in the url string
count_$	count of '$' characters in the url string
count_!	count of '!' characters in the url string
count_*	count of '*' characters in the url string
count_,	count of ',' characters in the url string
count_//	count of '//' characters in the url string
count_alphas	count of alphabets in the url string
count_digits	count of digits in the url string
count_puncs	count of punctuations in the url string
count_www	count of 'www' in the url string
abnormal_url	whether url is matching with any abnormal url or not, yes : 1 no : 0
https	whether the url is httpSecure or not, yes : 1 no : 0
shortening_service	whether the url has used any url shortening service or not, yes : 1 no : 0
has_ip_address	whether the given url has an ip address or not, yes : 1 no : 0

- Host based features
  Host-based features are obtained from the host-name properties of the URL[5]. They allow us to know the location, identity, and the management style and properties of malicious hosts. To extract the host-based features we have used the **WHOIS** API of python. The WHOIS information comprises domain name registration dates, registrars and registrants. The Location information comprises the physical geographic location - e.g. country/city to which the IP address belongs. Below are the list of host-based features extracted :

Host Feature	Description
city	registered city of the website
country	registered country of the website
expiration_date	expiration date of the website
creation_date	creation date of the website
age	number of days the website has existed since it's creation date till today
intended_life	number of days the website has from it's creation date till its expiration date
life_remaining	number of days the website has from today till its expiration date

- Content based features
  Content-based features are those obtained upon downloading the entire webpage. As compared to URL-based features, these are "heavy-weight", as a lot of information needs to be extracted, and at the same time, safety

concerns may arise. However, with more information available about a particular webpage, it is natural to assume that it would lead to a better prediction model. These were the features which took maximum time and effort to extract. To extract these features we have used **REQUEST** and **PYQUERY** libraries of python.

Content-based Feature	Description
status	status of the website
num_hyperlinks	number of hyperlinks on the page
num_embed	number of embed tags on the page
num_object	number of object tags on the page
num_iframe	number of iframe tags on the page
num_htmltags	number of html tags on the page
num_scripttags	number of script tags on the page

The reason we have used these features is that it makes our model more efficient. Considering we are not using much data it made sense to improve performance using other methods. Feature extraction took maximum time for us especially the content-based features as it involved scrapping the websites for different items. After this process we had 40 features excluding our class label.

3.2 Exploratory Data Analysis

Using the 40 features we came up with some interesting details about our data. Below are the few details. This is covered in **EDA_Feature_Reduction.ipynb** file.

1. Wordclouds to determine good and bad domain names. These can also be used to extract additional features. Like whether the most common words(bad and good) are present in the url or not.

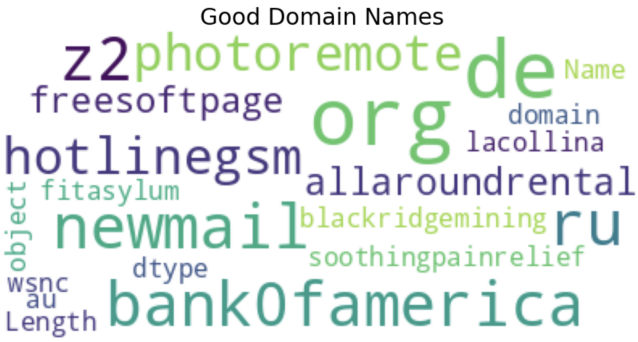


Figure 3: Good Domain Names

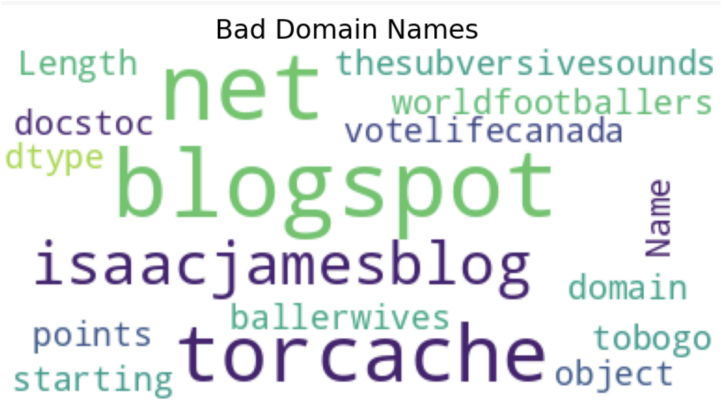


Figure 4: Bad Domain Names

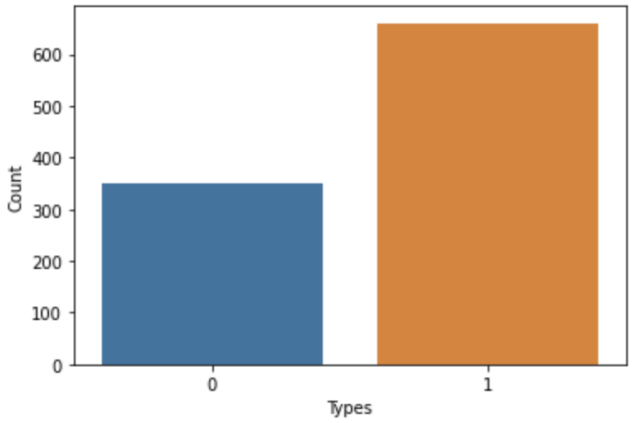


Figure 5: Data Distribution

2. Distrbution of data - This tells us that there is a slight class imbalance which needs to be fixed.

3. Length of URLs - We can see that most URLs fall into the length of 10 to 120 characters

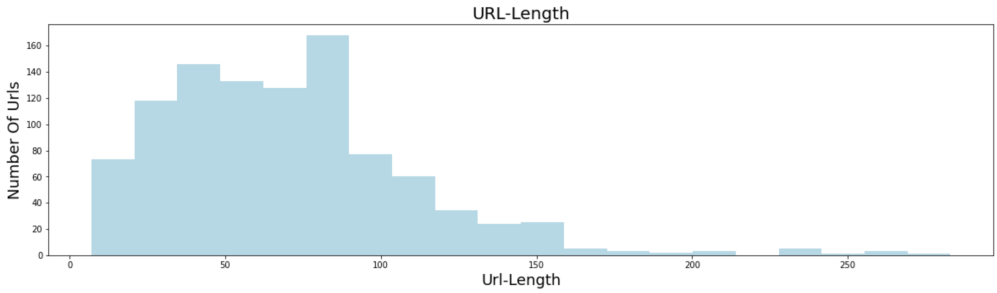


Figure 6: URL length distribution

4. Age of URLs - It is clear that most URLs are only 500 days old.

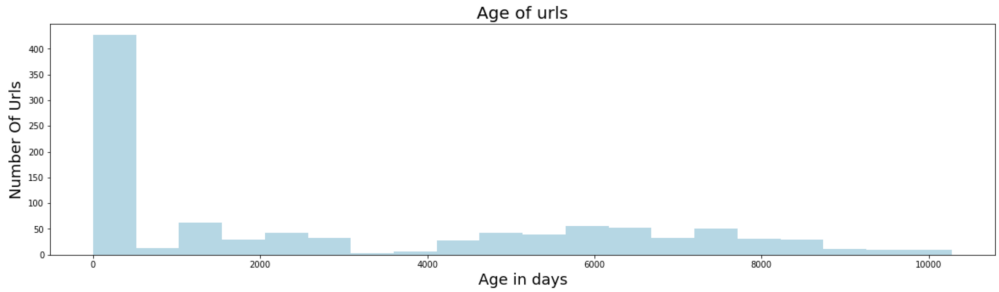


Figure 7: URL age distribution

3.3 Feature Reduction

For feature reduction we have used multiple strategies :

1. Removing the categorical features - In this case it is easier to drop the categorical features rather than use label encoding. This is because the text features have very different data and cannot necessarily be grouped. Type column can be encoded but since it is used to get the label column it is better to drop it. We have also dropped the

date columns as we already extracted the age, intended lifespan and expected lifespan from it. Now they are not useful for our model. The dropped columns include

URL	TYPE	DOMAIN	IP_ADDRESS	CITY	COUNTRY	EXPIRATION_DATE	CREATION_DATE

2. Removing low variance features - Features that have low variance or have the value 0 in all the rows have been removed. These features are not helpful in prediction. The dropped columns include

COUNT_@	COUNT_#	COUNT_+	COUNT_$	COUNT_!	COUNT_*	COUNT_,
HTTPS	HAS_IP_ADDRESS	NUM_EMBED	NUM_OBJECT			

After this we were left with 21 features excluding the label column.

3. Correlation Plot - Next we decided to bring down our features even more and drew up a correlation matrix with the existing features. It was plotted using **SEABORN** library's **HEATMAP** function. As is clear from the plot we found 3 sets of highly correlated features, url_len - count_alphas, abnormal_url - count_//, num_htmltags - num_hyperlinks. Hence we decided to remove

COUNT_ALPHAS	COUNT_//	NUM_HTMLTAGS

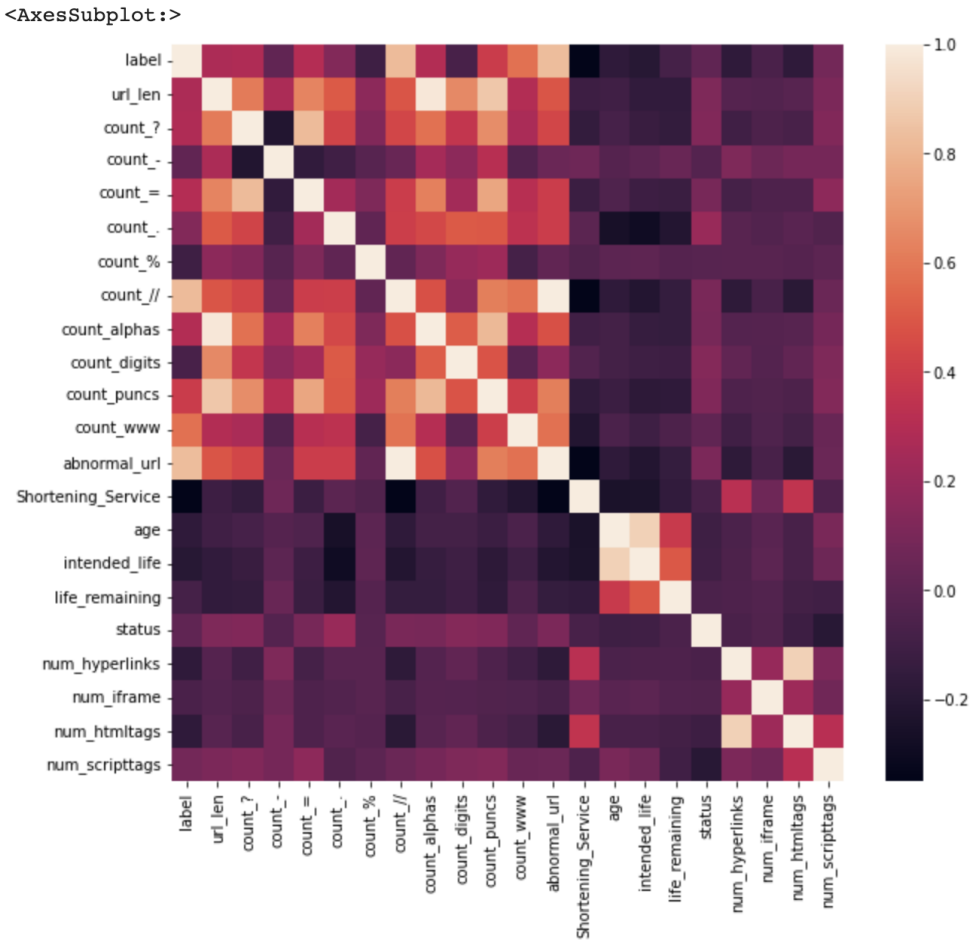


Figure 8: Correlation Plot

The dataset now consists of 1009 rows with 18 features. We have not applied PCA to our data.

4 MODEL BUILDING

For the model building we have used different types of machine learning models like linear models, tree-based models, unsupervised learning, statistical models and neural networks. This is done to get the best performing model which is later used in the predictor. This part is covered in the file **Model_Building.ipynb**

4.1 Data Split

The shape of our dataset is (1009,18). Since different columns have different numerical values it is important to scale or normalize the dataset. Here **MinMaxScaler** has been used from **sklearn.preprocessing**. While exploring our data previously we saw that there is a class imbalance. To overcome this issue, oversampling technique of **SMOTE** has been used. The reason we have used oversampling is two-folds. We have an excess of malicious data, oversampling will create extra benign data which will help in resolving class imbalance. Also by oversampling, we can see how our model fares against data that is being created by the sampler and not actual websites. SMOTE is an advanced oversampler and does not create duplicate datapoints but synthetic data points that are slightly different from the original data points. Hence it has been used. Then the dataset was split into 70% for training and 30% for testing using **train_test_split** from **sklearn.model_selection**.

4.2 Model Training

URL strings tend to be very unstructured and noisy. Hence, it was imperative to choose a classification algorithm that is not too sensitive to fluctuations in the training data. So it was decided to train multiple types of models and choose the best one. All the models are trained on the same data and are being tested on the same data as well. Our models consists of 5 groups of classifiers and 7 models :

1. Tree Based Models - Decision Trees, Random Forest, XGBoost
2. Linear Model - Logistic Regression
3. Unsupervised Learning - K means Clustering
4. Statistical Model - Naive Bayes
5. Neural Networks - Multi Layer Perceptron

4.3 Model Evaluation

We have optimized for precision, accuracy and false negative rate. We want to maximize catching the malicious url among different urls and minimize the wrong classification of urls. Our primary concern is malicious url being classified as benign which can be disastrous for many users. Considering these two are the major objectives it makes sense to optimize for precision, accuracy and false negative rate. We want a system that is able to detect malicious url with the least number of false alarms. Also these are the metrics that are majorly focused on according to papers[1],[2] for malicious url detection. The performance of the model is considered high if the precision is high and the false negative rate is low. Below are the tabulated results and confusion matrices we got from the different models :

Model	True Positives(TP)	False Positives(FP)	True Negatives(TN)	False Negatives(FN)
Logistic Regression	191	3	95	14
Decision Tree	200	0	98	5
Random Forest	192	10	88	13
XGBOOST	204	0	98	1
K means Clustering	66	46	52	139
Naive Bayes	159	3	95	46
Multi Layer Perceptron	193	4	94	12

```
Confusion Matrix :
[[ 95   3]
 [ 14 191]]
```

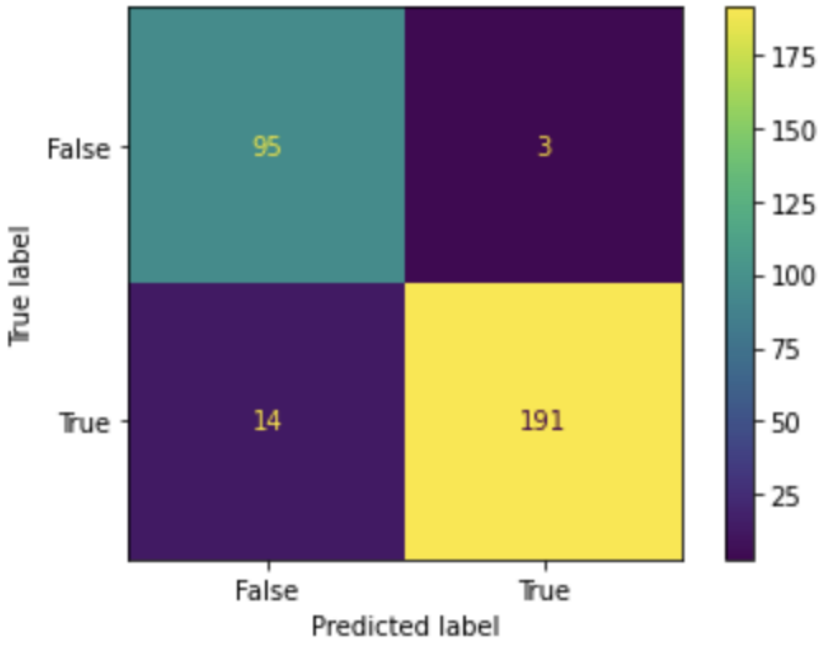


Figure 9: Confusion Matrix for Logistic Regression

```
Confusion Matrix :
[[ 98   0]
 [  5 200]]
```

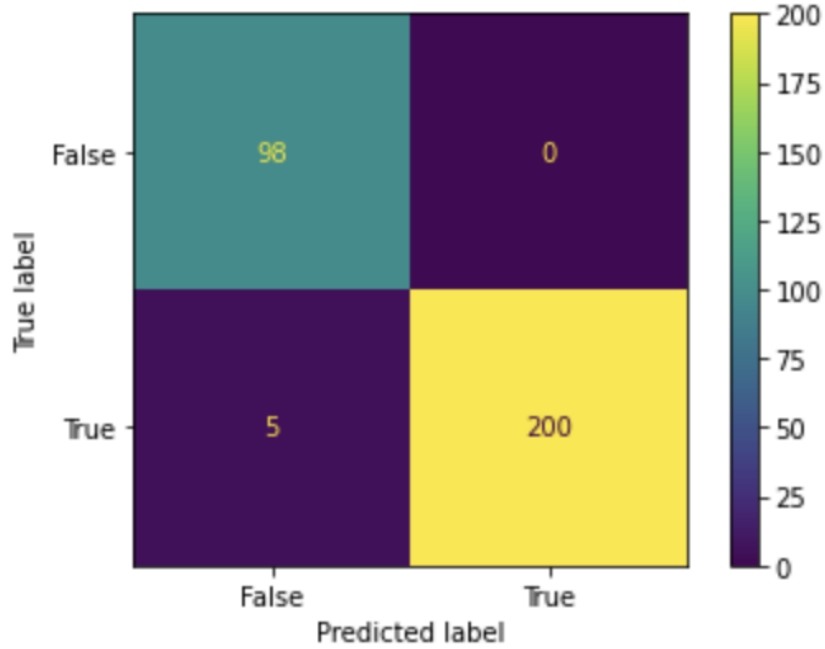


Figure 10: Confusion Matrix for Decision Tree

```
Confusion Matrix :
[[ 88  10]
 [ 13 192]]
```

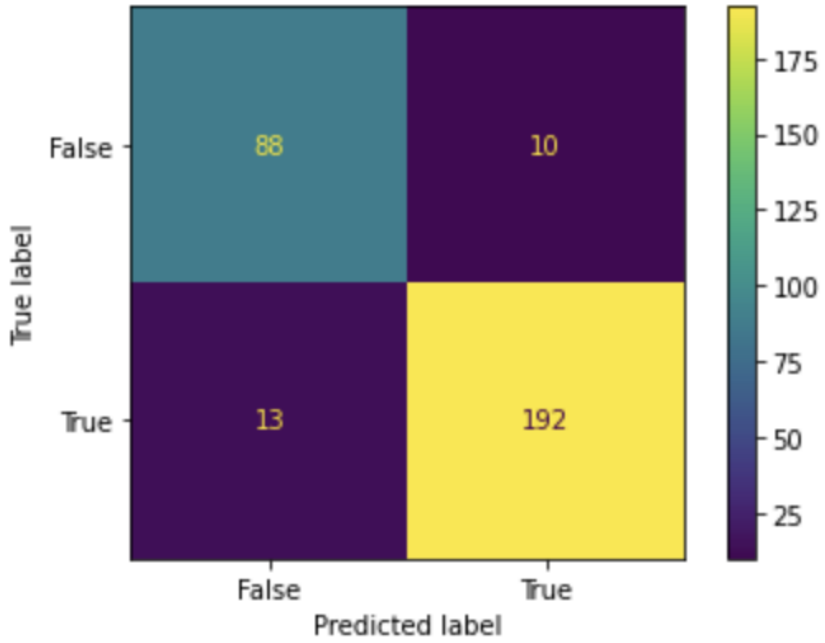


Figure 11: Confusion Matrix for Random Forest

```
Confusion Matrix :
[[ 98   0]
 [  1 204]]
```

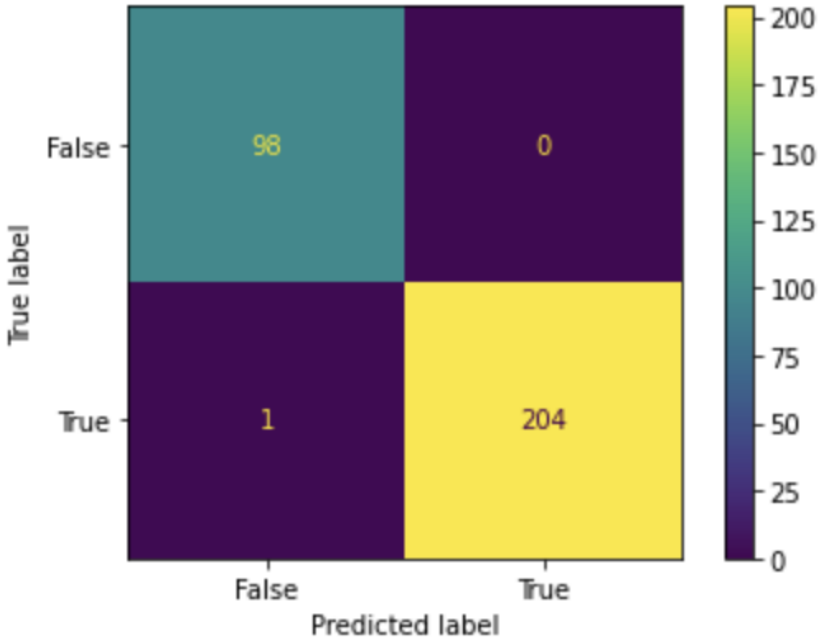


Figure 12: Confusion Matrix for XGBoost

```
Confusion Matrix :
[[ 52  46]
 [139  66]]
```

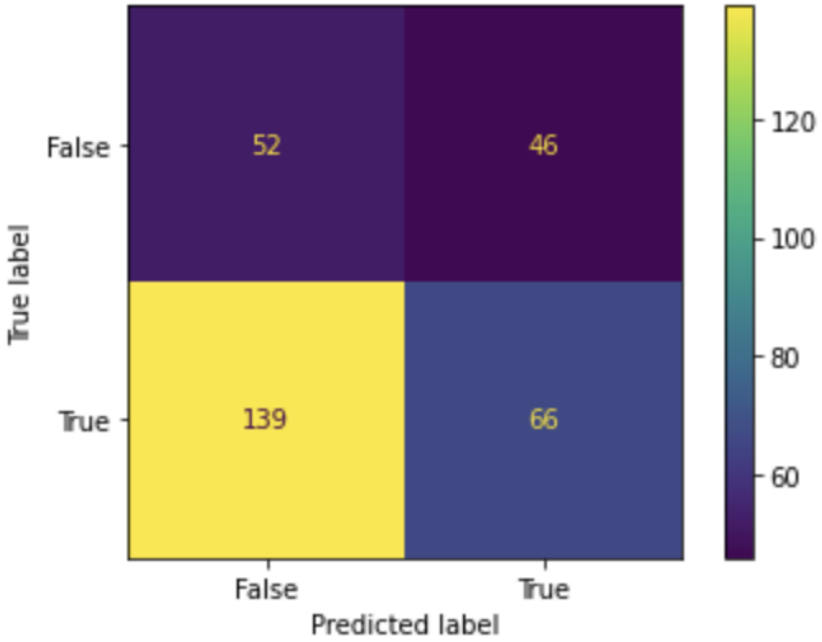


Figure 13: Confusion Matrix for K means Clustering

```
Confusion Matrix :
[[ 95   3]
 [ 46 159]]
```

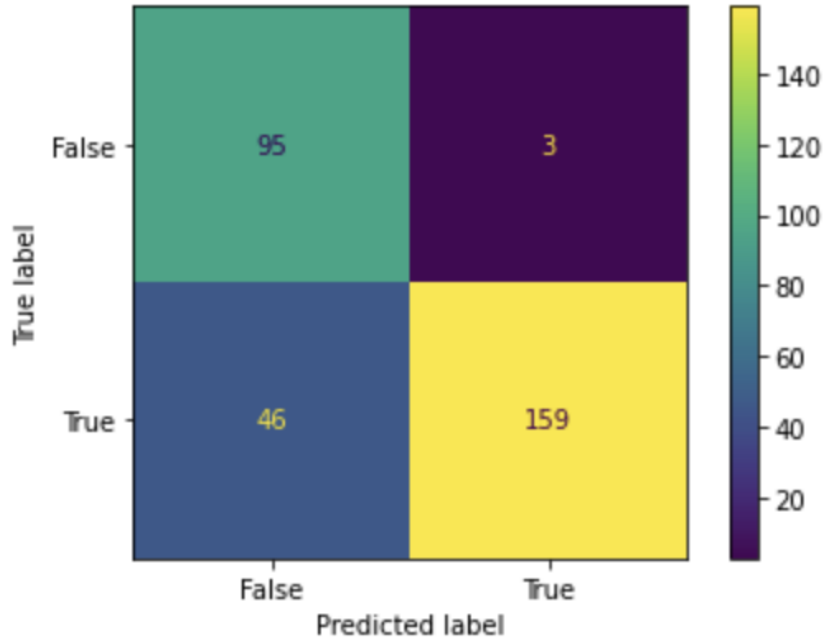


Figure 14: Confusion Matrix for Naive Bayes

```
Confusion Matrix :
[[ 94   4]
 [ 12 193]]
```

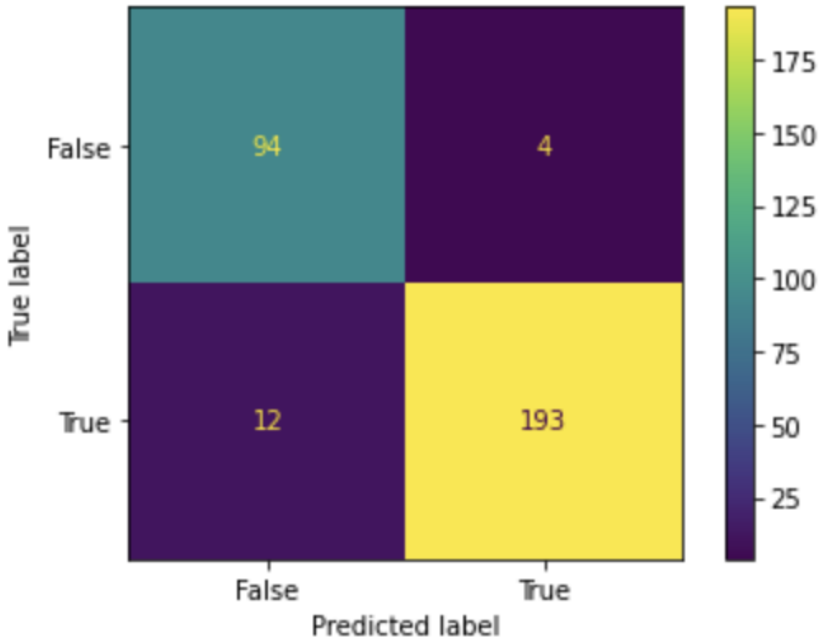


Figure 15: Confusion Matrix for Multi Layer Perceptron

Model	Accuracy	Precision	AUC	False Negative Rate(FNR)
Logistic Regression	0.94	0.985	0.950	0.068
Decision Tree	0.98	1.0	0.988	0.024
Random Forest	0.92	0.95	0.917	0.063
XGBOOST	1.0	1.0	0.997	0.005
K means Clustering	0.39	0.589	0.426	0.678
Naive Bayes	0.84	0.981	0.872	0.224
Multi Layer Perceptron	0.95	0.979	0.950	0.058

The metrics have been calculated using the following formulas :

$$accuracy = \frac{TP + TN}{(TP + FP + TN + FN)}$$

$$precision = \frac{TP}{(TP + FP)}$$

$$FNR = \frac{FN}{(FN + TP)}$$

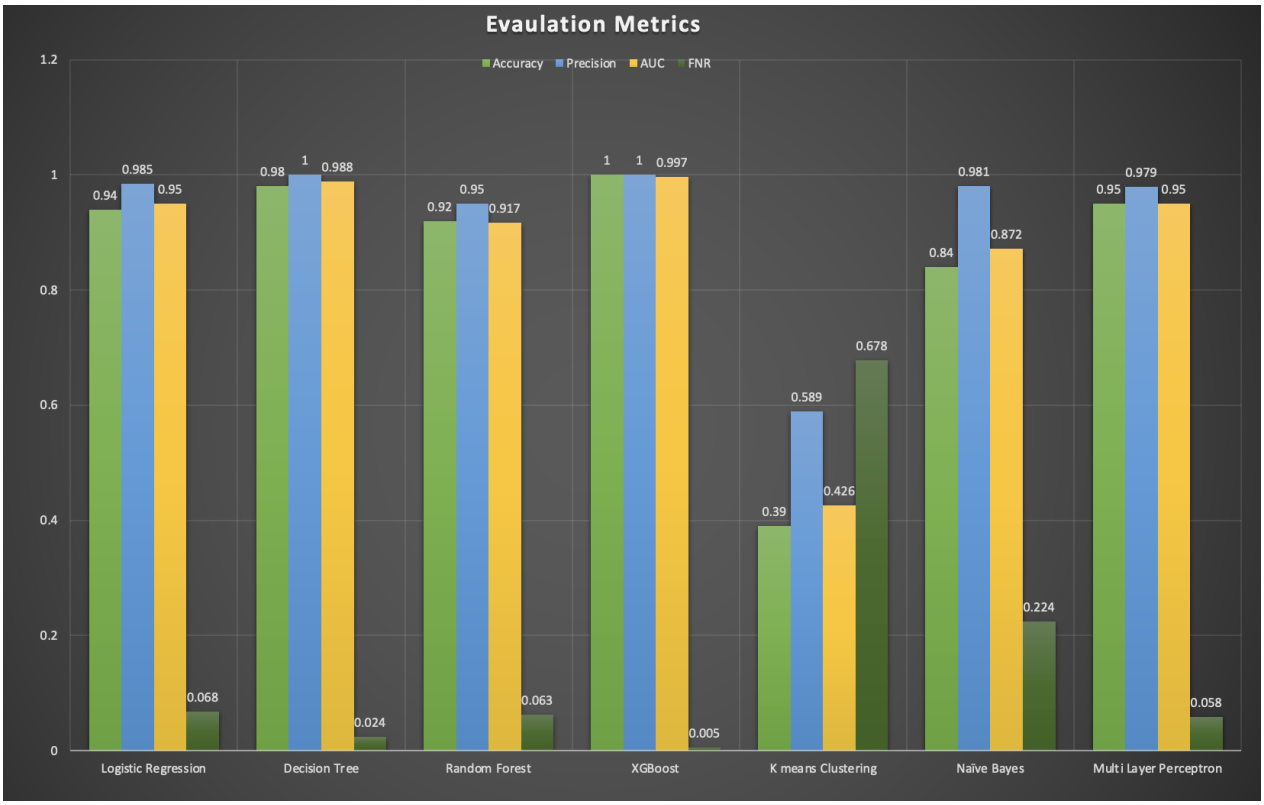


Figure 16: Graph showing Evaulation Metrics

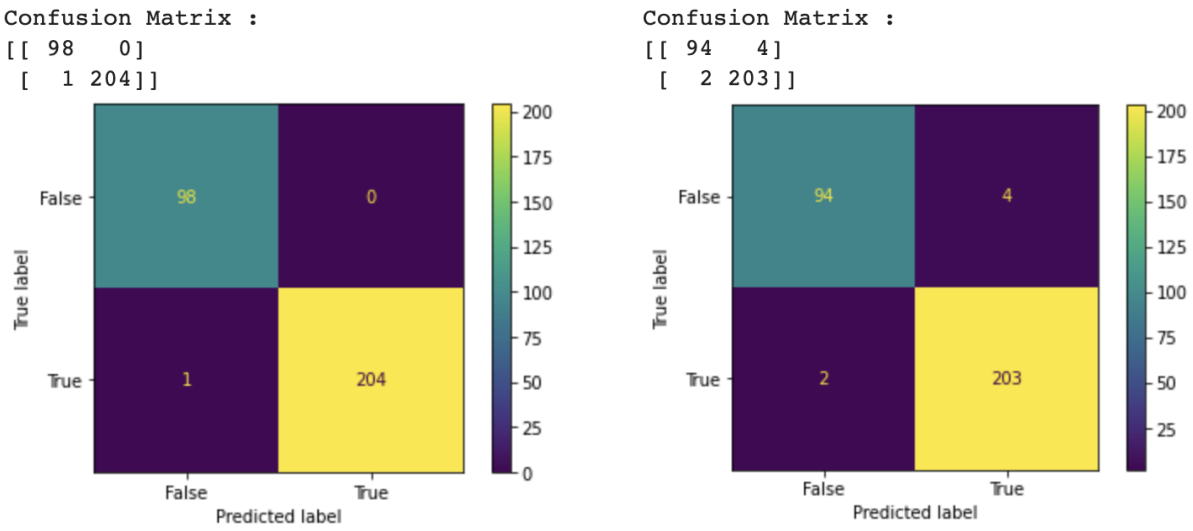
Results : It is clear from the graph and the tables that tree-based algorithms are best for malicious url detection. They have very good accuracy and precision values. Decision tree and xgboost both give a 100% precision value. Both their false negative rates are also extremely low. Logistic regression and neural network performs well also. They have a good accuracy around 95% and low false negative rates around 6%. Naive Bayes performs averagely but K-means clustering algorithm performs terribly. It is even worse than random guessing as it has an accuracy of 39%. The false negative rate is also very high for unsupervised learning. Hence, it is safe to say that clustering techniques must not be used while classifying malicious urls. Best way to classify them is to use one of the tree-based algorithms.

4.4 Model Selection

As is clear from the graph, XGBOOST is the best performing algorithm in our case. Hence we decided to proceed forward with it. The algorithm is already performing exceptionally but we decided to tune some hyperparameters and compare the performance. We have used the most common hyperparameters that are used for tree based learners like

- max_depth: The maximum depth per tree. A deeper tree might increase the performance, but also the complexity and chances to overfit.
- learning_rate: The learning rate determines the step size at each iteration while your model optimizes toward its objective. A low learning rate makes computation slower, and requires more rounds to achieve the same reduction in residual error as a model with a high learning rate. But it optimizes the chances to reach the best optimum.
- n_estimators: The number of trees in our ensemble. Equivalent to the number of boosting rounds.
- subsample: Represents the fraction of observations to be sampled for each tree. A lower values prevent overfitting but might lead to under-fitting.

It is clear that adding the hyperparameters is not helping us therefor we have used the model without hyperparameter tuning.



(a) XGBOOST without hyperparameter tuning

(b) XGBOOST with hyperparameter tuning

Figure 17: Comparison

Below figure shows the feature importance for XGBOOST algorithm. We can see that features 5 and 6, count_% and count_// are most important in predicting the outcomes

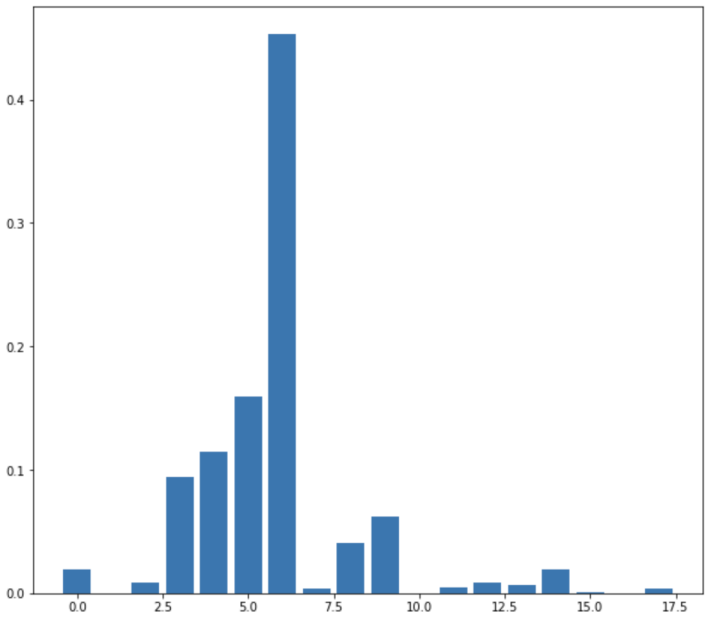


Figure 18: Feature Importance

5 APPLICATION DEVELOPMENT

5.1 Application Overview

After training and evaluating our model to detect malicious URL, we can leverage this model to build a real-world application i.e. a web extension that will alert the user if they try to visit any malicious website. Our extension is currently designed to work with Google Chrome browser. The user can use this extension by clicking on the "Malicious URL Detector" extension icon to check if the URL they are trying to visit is safe or not. If the URL is detected to be malicious, it will display a warning message to our user and advise them not to visit the site , thereby providing an additional layer of protection against phishing attacks, malware websites.

5.2 Application Building

Application Building for chrome extension is divided into two parts:

1. Developing a basic chrome extension
2. Integrating our machine learning model in this extension to detect malicious URL. Since chrome extension only allows HTML, CSS and Java script files, we had to dump our python model into model.pkl file using joblib and then fetch this model.pkl file in our java script using pyodide.js
(https://pyodide.org/en/stable/usage/loading-packages.html)

5.2.1 Application Development

Development of Basic Chrome Extension starts with preparing manifest.json file that contains all the details required for that extension like name, description, manifest version, permissions required, actions, background and content scripts etc.Then we create popup.html file which contains the title of the extension, a check now button, button id and the script sources which the popup of our extension will execute. We then move on to creating the scripts(popup.js & content.js) according to which the action will be performed in our extension's popup. In our content script we will load our python machine learning model through Pyodide and input our current tab url to predict whether that url is safe or malicious. Based on the prediction made by our machine learning model, we will send the response to our extension's popup which will display whether the site is safe to proceed or will display the warning if the url is detected as malicious.

5.2.2 Testing

Application has been tested with malicious and safe URLs.

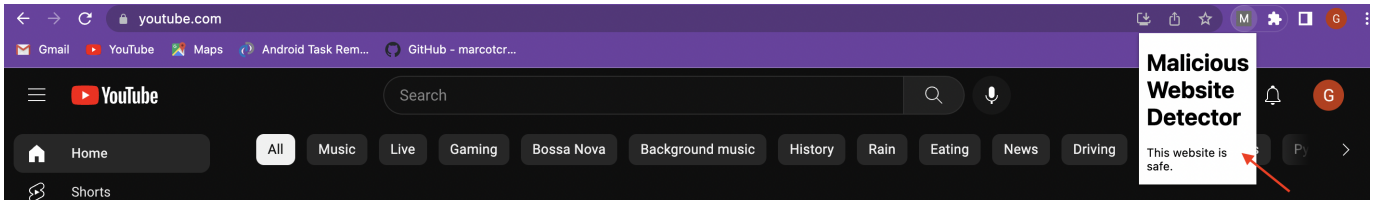


Figure 19: Benign Website

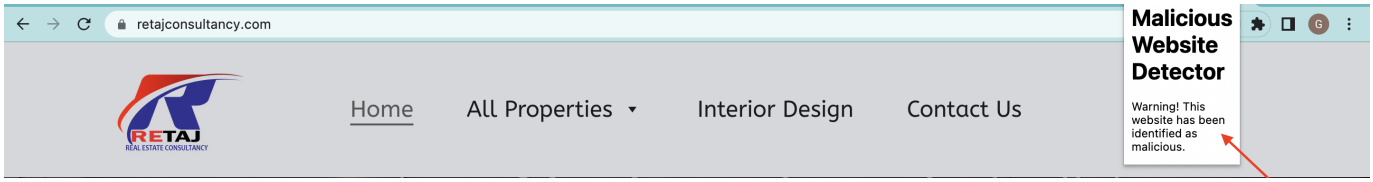


Figure 20: Malicious Website

5.3 Application Setup

Steps to Install Chrome Extension:

- Download the src folder of our code in your local machine
- Open Chrome Browser –> Go to More Options –> Go to More Tools –>Extension

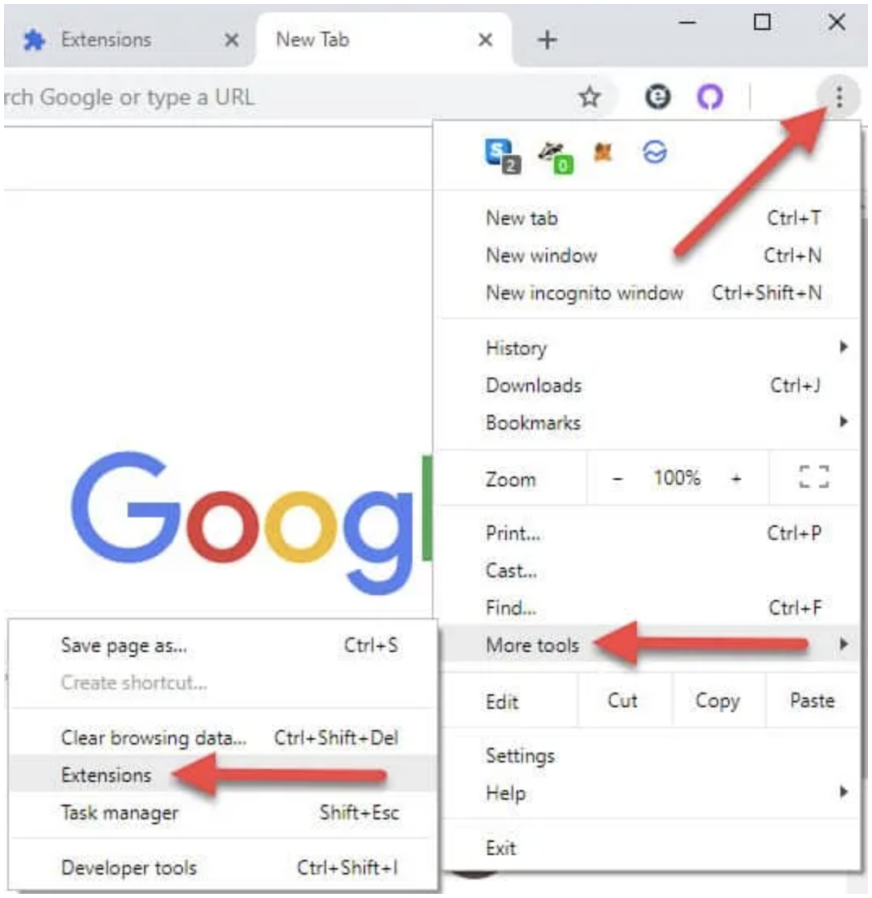


Figure 21: Installation

- Enable the Developer Mode

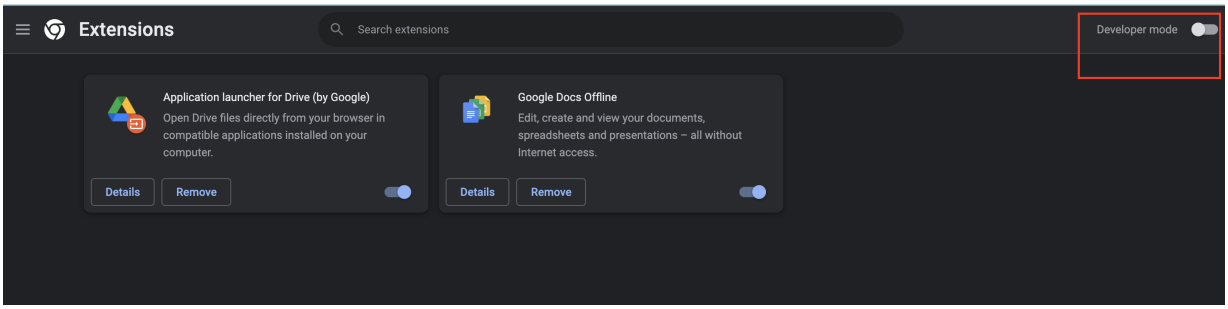


Figure 22: Developer Mode

- Click on 'Load Unpacked Extension'

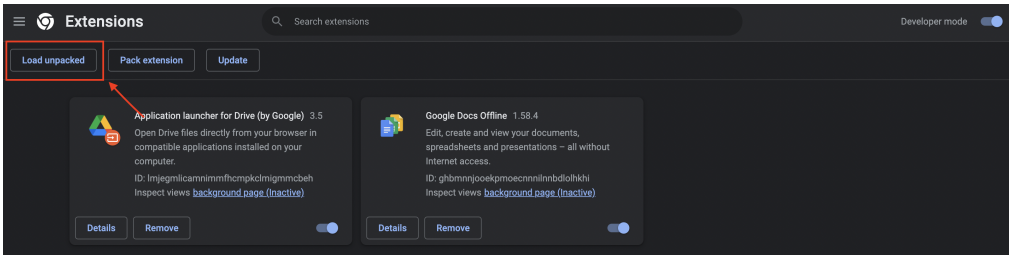


Figure 23: Load Unpacked Extension

- Select the src folder downloaded previously.

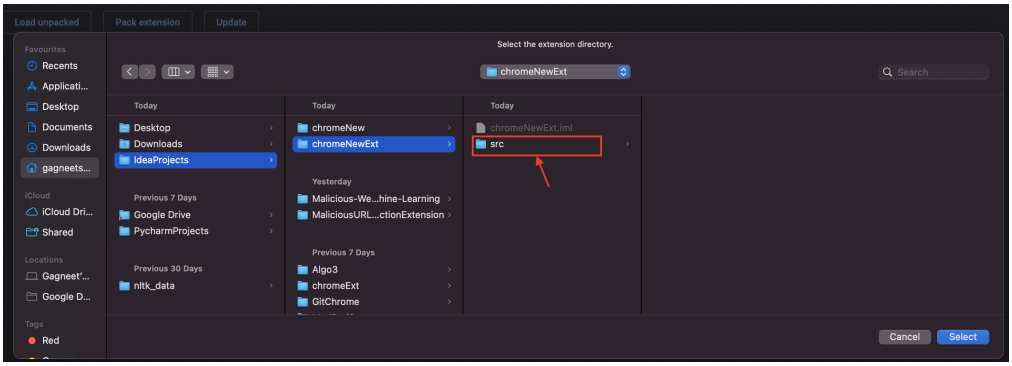


Figure 24: SRC Folder

- After src folder is loaded successfully, you should be able to see extension info as below.

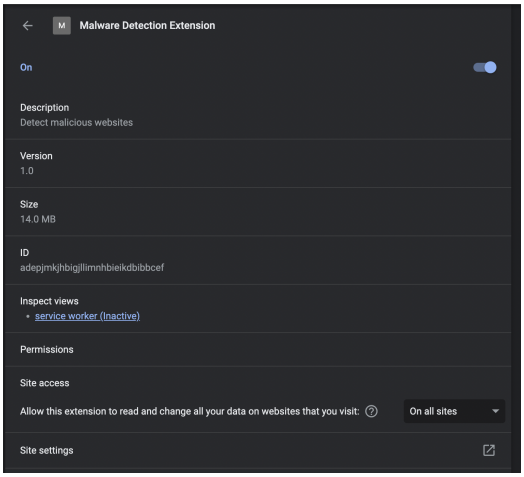


Figure 25: Extension Info

- Now open any webpage on the chrome browser and click on the extension icon.

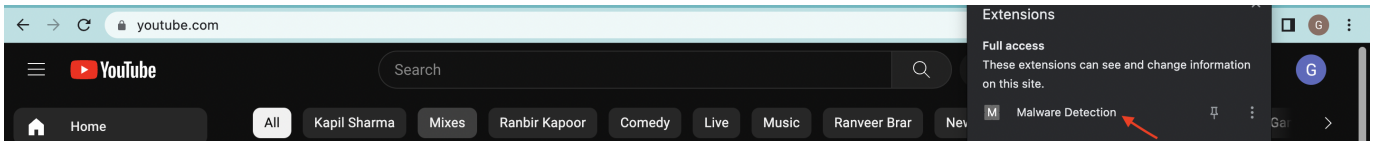


Figure 26: Using the extension

After clicking the extension icon, user should be able to see the message in the popup which states whether the URL is safe to use or malicious.

Figure 27: Output

5.4 Application Performance

Usability - The plugin is extremely easy to use. The look and feel of the plugin makes it quite user-friendly. It is not taking up much space on the browser window and hence not distracting for the user.

Effectiveness - Application is effective in identifying malicious websites. Out of the 10 websites we tested it for, it was able to identify 9 of them correctly. Although we are facing issues in integrating it with the ML model. Considering the time limitation we have submitted the working code.

5.5 Limitations

One limitation that can been seen in terms of design of the plugin is that you need to click on it. It would be easier if just when you visit the website the safe or malicious keyword appears. Also our application cannot measure the level of risk. It is just telling us whether the website is malicious or not.

5.6 ART Attacks

Adversarial machine learning technique aims to exploit models by developing harmful attacks using accessible model information [9]. The main reason for exploitation is to affect machine learning and fail it. Here, we will examine the adversarial attack on our models.

- Grey-Box Attack: The grey-box attacks use only training access to the target model to create adversarial samples.
- Black-Box Attack : Attackers have no access to the system. ZOO Attack may directly approximate the gradients of the target url using zeroth order stochastic descent.
- Poisoning Attack : Data can be poisoned by introducing specific lexical characters in the url which determine it's maliciousness.

6 TOOLS AND TECHNOLOGIES

Most of the work related to machine learning has been done in PYTHON using GOOGLE COLAB. Although we have used EXCEL for data cleaning and merging. For designing the application, we have used IntelliJ IDE, JAVASCRIPT, HTML,CSS and CHROME DEVELOPER TOOLS to build the extension. For our report we have used LaTeX. For code management we have used GITHUB.

7 CONCLUSION

Malicious URL detection plays a critical role for many cybersecurity applications, and clearly machine learning approaches are a promising direction. Therefore, there is an urgent need for detection methods to evolve and recognize the ever more sophisticated methods being used by attackers to target victims[6]. In this project, we conducted a comprehensive and systematic survey on Malicious URL Detection using machine learning techniques. We focused on feature engineering and building different types of classifiers. We evaluated the models' performance using a number of matrices including accuracy, precision, area under curve and false negative rate. The final features used in the dataset for experimentation contained a mix of lexical, content-based, and host-based features which were selected by performing three features extraction techniques. The results showed that XGBoost obtained the highest accuracy. Future work can focus on extracting one content-based feature that sometimes improves accuracy: the top keywords feature of the website content. Some researchers have used the term frequency-inverse document frequency (TF-IDF) technique to extract the keywords[6]. However, this method is not great at predicting mailiciousness [1]. Online approach should be considered to collect live data, which keeps the model up-to-date by training with recent malicious URLs that might use new techniques that are yet to be determined.

8 FINAL REFLECTIONS

The project is important because it helped us understand one of the real world cybersecurity threat in depth. Not to say we built everything correctly. We ran into multiple problems with the data size and computing capabilities. We decided to work with data that we can handle and can be used without crashing our system. Another major issue we faced was when trying to integrate the plugin with our machine learning model. It was to convert python sklearn model code into javascript code. To mitigate that use we tried to change our approach. We converted the model into a pickle file and passed that into a javascript file for the extension.

If we started the project again we would take more time and spend it on feature extraction, so that we would work with atleast 10k records. Another thing we would have added were more features, especially popularity based features like google index, ranking of the webpage, quality of the page to name a few. We could have used the wordcloud's top 5 good and bad words to create new features. These would necessarily indicate if that particular word is found in the domain or not.

Building the full project changed my thinking in terms of design and planning. It is of utmost importance to divide the project into phases. Additional time should be kept for integration and testing as we cannot predict the issues we would run into. When working on individual stages it was just like working on a task. But working on the full project made us realise that even if all your components are working does not mean you have a successful project. A successful project will be one where all the components are **integrated** and working well.

9 REFERENCES

https://pynative.com/python-datetime-format-strftime/: :text=Use%20datetime.strftime(format),hh%3Amm%3Ass%20format.
https://www.w3schools.com/python/ref_requests_response.asp
https://www.davedash.com/tutorial/pyquery/
https://machinelearningmastery.com/calculate-feature-importance-with-python/
https://www.geeksforgeeks.org/how-to-display-the-value-of-each-bar-in-a-bar-chart-using-matplotlib/
https://wisdomml.in/malicious-url-detection-using-machine-learning-in-python/
https://stackoverflow.com/questions/48558007/where-is-dumped-file-in-google-colab
https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663
https://towardsdatascience.com/extracting-feature-vectors-from-url-strings-for-malicious-url-detection-cbafc24737a
https://www.kaggle.com/code/hamzamanssor/detection-malicious-url-using-ml-models
https://www.kaggle.com/code/blakperlz/malicious-web-detection-eda-cnn
https://www.kaggle.com/code/kkhandekar/phishing-url-classification
https://www.kaggle.com/code/br0kej/feature-engineering-and-analysis
https://www.kaggle.com/code/taruntiwarihp/phishing-sites-detector-complete-info
https://pyodide.org/en/stable/usage/loading-packages.html

[1] - Oshingbesan, A., Ekoh, C., Okobi, C., Munezero, A. and Richard, K., 2022. Detection of Malicious Websites Using Machine Learning Techniques. arXiv preprint arXiv:2209.09630.

[2] - Sahoo, D., Liu, C. and Hoi, S.C., 2017. Malicious URL detection using machine learning: A survey. arXiv preprint arXiv:1701.07179.

[3] - Joshi, A., Lloyd, L., Westin, P. and Seethapathy, S., 2019. Using lexical features for malicious URL detection–a machine learning approach. arXiv preprint arXiv:1910.06277.

[4] - Alsaedi, M., Ghaleb, F.A., Saeed, F., Ahmad, J. and Alasli, M., 2022. Cyber threat intelligence-based malicious URL detection model using ensemble learning. Sensors, 22(9), p.3373.

[5] - Do Xuan, C., Nguyen, H.D. and Tisenko, V.N., 2020. Malicious URL detection based on machine learning. International Journal of Advanced Computer Science and Applications, 11(1).

[6] - Aljabri M, Alhaidari F, Mohammad RMA, Samiha Mirza, Alhamed DH, Altamimi HS, Chrouf SMB. An Assessment of Lexical, Network, and Content-Based Features for Detecting Malicious URLs Using Machine Learning and Deep Learning Models. Comput Intell Neurosci. 2022 Aug 25;2022:3241216. doi: 10.1155/2022/3241216. PMID: 36059391; PMCID: PMC9436524.

[7] - Vemuri, P., 2018. Detecting malicious shortened URLs using machine learning.

[8] - Khan, Hafiz Muhammad Junaid (2019): A MACHINE LEARNING BASED WEB SERVICE FOR MALICIOUS URL DETECTION IN A BROWSER. Purdue University Graduate School.

[9] - Nowroozi, E., Mohammadi, M. and Conti, M., 2022. An adversarial attack analysis on malicious advertisement url detection framework. IEEE Transactions on Network and Service Management.