

A Review III Report on
URL feature engineering and prediction of
maliciousURLs using ML models

Submitted by

Panjam Goutham Reddy – 19BCI0027

Submitted to

Dr. ANNAPURNA JONNALAGADDA,

Associate Professor Grade 2, Scope

Artificial Intelligence (CSE3013)

Slot : B1 + TB1

School of Computer Science and Engineering



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Problem Statement

As the technology has been becoming cheaper and faster, attackers are creating malicious websites quickly and stealing the user's privacy. This lead all the internet users to feel scared before clicking on any links. To be safe, we must check whether the link is malicious or not. A proper model or framework is not properly standardized or framed for this purpose. The main purpose is to put protection to a next level by analyzing the URL before the URL is actually used for browsing. This makes us very secure. AI has many features to be used with and this is one among the real time use cases.

Introduction

i. Motivation

Currently, the risk of network information insecurity is increasing rapidly in number and level of danger. The methods mostly used by hackers today is to attack end-to-end technology and exploit human vulnerabilities. These techniques include social engineering, phishing, pharming, etc. One of the steps in conducting these attacks is to deceive users with malicious Uniform Resource Locators (URLs). As a results, malicious URL detection is of great interest nowadays. So, this became the motivation.

ii. Significance of the Problem

- The most common type of social engineering and cyber-attack is phishing. The phisher uses similar attacks to prey on unsuspecting internet users, fooling them into disclosing private information to utilize it fraudulently.
- Users should be aware of phishing websites to prevent being phished. So, some methodologies can be to maintain a blacklist of phishing websites, which necessitates knowledge of the phishing website. We can use machine learning and deep neural network technologies to detect them in their early stages. These are currently the technology stack in demand as we all are being dependent on them. The machine learning-based technique has been shown to be more successful than the other two ways. Even yet, online consumers are still tricked into disclosing important information on phishing websites. So, a machine learning model with at most accuracy of prediction must be chosen for the real-time applications.

- The main purpose is to put protection to a next level by analyzing the URL before the URL is actually used for browsing. This makes us very secure. AI has many features to be used with and this is one among the real time use cases.

iii. Scope and Applications

The proposed work will create a prediction model for URLs as a classification task using supervised machine learning techniques and data feature extraction. I want to develop a system that can return whether the passed URL is safe or malicious based on learning model used. The results along with code snippets will be discussed through out the report along with how we have stepped into this concept with the help of various research papers that we have studies on. The datasets we used come from various legit sources that include both the benign and as well as malicious URLs.

Security is strengthened. The PKL file (in which the model store after the model training and we can use this for further usage) that contains the final model after comparisons of different models can be used in real time applications easily. In this way, the use case justifies the application requirement.

Literature Survey

a. Existing models

i) Malicious URL Detection : A Comparative Study

The author concluded that the analysis helps to establish that malicious URL detection is possible by training a model using a database with selected features and using it to predict new phishing attacks.

The author compares the results of the multiple machine learning classification techniques such as Logistic Regression (LR), Stochastic Gradient Descent (SGD), Random Forest (RF), Support Vector Machine (SVM), Naive Bayes (NB), K- Nearest Neighbors (KNN), and Decision Tree (DT). The dataset by author was taken from OpenPhish website. [1]

ii) Malicious URL Detection based on Machine Learning.

Feature extraction concept was introduced by the author. Some features and attributes were mentioned, and effectiveness of the proposed extracted attributes was discussed. The author proposes a malicious URL detection method using machine learning techniques based on our proposed URL behaviors and attributes. The author also mentioned some other tools for analyzing the site, but we are not aware of what kind of analysis the tools use. So, this using ML and feature engineering gets us to be at better step than the previous reference. [2]

iii) Feature Engineering Framework to detect Phishing Websites using URL Analysis.

The proposed work has got divided into five sections: work on the pre-processing phase, finding the relation between the features of the dataset, automatic selection of number of features using Extra Tree Classifier, comparison of the various ensemble algorithm and finally generates the best features for URL analysis.

This proposed model of feature engineering trained the dataset in split ratio of 80% and 20%. They tried on 112 features of a URL. The proposed developed algorithm extreme Gradient finds the important features among all the existing 112 features. [3]

b. Gaps identified / Issues in the existing models

The research might have some gaps as this finding [3] has discussed that they have extracted over 112 features successfully, but I were unable to get any pseudocode or any reference to implement for our models.

And, the findings haven't discussed about the concepts of Deep learning and multilayer perceptron, as these are also now rising along with the AI/ML models. The feature extraction process is time consuming as some of the features require to send requests to the databases present on the internet, to get some details about the website or domain, like the rank of website to determine the traffic, the domain age and expiry date, and many others. [1][2]

Implementation

a. Architecture / Frame Work

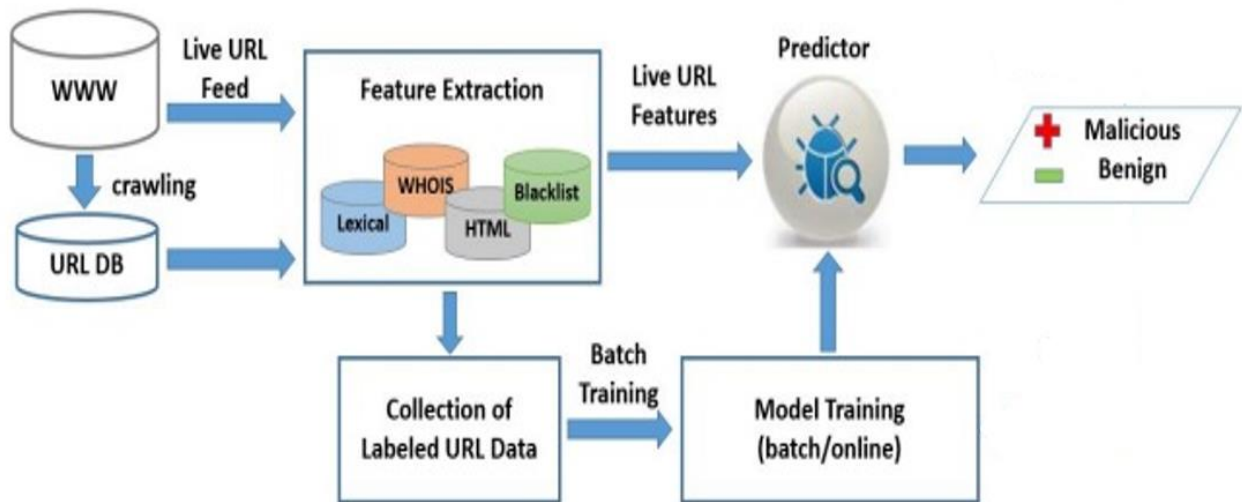


Fig 1: A general processing framework for maliciousURLs using ML models

b. Algorithms

This is a supervised machine learning task. We have classification and regression types. This data set comes under classification problem, as we have to predict whether the test case is 0 or 1. The supervised machine learning models (classification) to be used for training include:

- Decision Tree
- Random Forest
- Multilayer Perceptrons
- XGBoost
- Support Vector Machines
- Using Convolution Neural Networks

Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

```
from sklearn.tree import
DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth = 5)
tree.fit(X_train, y_train)

y_test_tree =
tree.predict(X_test) y_train_tree
= tree.predict(X_train)

acc_train_tree =
accuracy_score(y_train,y_train_tree)
acc_test_tree =
accuracy_score(y_test,y_test_tree)
```

Random Forest classifier

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. They are very powerful, often work well without heavy tuning of the parameters, and don't require scaling of the data. It takes the average of subsets of trees formed to improve the predictive accuracy of that dataset.

```
from sklearn.ensemble import
RandomForestClassifier
forest = RandomForestClassifier(max_depth=5)
forest.fit(X_train, y_train)

y_test_forest = forest.predict(X_test)
y_train_forest = forest.predict(X_train)

acc_train_forest =
accuracy_score(y_train,y_train_forest)
acc_test_forest =
accuracy_score(y_test,y_test_forest)
```

Multilayer preceptron (Deep Learning)

Multilayer perceptron (MLPs) is also known as just neural networks. Multilayer perceptron can be applied for both classification and regression problems. A multi-layer perception is a neural network that has multiple layers. To create a neural network, we combine neurons together so that the outputs of some neurons are inputs of other neurons.

```
from sklearn.neural_network import
MLPClassifier mlp =
MLPClassifier(alpha=0.001,
hidden_layer_sizes=([100,100,100]),
max_iter=500)

mlp.fit(X_train, y_train) y_test_mlp =
mlp.predict(X_test) y_train_mlp =
mlp.predict(X_train)

acc_train_mlp =
accuracy_score(y_train,y_train_mlp)
acc_test_mlp =
accuracy_score(y_test,y_test_mlp)
```

XGBoost Classifier

XGBoost is one of the most popular machine learning algorithms these days. XGBoost stands for eXtreme Gradient Boosting. XGBoost is an open-source software library which provides a regularizing gradient boosting framework. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. It yields superior results using fewer computing resources in the shortest amount of time.

```
from xgboost import XGBClassifier

xgb =
XGBClassifier(learning_rate=0.4,max_depth=7)
xgb.fit(X_train, y_train)

y_test_xgb = xgb.predict(X_test)
y_train_xgb =
xgb.predict(X_train)

acc_train_xgb =
accuracy_score(y_train,y_train_xgb)
acc_test_xgb =
accuracy_score(y_test,y_test_xgb)
```

Support Vector Machines

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.

```
from sklearn.svm import SVC

svm = SVC(kernel='linear', C=1.0,
random_state=12)svm.fit(X_train, y_train)

y_test_svm = svm.predict(X_test)
y_train_svm =
svm.predict(X_train)

acc_train_svm =
accuracy_score(y_train,y_train_svm)
acc_test_svm =
accuracy_score(y_test,y_test_svm)
```

Convolution Neural Network

Deep Learning – which has emerged as an effective tool for analyzing big data – uses complex algorithms and artificial neural networks to train machines/computers so that they can learn from experience, classify and recognize data/images just like a human brain does. Within Deep Learning, a Convolutional Neural Network or CNN is a type of artificial neural network, which is widely used for image/object recognition and classification. Deep Learning thus recognizes objects in an image by using a CNN. CNNs are playing a major role in diverse tasks/functions like image processing problems, computer vision tasks like localization and segmentation, video analysis, to recognize obstacles in self-driving cars, as well as speech recognition in natural language processing. As CNNs are playing a significant role in these fast-growing and emerging areas, they are very popular in Deep Learning.

```
import numpy as np

from os import listdir
```



```
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(64, input_shape=(16,), activation =
'relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(64, kernel_initializer = 'he_uniform',
kernel_regularizer = None, kernel_constraint = 'MaxNorm', activation =
'relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(128, kernel_initializer = 'he_uniform',
kernel_regularizer = None, kernel_constraint = 'MaxNorm', activation =
'relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(64, kernel_initializer = 'he_uniform',
kernel_regularizer = None, kernel_constraint = 'MaxNorm', activation =
'relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(32, kernel_initializer = 'he_uniform',
kernel_regularizer = None, kernel_constraint = 'MaxNorm', activation =
'relu'))
model.add(tf.keras.layers.Dropout(0.45))
model.add(tf.keras.layers.Dense(16, kernel_initializer = 'he_uniform',
kernel_regularizer = None, kernel_constraint = 'MaxNorm', activation =
'relu'))
model.add(tf.keras.layers.Dropout(0.45))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

c. Complexity Analysis

The complexity analysis of Decision Tree

Training Time Complexity= $O(n \cdot \log(n) \cdot d)$

n = number of points in the Training set

d =dimensionality of the data

Run-time Complexity= $O(\text{maximum depth of the tree})$

We use Decision Tree when we have large data with low dimensionality.

The complexity analysis of Random Forest

Training Time Complexity= $O(n \cdot \log(n) \cdot d \cdot k)$

k =number of Decision Trees

When we have a large number of data with reasonable features. Then we can use multi-core to parallelize our model to train different Decision Trees.

Run-time Complexity= $O(\text{depth of tree} \cdot k)$

Space Complexity= $O(\text{depth of tree} \cdot k)$

Random Forest is comparatively faster than other algorithms.

The complexity analysis of Support Vector Machines

Training Time Complexity= $O(n^2)$

if n is large, avoid using SVM.

Run-time Complexity= $O(k \cdot d)$

K = number of Support Vectors, d =dimensionality of the data

The complexity analysis of XGBoost Classifier

Training with XGBoost takes $O(tdx \log n)$, where t is the number of trees, d is the height of the trees, and x is the number of non-missing entries in the training data. Prediction for a new sample takes $O(td)$.

The complexity analysis of Multilayer Perceptrons

Given a network with n neurons, this step would be in $O(n)$. Given the fact, that the number of neurons n for a given problem can be regarded as a constant, the overall complexity of $O(n^2)$ equals $O(1)$. This depends in the structure and number of layers in a neural network.

The complexity analysis of Convolution Neural Network

For $k \rightarrow j$, we have the time complexity $O(kt + klt + ktj + kj) = O(k * t(1 + j))$. which is the same as the feedforward pass algorithm. Since they are the same, the total time complexity for one epoch will be $O(t * (ij + jk + kl))$. This time complexity is then multiplied by the number of iterations (epochs).

d. Program (code)

<https://github.com/GouthamReddy-Panjam/URL-feature-engineering-and-prediction-of-malicious-URLs-using-ML-models>

e. Approach

Below mentioned are the steps involved in the completion of this project:

1. Extraction of features from URLs.
 - Gather data from open-source platforms containing phishing and trustworthy websites.
 - Create a program to extract the necessary characteristics from the URL database.
 - Using Exploratory Data Analysis (EDA) and data preprocessing methods analyze and preprocess the dataset.
2. Training with current URL datasets
 - Separate the dataset into two parts: training and testing.
 - Apply selected machine learning and deep neural network techniques to the dataset, such as SVM, Random Forest, Decision tree, Multilayer perceptron and XGBoost.
3. Accuracy and performance measurement.
 - Develop code to present the assessment result while taking accuracy metrics into account.
 - Compare the outcomes produced for trained models and determine which is better, by visualizing and accuracy comparison.
4. Detection of new unknown URL.
 - Ask user for new URL
 - Repeat the steps 1,2,3 and get the prediction whether it is malicious or not.

f. Data Collection

- Legitimate URLs are obtained from the University of New Brunswick

dataset, <https://www.unb.ca/cic/datasets/url-2016.html>.

- 1000 URLs are chosen at random from the list.
- Phishing URLs are gathered through the opensource service Phish Tank. The site provides a collection of phishing URLs in various forms such as csv, json, and so on. This is updated every hour. https://www.phishtank.com/developer_info.php
- 1000 URLs are chosen at random from the acquired collection

```
#Collecting 5,000 Phishing URLs randomly
phishurl = data0.sample(n = 5000, random_state = 12).copy()
phishurl = phishurl.reset_index(drop=True)
phishurl.head()
```

	phish_id	url	phish_detail_url	submission_time	verified	verification_time	online	target
0	7046111	https://3j124.csb.app	http://www.phishtank.com/phish_detail.php?phis...	2021-03-26T17:23:02+00:00	yes	2021-05-07T07:49:12+00:00	yes	Microsoft
1	7486353	https://folder909303-3u7878d78j893ik3.firebaseio...	http://www.phishtank.com/phish_detail.php?phis...	2022-04-12T08:37:42+00:00	yes	2022-04-12T08:42:46+00:00	yes	Microsoft
2	7421864	https://waqassupplies.com/admin/api.html	http://www.phishtank.com/phish_detail.php?phis...	2022-01-24T17:35:03+00:00	yes	2022-01-24T18:24:54+00:00	yes	Internal Revenue Service
3	7476607	https://knightfallfc.com/user1/linkedin/linkne...	http://www.phishtank.com/phish_detail.php?phis...	2022-04-01T04:56:42+00:00	yes	2022-04-01T05:03:02+00:00	yes	LinkedIn
4	7481001	https://www.mpdwe2fe.top/	http://www.phishtank.com/phish_detail.php?phis...	2022-04-06T17:14:45+00:00	yes	2022-04-06T17:22:27+00:00	yes	Other

Fig 2: Malicious URL dataset

```
#Collecting 5,000 Legitimate URLs randomly
legiurl = data1.sample(n = 5000, random_state = 12).copy()
legiurl = legiurl.reset_index(drop=True)
legiurl.head()
```

	URLs
0	http://graphicriver.net/search?date=this-month...
1	http://ecnavi.jp/redirect/?url=http://www.cros...
2	https://hubpages.com/signin?explain=follow+Hub...
3	http://extratorrent.cc/torrent/4190536/AOMEI+B...
4	http://icicibank.com/Personal-Banking/offers/o...

Fig 3: Legit URLs dataset.

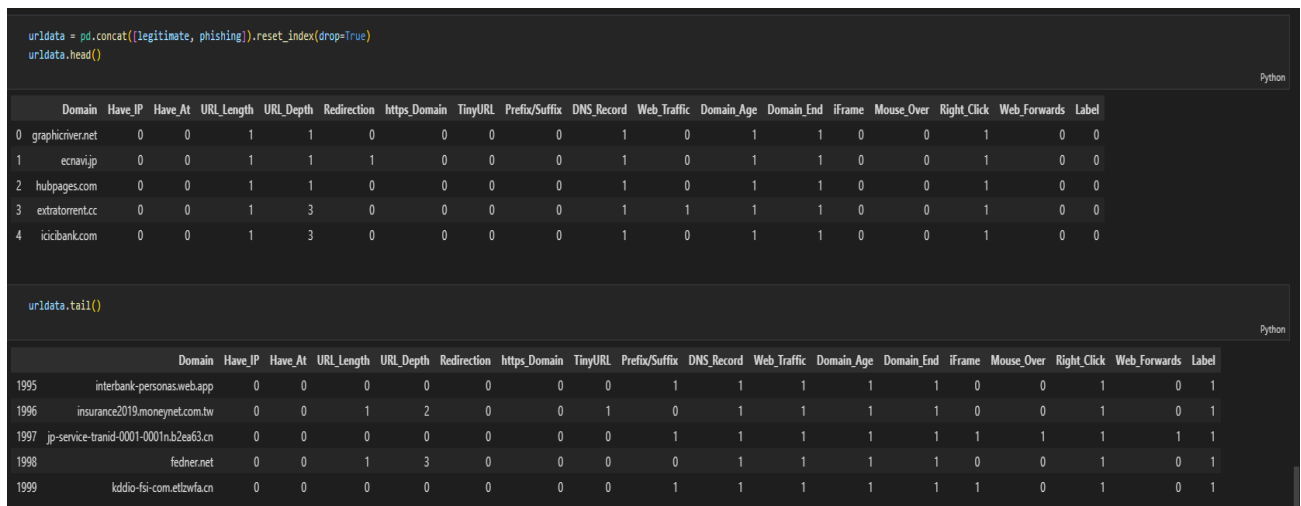
g. Combining Data

We then combine both the legit and malicious url features data and mix them both randomly for equal balance and no bias while learning. We have formed two dataframes of legitimate & phishing URL features. Now, we will combine them

toa single dataframe and export the data to csv file for the Machine Learning training.

```
urldata =
pd.concat([legitimate,
phishing]).reset_index(drop
=True)urldata.head()

urldata.to_csv('./data/urldata.csv', index=False)
```



The screenshot shows a Jupyter Notebook interface with two code cells. The first cell concatenates 'legitimate' and 'phishing' dataframes and displays the first five rows. The second cell displays the last five rows of the resulting dataframe.

urldata.head()

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iframe	Mouse_Over	Right_Click	Web_Forwards	Label
0	graphicriver.net	0	0	1	1	0	0	0	0	1	0	1	1	0	0	1	0	0
1	ecnavi.jp	0	0	1	1	1	0	0	0	1	0	1	1	0	0	1	0	0
2	huppages.com	0	0	1	1	0	0	0	0	1	0	1	1	0	0	1	0	0
3	extratorrent.cc	0	0	1	3	0	0	0	0	1	1	1	1	0	0	1	0	0
4	icicibank.com	0	0	1	3	0	0	0	0	1	0	1	1	0	0	1	0	0

urldata.tail()

	Domain	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iframe	Mouse_Over	Right_Click	Web_Forwards	Label
1995	interbank-personas.web.app	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0
1996	insurance2019.money.net.com.tw	0	0	1	2	0	0	1	0	1	1	1	1	1	0	0	1	0
1997	jp-service-tranid-0001-0001.nb2ea63.cn	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1998	fedner.net	0	0	1	3	0	0	0	0	1	1	1	1	1	0	0	1	0
1999	lkdio-fsi-com.ettzwfa.cn	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	0

Fig 4: Combined data features.

h. Data Processing

Except 'Domain' & 'URL_Depth' columns, all other data are either 0 or 1. The Domain column doesn't have any significance to the machine learning model training. So, dropping the Domain column from the dataset. In the feature extraction file, there has been no shuffling before or after concatenation of legit and phishing URLs. This resulted in top 5000 rows of legitimate url data & bottom 5000 of phishing url data. We need to shuffle to evenly distribute them while splitting into train and test data, as this will remove the case of overfitting while model training.

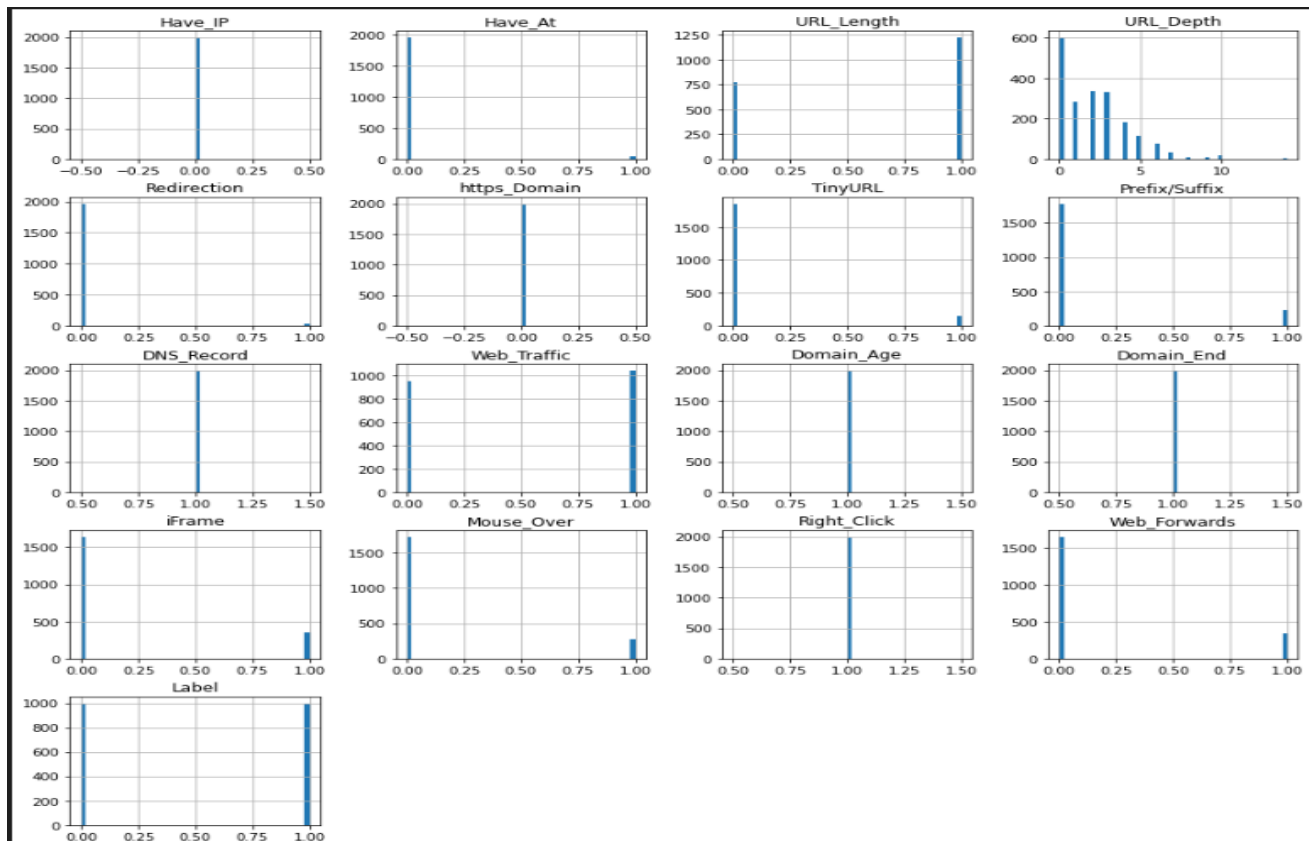


Fig 5: Data distributions plotting for visualization

i. Model Testing

Training and testing datasets are separated before training phase, and after the training phase, the accuracy score is generated based on the labels that are given by us. The prediction is compared to the given label as whether the URL is malicious or not, and based on those, we get the fraction terms, with which we can get prediction percentages.

```
#creating dataframe
results = pd.DataFrame({ 'ML Model': ML_Model,
                          'Train Accuracy': acc_train,
                          'Test Accuracy': acc_test})
results
```

	ML Model	Train Accuracy	Test Accuracy
0	Decision Tree	0.924	0.922
1	Random Forest	0.926	0.922
2	Multilayer Perceptrons	0.937	0.930
3	XGBoost	0.938	0.935
4	SVM	0.918	0.925
5	Neural Network	0.931	0.920

Fig 6 : Training and testing accuracy decimals of each model used.

j. Best Model for Applications

So, by sorting the dataframe based on descending order of test accuracies, we can observe that for 2000 (1000+1000) samples given for training, XGBoost performs well in terms of classifying whether the given URL is malicious or not based on its features passed as input vector.

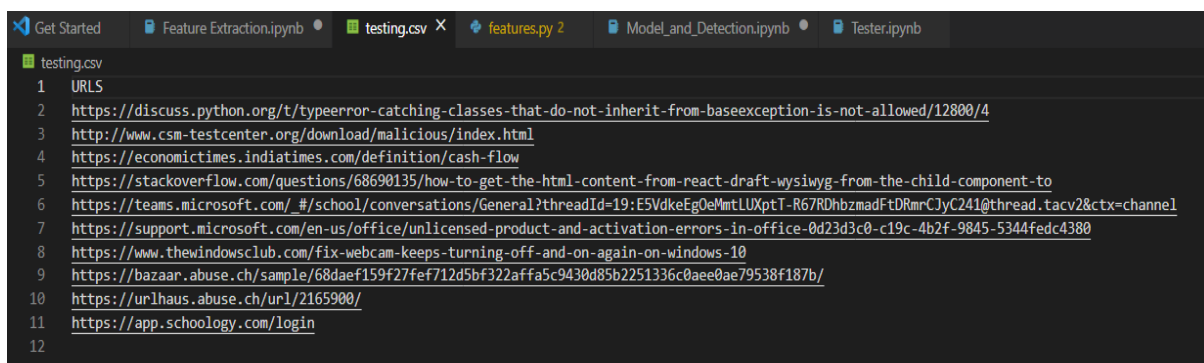
XGBoost Classifier works well with this train dataset.

```
# save XGBoost model to file
import pickle
pickle.dump(xgb, open("XGBoostClassifier.pickle.dat", "wb"))

# load model from file
loaded_model = pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))
loaded_model

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
               colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
               early_stopping_rounds=None, enable_categorical=False,
               eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
               importance_type=None, interaction_constraints='',
               learning_rate=0.4, max_bin=256, max_cat_to_onehot=4,
               max_delta_step=0, max_depth=7, max_leaves=0, min_child_weight=1,
               missing=nan, monotone_constraints='()', n_estimators=100,
               n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
               reg_alpha=0, reg_lambda=1, ...)
```

Fig 7: Saving the model as pickle file for further use in applications that use this prediction feature.



```
testing.csv
1  URLs
2  https://discuss.python.org/t/typeerror-catching-classes-that-do-not-inherit-from-baseexception-is-not-allowed/12800/4
3  http://www.csm-testcenter.org/download/malicious/index.html
4  https://economictimes.indiatimes.com/definition/cash-flow
5  https://stackoverflow.com/questions/68690135/how-to-get-the-html-content-from-react-draft-wysiwyg-from-the-child-component-to
6  https://teams.microsoft.com/_#/school/conversations/General?threadId=19:E5VdkeEg0eMmtLUXptI-R67RDhzbmadFdRmrCJyC241@thread.tacv2&ctx=channel
7  https://support.microsoft.com/en-us/office/unlicensed-product-and-activation-errors-in-office-0d23d3c0-c19c-4b2f-9845-5344fedc4380
8  https://www.thewindowsclub.com/fix-webcam-keeps-turning-off-and-on-again-on-windows-10
9  https://bazaar.abuse.ch/sample/68daef159f27fef712d5bf322affa5c9430d85b2251336c0aee0ae79538f187b/
10 https://urlhaus.abuse.ch/url/2165900/
11 https://app.schoology.com/login
12
```

Fig 8: Testing purpose URLs given by us to the pickle model


```
import pickle
loaded_model = pickle.load(open("XGBoostClassifier.pickle.dat", "rb"))

#importing required packages for this module
import pandas as pd
testurldata = pd.read_csv("testing.csv")
testurldata.head(10)
```

	URLS
0	https://discuss.python.org/t/typeerror-catchin...
1	http://www.csm-testcenter.org/download/malicio...
2	https://economictimes.indiatimes.com/definitio...
3	https://stackoverflow.com/questions/68690135/h...
4	https://teams.microsoft.com/_#/school/conversa...
5	https://support.microsoft.com/en-us/office/unl...
6	https://www.thewindowsclub.com/fix-webcam-keep...
7	https://bazaar.abuse.ch/sample/68daef159f27fef...
8	https://urlhaus.abuse.ch/url/2165900/
9	https://app.schoology.com/login

Fig 9: Loading dataset for the testing purpose by extraction

	Have_IP	Have_At	URL_Length	URL_Depth	Redirection	https_Domain	TinyURL	Prefix/Suffix	DNS_Record	Web_Traffic	Domain_Age	Domain_End	iFrame	Mouse_Over	Right_Click	Web_Forwards
0	0	0	1	4	0	0	0	0	1	1	1	1	0	0	1	0
1	0	0	1	3	0	0	0	1	1	1	1	1	0	0	1	0
2	0	0	1	2	0	0	0	0	1	1	1	1	0	0	1	0
3	0	0	1	3	0	0	0	0	1	1	1	1	0	0	1	0
4	0	1	1	1	0	0	1	0	1	1	1	1	0	0	1	0

Fig 10: Test data feature extraction in the form of dataframe.

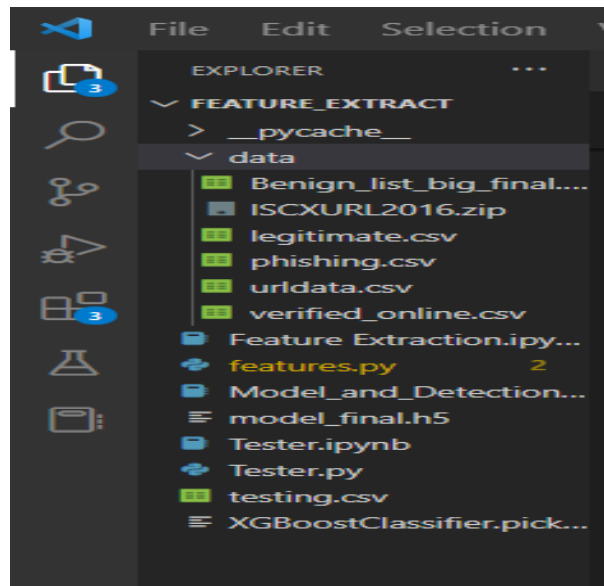


Fig 11: Project directory after the whole execution of project

Result analysis (analytics and visualizations)

```
#Sorting the dataframe on accuracy
results.sort_values(by=['Test Accuracy', 'Train Accuracy'], ascending=False)
```

	ML Model	Train Accuracy	Test Accuracy
3	XGBoost	0.938	0.935
2	Multilayer Perceptrons	0.937	0.930
4	SVM	0.918	0.925
1	Random Forest	0.926	0.922
0	Decision Tree	0.924	0.922
5	Neural Network	0.931	0.920

XGBoost Classifier works well with this train dataset.

Fig 12: Sorted test accuracies

The predictions are somewhat accurate and as expected. This is because the size of training data is 5 to 10 times reduced. We only have taken 1000 samples of each legit and malicious URLs and extract and combine and train the models. And one more thing to observe is that the phishing/malicious websites that are present in dataset we downloaded are all short URLs having less length when compared to legit URLs. And, we have assumed that the URL having more length is malicious and considered as label “1” because long URL will make the malicious part out of the URL search box in browser, that makes the user feel that it is ok. So, there is still some developments that can be made for our model and extraction logics to make the model think very accurately than now.

In the output, the “1” stands for malicious and “0” stands for legitimate URL.

```
y_pred = loaded_model.predict(test_urls_features)
print("The output is ",y_pred)
```

✓ 0.1s

The output is [0 1 0 0 1 1 0 0 1 1]

Fig 13: Predictions of test data separately

Conclusion

I conclude that Machine learning and Deep learning help us to study all the previously experienced data, and with data processing concepts, we can extract the features and remove unnecessary data to pass dataset as data frames into the models for predictions. Greater the test prediction, better the model to convert into pickle file and store in hard drive and use this in real time applications. There are also many ML models that can be used for creating better accurate models and the training data can also be improved. Working on this project is very knowledgeable and worth the effort.

Future work

I planned to develop a framework using this approach and deploy it for a large scale real world test. We will also investigate the effectiveness of online algorithms as they have been found to outperform traditional batch algorithms in problem similar to ours. We believe that by looking into the contents of web pages, we can further improve false positives and negatives. We will also investigating this matter as well.

Various famous deep learning architecture proposed which are improved the accuracy upto great extend. We will tried to implement attention network which gives special attention on specific features of URLs. Suppose, in transaction websites we must have to pay special attention to the protocol of URL and domain name of URL instead of other features etc. Attention network improve result in most of the case. We are working it in our case but we did not reached upto the standard, but we are unable to find the reason, may be some fine tuning are required which we did not done. Complexity of attention network is high. Since Deep learning model learns weights and create a model, everything inside the model acts like a black box. Its was proven that in deep learning as we increase the complexity of model explainabilty going to reduce.

References

- [1] Shantanu, B. Janet and R. Joshua Arul Kumar, "Malicious URL Detection: A Comparative Study," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 1147-1151, doi: 10.1109/ICAIS50930.2021.9396014.

- [2] Xuan, Cho & Dinh, Hoa & Victor, Tisenko. (2020). Malicious URL Detection based on Machine Learning. International Journal of Advanced Computer Science and Applications. 11. 10.14569/IJACSA.2020.0110119.

- [3] Goud, N. & Mathur, Anjali. (2021). Feature Engineering Framework to detect Phishing Websites using URL Analysis. International Journal of Advanced Computer Science and Applications. 12. 10.14569/IJACSA.2021.0120733.

Datasets :

1. <https://www.unb.ca/cic/datasets/url-2016.html>
2. <https://archive.ics.uci.edu/ml/datasets/phishing+websites>
3. https://www.phishtank.com/developer_info.php

Github link for code

<https://github.com/GouthamReddy-Panjam/URL-feature-engineering-and-prediction-of-malicious-URLs-using-ML-models>