M. Tech. 2nd Semester Mini-Project Report

On

Phishing Detection Using Machine Learning Techniques

Pratik Joshi

(232IS014)

Guide

Dr. Radhika

Department of Computer science and Engineering,

NITK, Surathkal

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL, MANGALORE - 575025

April, 2024

DECLARATION

I hereby declare that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Phishing Detection Using Machine Learning Techniques** which is being submitted to the National Institute of Technology Karnataka Surathkal, in partial fulfilment of the requirements for the award of the Degree of **Master of Technology in Computer Science and Engineering** in the department of **Computer Science and Engineering**, is a bonafide report of the work carried out by me. The material contained in this Report has not been submitted to any University or Institution for the award of any degree.

Pratik Joshi
232IS014
Department of Computer Science and Engineering
NITK, Surathkal

Place: NITK, Surathkal.
Date: 09/04/2024

CERTIFICATE

This is to certify that the M. Tech. 2nd Semester **Mini-Project** Report entitled **Phishing Detection Using Machine Learning Techniques** submitted by **Pratik Joshi**, (Roll Number: 232IS014) as the record of the work carried out by him/her, is accepted as the M. Tech. 2nd Semester Mini-Project Report submission in partial fulfilment of the requirements for the award of degree of Master of **Master of Technology in Computer Science and Engineering** in the Department of **Computer Science and Engineering**.

Guide
Dr. Radhika
Department of Computer Science and Engineering
NITK, Surathkal

Chairman - DPGC
Dr.Manu Basavaraju
Department of Computer Science and Engineering
NITK, Surathkal

Abstract

The Internet has become an indispensable part of our life, However, It also has provided opportunities to anonymously perform malicious activities like Phishing. Phishers try to deceive their victims by social engineering or creating mock- up websites to steal information such as account ID, username, password from individuals and organizations. Although many methods have been proposed to detect phishing websites, Phishers have evolved their methods to escape from these detection methods. One of the most successful methods for detecting these malicious activities is Machine Learning. This is because most Phishing attacks have some common characteristics which can be identified by machine learning methods. In this paper, we compared the results of multiple machine learning methods for predicting phishing website

Table of contents

List of Figure

1.Introduction

Phishing is a kind of Cybercrime trying to obtain important or confidential information from users which is usually carried out by creating a counterfeit website that mimics a legitimate website. Phishing attacks employ a variety of techniques such as link manipulation, filter evasion, website forgery, covert redirect, and social engineering. The most common approach is to set up a spoofing web page that imitates a legitimate website. These type of attacks were top concerns in the latest 2018 Internet Crime Report, issued by the U.S. Federal Bureau of Investigations Internet Crime Complaint Center (IC3). The statistics gathered by the FBIs IC3 for 2018 showed that internet-based theft, fraud, and exploitation remain pervasive and were responsible for a staggering $2.7 billion in financial losses in 2018. In that year, the IC3 received 20,373 complaints against business email compromise (BEC) and email account compromise (EAC), with losses of more than $1.2 billion [1]. The report notes that the number of these sophisticated attacks have grown increasingly in recent years.

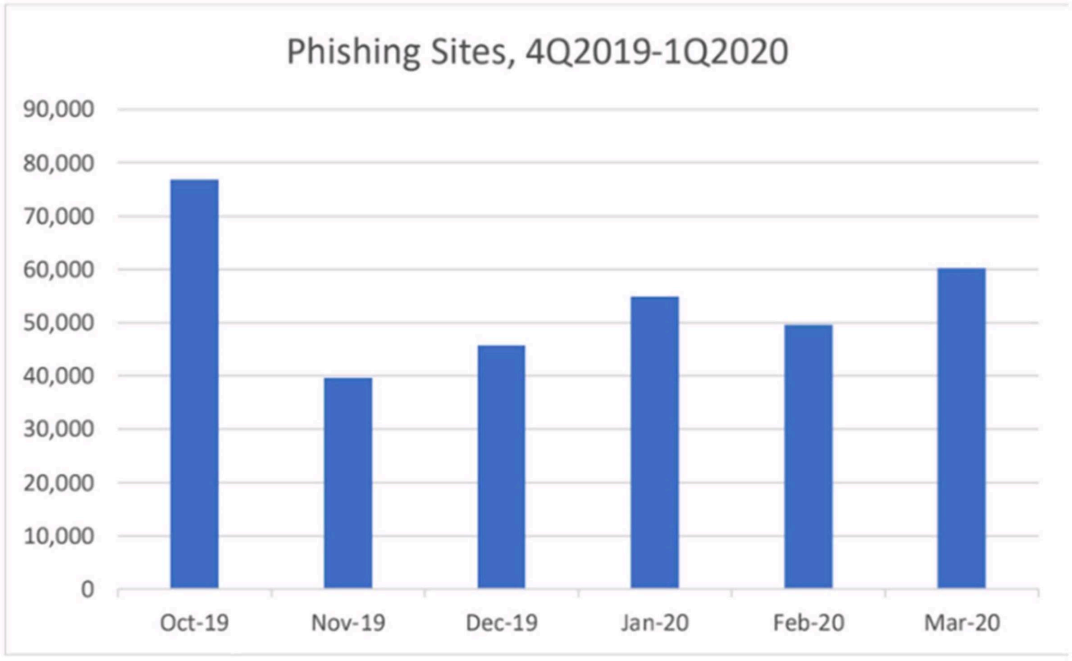

Fig. 1. Total number of phishing websites detected by APWG [2

Anti-Phishing Working Group(APWG) emphasizes that phishing attacks have grown in recent years, Figure 1 illustrates the total number of phishing sites detected by APWG in the first quarter of 2020 and the last quarter of 2019. This number has a gradual growth rising from 162,155 in the last quarter of 2019 to 165,772 cases in the first quarter of 2020. Phishing has caused severe damages to many organizations and the global economy, in the fourth quarter of 2019, APWG member OpSec Security found that SaaS and webmail sites remained the most frequent targets of phishing attacks. Phishers continue to harvest credentials from these targets by operating BEC and subsequently gain access to corporate SaaS accounts [2]. Many approaches have been used to filter out phishing websites. Each of these methods is applicable on different stages of attack flow, for example, network-level protection, authenti- cation, client-side tool, user education, server-side filters, and classifiers. Although there are some unique features in every type of phishing attack, most of these attacks depict some similarities and patterns. Since machine learning methods proved to be a powerful tool for detecting patterns in data, these methods have made it possible to detect some of the common phishing traits, therefore, recognizing phishing web- sites. In this paper, we provide a comparative and analytical evaluation of different machine learning methods on detecting the phishing websites. The machine learning methods that we studied are Decision Tree, Ada-Boost, Artificial Neural Networks, and XGBoost.

2. Dataset

One of the main challenges in our research was the scarcity of phishing dataset. Although many scientific papers about phishing detection have been published, they have not pro- vided the dataset on which they used in their research. Moreover, another factor that hinders finding a desirable dataset is the lack of a standard feature set to record characteristics of a phishing website. The dataset that we used in our research was well researched and benchmarked by some researchers. Fortunately, the accompanying wiki of the dataset comes with a data description document which discusses the data generation strategies taken by the authors of the dataset [23]. For updating our dataset with new phishing websites we have also implemented a code that extracts features of new phishing websites that are provided by the PhishTank website. The dataset contains about 11,000 sample websites, we used 10% of samples in the testing phase. Each website is marked either legitimate or phishing. The features of our dataset are as follows:

1) Having IP Address: If an IP address is used in- stead of the domain name in the URL, such as http://217.102.24.235/sample.html.

2) URL Length: Phishers can use a long URL to hide the doubtful part in the address bar.

3) Shortening Service: Links to the webpage that has a long URL. For example, the URL http://sharif.hud.ac.uk/ can be shortened to bit.ly/1sSEGTB.

4) Having @ Symbol: Using the @ symbol in the URL leads the browser to ignore everything preceding the @ symbol and the real address often follows the @ symbol

5) Double Slash Redirection: The existence of // within the URL which means that the user will be redirected to another website

6) Prefix Suffix: Phishers tend to add prefixes or suffixes separated by (-) to the domain name so that users feel that they are dealing with a legitimate webpage. For example http://www.Confirme-paypal.com.

7) Having Sub Domain: Having subdomain in URL.

8) SSL State: Shows that website use SSL.

9) Domain Registration Length: Based on the fact that a phishing website lives for a short period.

10) Favicon: A favicon is a graphic image (icon) associated with a specific webpage. If the favicon is loaded from a domain other than that shown in the address bar, then the webpage is likely to be considered a Phishing attempt.

11) Using Non-Standard Port: To control intrusions, it is much better to merely open ports that you need. Several firewalls, Proxy and Network Address Translation (NAT) servers will, by default, block all or most of the ports and only open the ones selected

12) HTTPS token: Having deceiving https token in URL. For example, http://https-www-mellat-phish.ir

13) Request URL: Request URL examines whether the external objects contained within a webpage such as images, videos, and sounds are loaded from another domain.

14) URL of Anchor: An anchor is an element defined by the < a > tag. This feature is treated exactly as Request URL.

15) Links In Tags: It is common for legitimate websites to use ¡Meta¿ tags to offer metadata about the HTML document; ¡Script¿ tags to create a client side script; and ¡Link¿ tags to retrieve other web resources.

16) Server Form Handler: If the domain name in SFHs is different from the domain name of the webpage.

17) Submitting Information To E-mail: A phisher might redirect the users information to his email.

18) Abnormal URL: It is extracted from the WHOIS database. For a legitimate website, identity is typically part of its URL.

19) Website Redirect Count: If the redirection is more than four-time.

20) Status Bar Customization: Use JavaScript to show a fake URL in the status bar to users.

21) Disabling Right Click: It is treated exactly as Using onMouseOver to hide the Link.

22) Using Pop-up Window: Showing having popo-up win- dows on the webpage.

23) IFrame: IFrame is an HTML tag used to display an additional webpage into

one that is currently shown.

24) Age of Domain: If the age of the domain is less than a month.

25) DNS Record: Having the DNS record

26) Web Traffic: This feature measures the popularity of the website by determining the number of visitors.

27) Page Rank: Page rank is a value ranging from 0 to 1. PageRank aims to measure how important a webpage is on the Internet.

28) Google Index: This feature examines whether a website is in Googles index or not.

29) Links Pointing To Page: The number of links pointing to the web page.

30) Statistical Report: If the IP belongs to top phishing IPs or not.

3. Various Machine Learning Techniques used

1. Decision Tree

Decision tree classifiers are used as a well-known classifica- tion technique. A decision tree is a flowchart-like tree structure where an internal node represents a feature or attribute, the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition based on the attribute value. It partitions the tree in a recursive manner called recursive partitioning. This particular feature gives the tree classifier a higher resolution to deal with a variety of data sets, whether numerical or categorical data. Also, decision trees are ideal for dealing with nonlinear relationships between attributes and classes. Regularly, an impurity function is determined to assess the quality of the division for each node, and the Gini Variety Index is used as a known criterion for the total performance. In practice, the decision tree is flexible in the sense that it can easily model nonlinear or unconventional relationships. It can interpret the interaction between predictors. It can also be interpreted very well because of its binary structure. However, the decision tree has various drawbacks that tend to overuse data. Besides, updating a decision tree by new samples is difficult.

2. Ada-Boost

From some aspects, Ada-boost is like Random Forest, the Ada-Boost classification like Random Forest groups weak classification models to form a strong classifier. A single model may poorly categorize objects. But if we combine several classifiers by selecting a set of samples in each iteration and assign enough weight to the final vote, it can be good for the overall classification. Trees are created sequentially as weak learners and correcting incorrectly predicted samples by assigning a larger weight to them after each round of prediction. The model is learning from previous errors. The final prediction is the weighted majority vote (or weighted median in case of regression problems). In short Ada-Boost algorithm is repeated by selecting the training set based on the accuracy of the previous training. The weight of each classifier trained in each iteration depends on the accuracy obtained from previous ones [19].

3. XGBoost

XGBoost is a refined and customized version of a Gradient Boosting to provide better performance and speed. The most important factor behind the success of XGBoost is its scalability in all scenarios. The XGBoost runs more than ten times faster than popular solutions on a single machine and scales to billions of examples in distributed or memory-limited set- tings. The scalability of XGBoost is due to several important algorithmic optimizations. These innovations include a novel tree learning algorithm for handling sparse data; a theoretically justified weighted quantile sketch procedure enables handling instance weights in approximate tree learning. Parallel and dis- tributed computing make learning faster which enables quicker model exploration. More importantly, XGBoost exploits out- of-core computation and enables data scientists to process hundreds of millions of examples on a desktop. Finally, it is even more exciting to combine these techniques to make an end-to-end system that scales to even.

4. Artificial Neural Networks

Artificial neural networks (ANNS) are a learning model roughly inspired by biological neural networks. These models are multilayered, each layer containing several processing units called neurons. Each neuron receives its input from its adjacent layers and computes its output with the help of its weight and a non-linear function called the activation function. In feed-forward neural networks like in 3, data flows from the first layer to the last layer. Different layers may perform different transformations on their input. The weights of neurons are set randomly at the start of the training and they are gradually adjusted by the help of the gradient descent method to get close to the optimal solution. The power of neural networks is due to the non-linearity of hidden nodes. As a result, introducing non-linearity in the network is very important so that you can learn complex functions.

4. Evaluation Matrices

For evaluating phishing classification performance we use accuracy(acc) recall(r), precision(p), F1 score, test time, and train time of classifiers. Recall measures the percentage of phishing websites that the model manages to detect (mod- els effectiveness). Precision measures the degree to which the phishing detected websites are indeed phishing (models safety). F1 score is the weighted harmonic mean of precision and recall. Let NL→L be the number of legitimate websites classified as legitimate, NL→P be the number of legitimate websites misclassified as phishing, NP →L be the number of phishing misclassified as legitimate and NP →P be the number of phishing websites classified as phishing. Thus the following equations hold

$$acc = \frac{N_{L\to L} + N_{P\to P}}{N_{L\to L} + N_{L\to P} + N_{P\to L} + N_{P\to P}} \quad (1)$$

$$r = \frac{N_{P\to P}}{N_{P\to L} + N_{P\to P}} \quad (2)$$

$$p = \frac{N_{P\to P}}{N_{L\to P} + N_{P\to P}} \quad (3)$$

$$F1 = \frac{2pr}{p + r} \quad (4)$$

5. Experimental Results

In our experiments, we used 10-fold cross-validation for model performance evaluation. We divided the data set into 10 sub-samples. A sub-sample is used for testing data and the rest is used for training models. Since phishing detection is a classification problem we must use a binary classification model, we consider "-1" as a phishing sample and "1" as a legitimate one. In our study, we used various machine learning models for detecting phishing websites which are Logistic regression, Ada booster, random forest, KNN, neural networks, SVM, Gradient boosting, XGBoost. We evaluate the accuracy, preci- sion, recall, F1 score, training time, and testing time of these models and we used different methods of feature selection and hyperparameters tuning for getting the best results. Table III shows the comparison between accuracy, precision, recall, and F1 score of these models.

classifier	train time(s)	test time(s)	accuracy	recall	precision	F1 score
decision tree	0.021452	0.003737	0.965988	0.971414	0.967681	0.969531
ada booster	0.336519	0.016766	0.936953	0.954362	0.933943	0.944032
neural network	9.088517	0.006925	0.969879	0.978723	0.967605	0.973112
XGBoost	0.506072	0.006237	0.983235	0.981047	0.987235	0.976802

Fig 2 .CLASSIFICATION RESULTS FOR DIFFERENT METHODS

The main advantage of XGBoost is its fast speed compared to other algorithms, such as ANN and SVM, and its regularization parameter that successfully reduces variance. But even aside from the regularization parameter, this algorithm leverages a learning rate and subsamples from the features like random forests, which increases its ability to generalize even further. However, XGBoost is more difficult to understand, visualize, and to tune compared to AdaBoost and Random Forests. There are a multitude of hyperparameters that can be tuned to increase performance.XGBoost is a particularly interesting algorithm when speed as well as high accuracies are of the essence. Nevertheless, more resources in training the model are required because the model tuning needs more time and expertise from the user to achieve meaningful outcomes.

As expected, neural network's training time was consid- erably higher compared to other machine learning models. XGBoost's F1 score was slightly better compared with neural network's. This is due to the fact that our training data size is small. Unlike XGBoost, neural network model is also unable to explain why it have predicted a website as a phishing one. The explainability will help us to specify key features more easily. In the implementation of neural networks we use Adam optimizer and relu activation function in the hidden layer, figure 7 shows the performance of the neural network with a different number of the hidden layer, we get the best performance with 30 hidden layers. We trained our model on 500 epochs with early stopping.

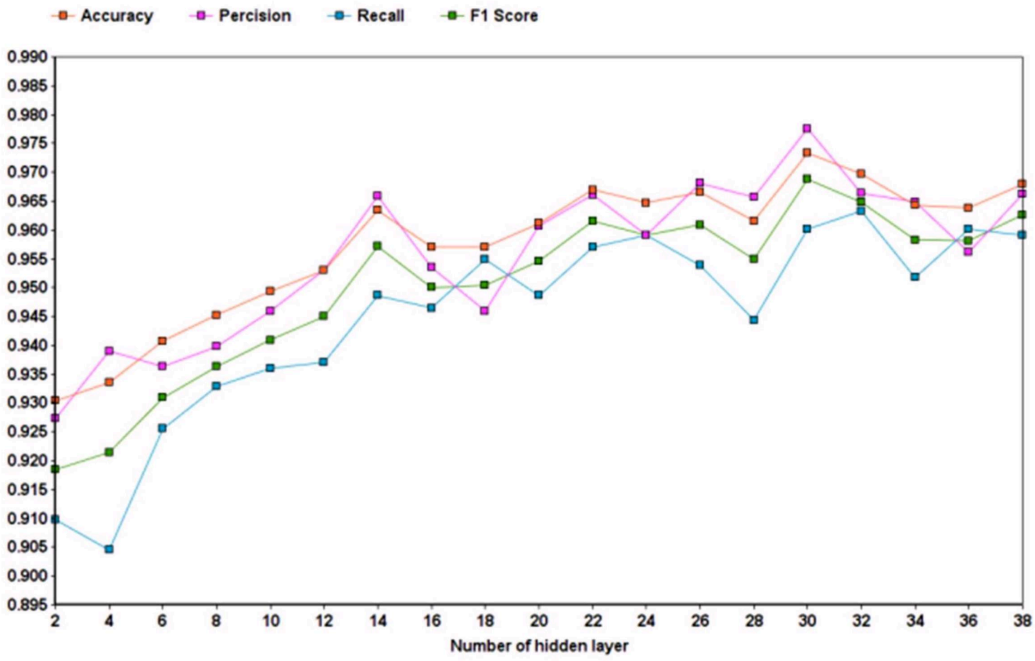

Fig 3. Number of hidden layer

6. Implementation

First of all I have imported all the necessary libraries.

```
!pip3 install catboost
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_validate
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
Collecting catboost
  Downloading catboost-1.2.3-cp310-cp310-manylinux2014_x86_64.whl (98.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.5/98.5 MB 8.9 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.25.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.11.4)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2023.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.1.2)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (8.2.3)
Installing collected packages: catboost
Successfully installed catboost-1.2.3
```

Then I mounted my Google Drive with Colab.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
def mean_score(scoring):
    return {i:j.mean() for i,j in scoring.items()}
```

```
scoring = {'accuracy': 'accuracy',
           'recall': 'recall',
           'precision': 'precision',
           'f1': 'f1'}
fold_count=10
```

`def mean_score(scoring)`: This line defines a function named `mean_score` which takes a parameter named `scoring`.

`return {i:j.mean() for i,j in scoring.items()}`: This line returns a dictionary comprehension that iterates over the key-value pairs of the `scoring` dictionary. For each key-value pair `(i, j)`, it calculates the mean of the values (assuming `j` is an iterable) and returns a new dictionary with the same keys `i` but with values replaced by their means.

`scoring = {'accuracy': 'accuracy', 'recall': 'recall', 'precision': 'precision', 'f1': 'f1'}`: This line initializes a dictionary named `scoring` with keys `'accuracy'`, `'recall'`, `'precision'`, and `'f1'`, each mapped to the corresponding string values.

`fold_count=10`: This line assigns the integer value `10` to the variable `fold_count`.

```
#Dataset 1

# df = pd.read_csv("/content/drive/MyDrive/dataset.csv",index_col=0)

# df = sklearn.utils.shuffle(df)

# #extract features
# X = df.drop("Result",axis=1).values

# #Standardization
# X = preprocessing.scale(X)

# y = df['Result'].values
```

```
#Dataset 2
df=pd.read_csv('/content/drive/MyDrive/dataset_phishing.csv')
classes={'legitimate':0, 'phishing':1}
df['status'] = df['status'].map(classes)
corr_matrix = df.corr()
status_corr = corr_matrix['status']

def feature_selector_correlation(cmatrix, threshold):

    selected_features = []
    feature_score = []
    i=0
    for score in cmatrix:
        if abs(score)>threshold:
            selected_features.append(cmatrix.index[i])
            feature_score.append( ['{:3f}'.format(score)])
        i+=1
    result = list(zip(selected_features,feature_score))
    return result
features_selected = feature_selector_correlation(status_corr, 0.2)
selected_features = [i for (i,j) in features_selected if i != 'status']
X= df[selected_features]
y = df['status']
```

`df=pd.read_csv('/content/drive/MyDrive/dataset_phishing.csv')`: This line reads a CSV file named `'dataset_phishing.csv'` located at the specified path into a DataFrame named `df` using Pandas.

`classes={'legitimate':0, 'phishing':1}`: This line initializes a dictionary named `classes` where the keys are strings `'legitimate'` and `'phishing'`, and the corresponding values are `0` and `1`.

`df['status'] = df['status'].map(classes)`: This line maps the values in the column `'status'` of the DataFrame `df` using the dictionary `classes` and assigns the result back to the `'status'` column.

`corr_matrix = df.corr()`: This line calculates the correlation matrix of the DataFrame `df` and assigns it to the variable `corr_matrix`.

`status_corr = corr_matrix['status']`: This line extracts the correlation values of the column `'status'` from the correlation matrix and assigns it to the variable `status_corr`.

`features_selected = feature_selector_correlation(status_corr, 0.2)`: This line calls the function `feature_selector_correlation` with `status_corr` (the correlation values) and a threshold value of `0.2` to select features based on their correlation.

`selected_features = [i for (i,j) in features_selected if i != 'status']`: This line extracts the feature names from the `features_selected` list, excluding the feature `'status'`.

`X= df[selected_features]`: This line selects the features specified in `selected_features` from the DataFrame `df` and assigns it to the variable `X`.

`y = df['status']`: This line assigns the column `'status'` of the DataFrame `df` to the variable `y`.

```
neural_clf=MLPClassifier(hidden_layer_sizes=(20,),max_iter=500)
cross_val_scores = cross_validate(neural_clf, X, y, cv=fold_count, scoring=scoring)
neural_clf_score = mean_score(cross_val_scores)

fit_time_seconds = neural_clf_score['fit_time']
score_time_seconds = neural_clf_score['score_time']

test_accuracy_percentage = neural_clf_score['test_accuracy'] * 100
test_recall_percentage = neural_clf_score['test_recall'] * 100
test_precision_percentage = neural_clf_score['test_precision'] * 100
test_f1_percentage = neural_clf_score['test_f1'] * 100
print("Fit Time (seconds):", fit_time_seconds)
print("Score Time (seconds):", score_time_seconds)
print("Test Accuracy (%):", test_accuracy_percentage)
print("Test Recall (%):", test_recall_percentage)
print("Test Precision (%):", test_precision_percentage)
```

```
Fit Time (seconds): 1.1730025291442872
Score Time (seconds): 0.010576939582824707
Test Accuracy (%): 91.94225721784778
Test Recall (%): 91.4431802873134
Test Precision (%): 92.70106374773255
```

`neural_clf=MLPClassifier(hidden_layer_sizes=(20,),max_iter=500)`: This line initializes a multilayer perceptron classifier with one hidden layer of 20 neurons and a maximum iteration limit of 500 and assigns it to the variable `neural_clf`.

`cross_val_scores = cross_validate(neural_clf, X, y, cv=fold_count, scoring=scoring)`: This line performs cross-validation using the neural classifier (`neural_clf`) on the features `X` and target `y` with `fold_count` folds, using the scoring dictionary defined earlier.

`neural_clf_score = mean_score(cross_val_scores)`: This line calculates the mean scores across all cross-validation folds using the `mean_score` function defined earlier.

The following lines extract various performance metrics (e.g., fit time, score time, accuracy, recall, precision, F1 score) from the `neural_clf_score` dictionary and print them.

```
import numpy as np

XGB_clf=XGBClassifier()
y_mapped = np.where(y == -1, 0, y)
cross_val_scores = cross_validate(XGB_clf, X, y_mapped, cv=fold_count, scoring=scoring)
XGB_clf_score= mean_score(cross_val_scores)

fit_time_seconds = XGB_clf_score['fit_time']
score_time_seconds = XGB_clf_score['score_time']

test_accuracy_percentage = XGB_clf_score['test_accuracy'] * 100
test_recall_percentage = XGB_clf_score['test_recall'] * 100
test_precision_percentage = XGB_clf_score['test_precision'] * 100
test_f1_percentage = XGB_clf_score['test_f1'] * 100
print("Fit Time (seconds):", fit_time_seconds)
print("Score Time (seconds):", score_time_seconds)
print("Test Accuracy (%):", test_accuracy_percentage)
print("Test Recall (%):", test_recall_percentage)
print("Test Precision (%):", test_precision_percentage)
```

```
Fit Time (seconds): 0.40711753368377684
Score Time (seconds): 0.018477320671081543
Test Accuracy (%): 96.73665791776027
Test Recall (%): 96.83290877248845
Test Precision (%): 96.64984978131612
```

`XGB_clf=XGBClassifier()`: This line initializes an XGBoost classifier with default parameters and assigns it to the variable `XGB_clf`.

`y_mapped = np.where(y == -1, 0, y)`: This line maps the target variable `y` such that where `y` equals `-1`, it is replaced with `0`, otherwise, it remains the same. This

mapping is done to ensure compatibility with XGBoost, as it typically expects binary labels starting from `0`.

`cross_val_scores = cross_validate(XGB_clf, X, y_mapped, cv=fold_count, scoring=scoring)`: This line performs cross-validation using the XGBoost classifier (`XGB_clf`) on the features `X` and mapped target `y_mapped`, with `fold_count` folds, using the scoring dictionary defined earlier.

`XGB_clf_score= mean_score(cross_val_scores)`: This line calculates the mean scores across all cross-validation folds using the `mean_score` function defined earlier.

22-26. These lines extract various performance metrics (fit time, score time, accuracy, recall, precision, F1 score) from the `XGB_clf_score` dictionary and print them.

```
dtree_clf=DecisionTreeClassifier()
cross_val_scores = cross_validate(dtree_clf, X, y, cv=fold_count, scoring=scoring)
dtree_score = mean_score(cross_val_scores)

fit_time_seconds = dtree_score['fit_time']
score_time_seconds = dtree_score['score_time']

test_accuracy_percentage = dtree_score['test_accuracy'] * 100
test_recall_percentage = dtree_score['test_recall'] * 100
test_precision_percentage = dtree_score['test_precision'] * 100
test_f1_percentage = dtree_score['test_f1'] * 100
print("Fit Time (seconds):", fit_time_seconds)
print("Score Time (seconds):", score_time_seconds)
print("Test Accuracy (%):", test_accuracy_percentage)
print("Test Recall (%):", test_recall_percentage)
print("Test Precision (%):", test_precision_percentage)
```

```
Fit Time (seconds): 0.06539645195007324
Score Time (seconds): 0.007317137718200683
Test Accuracy (%): 93.66579177602799
Test Recall (%): 93.92814715932053
Test Precision (%): 93.44766793200782
```

`dtree_clf=DecisionTreeClassifier()`: This line initializes a Decision Tree Classifier with default parameters and assigns it to the variable `dtree_clf`.

`cross_val_scores = cross_validate(dtree_clf, X, y, cv=fold_count, scoring=scoring)`: This line performs cross-validation using the Decision Tree Classifier (`dtree_clf`) on the features `X` and target `y`, with `fold_count` folds, using the scoring dictionary defined earlier.

`dtree_score = mean_score(cross_val_scores)`: This line calculates the mean scores across all cross-validation folds using the `mean_score` function defined earlier.

30-34. These lines extract various performance metrics (fit time, score time, accuracy, recall, precision, F1 score) from the `dtree_score` dictionary and print them.

7. Conclusion

In this mini project, I have implemented and evaluated twelve classifiers on the phishing website dataset that consists of 6157 legitimate websites and 4898 phishing websites. The examined classifiers are Decision Tree, Ada Boost, Neural Networks, and XGBoost. According to our result in Table III, we get very good performance in an ensemble classifier namely XGBoost both on computation duration and accuracy. The main idea behind ensemble algorithms is to combine several weak learners into a stronger one, this is perhaps the primary reason why ensemble- based learning is used in practice for most of the classification problems. There are certain advantages and disadvantages inherent to the AdaBoost algorithm. AdaBoost is relatively robust to overfitting in low noisy datasets. AdaBoost has only a few hyperparameters that need to be tuned to improve model performance. Moreover, this algorithm is easy to understand and to visualize. However, for noisy data, the performance of AdaBoost is debated with some arguing that it generalizes well, while others show that noisy data leads to poor performance due to the algorithm spending too much time on learning extreme cases and skewing results. Compared to XGBoost, Moreover, AdaBoost is not optimized for speed, therefore being significantly slower than XGBoost. It is worth mentioning that there is no guarantee that the combination of multiple classifiers will always perform better than the best individual classifier in the ensemble classifiers. The results motivate future works to add more features to the dataset, which could improve the performance of these models, hence it could combine machine learning models with other phishing detection techniques like example List-Base methods to obtain better performance.

Bibliography

1. FBI, “Ic3 annual report released.”
2. APWG, “Phishing activity trends report.”
3. V. B. et al, “study on phishing attacks,” International Journal of Computer Applications, 2018.
4. I.-F. Lam, W.-C. Xiao, S.-C. Wang, and K.-T. Chen, “Counteracting phishing page polymorphism: An image layout analysis approach,” in International Conference on Information Security and Assurance, pp. 270–279, Springer, 2009.
5. W. Jing, “Covert redirect vulnerability,” 2017.
6. K. Krombholz, H. Hobel, M. Huber, and E. Weippl, “Advanced social engineering attacks,” Journal of Information Security and applications, vol. 22, pp. 113–122, 2015.
7. P. Kumaraguru, J. Cranshaw, A. Acquisti, L. Cranor, J. Hong, M. A. Blair, and T. Pham, “School of phish: a real-world evaluation of anti-phishing training,” in Proceedings of the 5th Symposium on Usable Privacy and Security, pp. 1–12, 2009.
8. R. C. Dodge Jr, C. Carver, and A. J. Ferguson, “Phishing for user security awareness,” computers & security, vol. 26, no. 1, pp. 73–80, 2007.
9. R. Dhamija, J. D. Tygar, and M. Hearst, “Why phishing works,” in Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 581–590, 2006.
10. C. Ludl, S. McAllister, E. Kirda, and C. Kruegel, “On the effectiveness of techniques to detect phishing sites,” in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 20–39, Springer, 2007.