**Web Phishing Detection Using ML**

**ABOUT :**

There are a number of users who purchase products online and make payments through e-banking. There are e-banking websites that ask users to provide sensitive data such as username, password & credit card details, etc., often for malicious reasons. This type of e-banking website is known as a phishing website. Web service is one of the key communications software services for the Internet. Web phishing is one of many security threats to web services on the Internet.

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages. The objective of this project is to train machine learning models and deep neural nets on the dataset created to predict phishing websites. Both phishing and benign URLs of websites are gathered to form a dataset and from them required URL and website content-based features are extracted. The performance level of each model is measures and compared.

APPROACH :

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

• Collect dataset containing phishing and legitimate websites from the open source platforms.

• Write a code to extract the required features from the URL database.

• Analyse and pre-process the dataset by using EDA techniques.

• Divide the dataset into training and testing sets.

• Run selected machine learning and deep neural network algorithms like SVM, Random Forest,

Autoencoder on the dataset.

• Write a code for displaying the evaluation result considering accuracy metrics.

• Compare the obtained results for trained models and specify which is better

Data Collection:

The set of phishing URLs are collected from opensource service called **Phish Tank**. This service provide a set of phishing URLs in multiple formats like csv, json etc. that gets updated hourly. To download the data: <https://www.phishtank.com/developer_info.php>. From this dataset, 5000 random phishing URLs are collected to train the ML models.

The legitimate URLs are obtained from the open datasets of the University of New Brunswick, <https://www.unb.ca/cic/datasets/url-2016.html>. This dataset has a collection of benign, spam, phishing, malware & defacement URLs. Out of all these types, the benign url dataset is considered for this project. From this dataset, 5000 random legitimate URLs are collected to train the ML models.

## Feature Extraction :

So, all together 17 features are extracted from the 10,000 URL dataset and are stored in '[5.urldata.csv](https://github.com/shreyagopal/Phishing-Website-Detection-by-Machine-Learning-Techniques/blob/master/DataFiles/5.urldata.csv)' file in the DataFiles folder.  
The features are referenced from the <https://archive.ics.uci.edu/ml/datasets/Phishing+Websites>.

## Models & Training. :

Before stating the ML model training, the data is split into 80-20 i.e., 8000 training samples & 2000 testing samples. From the dataset, it is clear that this is a supervised machine learning task. There are two major types of supervised machine learning problems, called classification and regression.

This data set comes under classification problem, as the input URL is classified as phishing (1) or legitimate (0). The supervised machine learning models (classification) considered to train the dataset in this project are:

* Decision Tree
* Random Forest
* Multilayer Perceptrons
* XGBoost
* Autoencoder Neural Network
* Support Vector Machines

All these models are trained on the dataset and evaluation of the model is done with the test dataset

**IMPORT ALL LIBRARIES :**

## **Loading Data:**

The features are extracted and store in the csv file.

The resulted csv file is uploaded to this notebook and stored in the data frame.

#importing basic packages

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

#Loading the data

data0 = pd.read\_csv('5.urldata.csv')

data0.head()

**Visualizing the data :**

data0.hist(bins = 50,figsize = (15,15))

plt.show()

A picture containing shoji, window, crossword puzzle, building

Description automatically generated

## **Data Preprocessing & EDA :**

data0.describe()

#Dropping the Domain column

data = data0.drop(['Domain'], axis = 1).copy()

#checking the data for null or missing values

data.isnull().sum()

In the feature extraction file, the extracted features of legitimate & phishing URL datasets are just concatenated without any shuffling. This resulted in top 5000 rows of legitimate URL data & bottom 5000 of phishing URL data.

To even out the distribution while splitting the data into training & testing sets, we need to shuffle it. This even evades the case of overfitting while model training.

## **Splitting the Data :**

# Sepratating & assigning features and target columns to X & y

y = data['Label']

X = data.drop('Label',axis=1)

X.shape, y.shape

# Splitting the dataset into train and test sets: 80-20 split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size = 0.2, random\_state = 12)

X\_train.shape, X\_test.shape

## **Machine Learning Models & Training :**

#importing packages

# Creating holders to store the model performance results

from sklearn.metrics import accuracy\_score

ML\_Model = []

acc\_train = []

acc\_test = []

#function to call for storing the results

def storeResults(model, a,b):

ML\_Model.append(model)

acc\_train.append(round(a, 3))

acc\_test.append(round(b, 3))

Decision Tree Classifier :

Decision trees are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision. Learning a decision tree means learning the sequence of if/else questions that gets us to the true answer most quickly.

In the machine learning setting, these questions are called tests (not to be confused with the test set, which is the data we use to test to see how generalizable our model is). To build a tree, the algorithm searches over all possible tests and finds the one that is most informative about the target variable.

# Decision Tree model

from sklearn.tree import DecisionTreeClassifier

# instantiate the model

tree = DecisionTreeClassifier(max\_depth = 5)

# fit the model

tree.fit(X\_train, y\_train)

#predicting the target value from the model for the samples

y\_test\_tree = tree.predict(X\_test)

y\_train\_tree = tree.predict(X\_train)

**Performance Evaluation:**

#computing the accuracy of the model performance

acc\_train\_tree = accuracy\_score(y\_train,y\_train\_tree)

acc\_test\_tree = accuracy\_score(y\_test,y\_test\_tree)

print("Decision Tree: Accuracy on training Data: {:.3f}".format(acc\_train\_tree))

print("Decision Tree: Accuracy on test Data: {:.3f}".format(acc\_test\_tree))

**OUTPUT :**

Decision Tree: Accuracy on training Data: 0.810

Decision Tree: Accuracy on test Data: 0.826

**Random Forest Classifier :**

Random forests for regression and classification are currently among the most widely used machine learning methods.A random forest is essentially a collection of decision trees, where each tree is slightly different from the others. The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data.

If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. To build a random forest model, you need to decide on the number of trees to build (the n\_estimators parameter of RandomForestRegressor or RandomForestClassifier). They are very powerful, often work well without heavy tuning of the parameters, and don’t require scaling of the data.

# Random Forest model

from sklearn.ensemble import RandomForestClassifier

# instantiate the model

forest = RandomForestClassifier(max\_depth=5)

# fit the model

forest.fit(X\_train, y\_train)

#predicting the target value from the model for the samples

y\_test\_forest = forest.predict(X\_test)

y\_train\_forest = forest.predict(X\_train)

**Performance Evaluation:**

#computing the accuracy of the model performance

acc\_train\_forest = accuracy\_score(y\_train,y\_train\_forest)

acc\_test\_forest = accuracy\_score(y\_test,y\_test\_forest)

print("Random forest: Accuracy on training Data: {:.3f}".format(acc\_train\_forest))

print("Random forest: Accuracy on test Data: {:.3f}".format(acc\_test\_forest))

**OUTPUT :**

Random forest: Accuracy on training Data: 0.814

Random forest: Accuracy on test Data: 0.834

**Multilayer Perceptron’s (MLPs): Deep Learning**

Multilayer perceptron’s (MLPs) are also known as (vanilla) feed-forward neural networks, or sometimes just neural networks. Multilayer perceptron’s can be applied for both classification and regression problems.

MLPs can be viewed as generalizations of linear models that perform multiple stages of processing to come to a decision.

# Multilayer Perceptrons model

from sklearn.neural\_network import MLPClassifier

# instantiate the model

mlp = MLPClassifier(alpha=0.001, hidden\_layer\_sizes=([100,100,100]))

# fit the model

mlp.fit(X\_train, y\_train)

#predicting the target value from the model for the samples

y\_test\_mlp = mlp.predict(X\_test)

y\_train\_mlp = mlp.predict(X\_train)

**OUTPUT :**

Multilayer Perceptrons: Accuracy on training Data: 0.859

Multilayer Perceptrons: Accuracy on test Data: 0.863

**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.

#Support vector machine model

from sklearn.svm import SVC

# instantiate the model

svm = SVC(kernel='linear', C=1.0, random\_state=12)

#fit the model

svm.fit(X\_train, y\_train)

#predicting the target value from the model for the samples

y\_test\_svm = svm.predict(X\_test)

y\_train\_svm = svm.predict(X\_train)

**Performance Evaluation:**

acc\_train\_svm = accuracy\_score(y\_train,y\_train\_svm)

acc\_test\_svm = accuracy\_score(y\_test,y\_test\_svm)

print("SVM: Accuracy on training Data: {:.3f}".format(acc\_train\_svm))

print("SVM : Accuracy on test Data: {:.3f}".format(acc\_test\_svm))

OUTPUT :

SVM: Accuracy on training Data: 0.798

SVM : Accuracy on test Data: 0.818