Machine Learning - 19CSE305

Phishing URL Detection using Machine Learning

Yadhu Krishna M
yadhukrishna.mpm@gmail.com

November 20, 2021

1 Problem Definition

The project aims to build a machine learning model that would predict whether a given Uniform Resource Locator (URL) is a potential phishing link (Task). The algorithm would be trained using a dataset of URLs in which some are phishing URLs and others are legitimate ones (Experience).

2 Dataset Used

The dataset used in this project contains 11430 URLs out of which 50% are phishing and 50% legitimate URLs. It has 87 extracted features out of which 56 are extracted from the structure and syntax of URLs, 24 extracted from the content of their correspondent pages, and 7 extracted by querying external services.
The following are some of the major features present in the dataset.

1. Length of URL
2. Length of hostname
3. Use of any URL shortening service
4. Suspicious TLD
5. No of redirections - external or internal
6. Number occurrences of special characters in URL
7. Search Index ranking

Three other datasets were also used for the same project, and would be documented in other submissions.

3 Data Pre-processing

This section describes the steps that were done to preprocess the data.

- Most of the features in the dataset contained numerical values indicating features that were derived from a URL string.
- The result column initially contained non-numeric values which were used to indicate legitimate and phishing URLs. This column was mapped into 0 and 1 respectively.
- Since this project was focused on only features derived from URL such as length of URL, number of special characters etc. So, other features in the dataset such as results from third-party services (results from search engine, WHOIS lookups etc.) were omitted.
- The dataset was checked for null values, and duplicate entries however, no such records were observed.
- Normalization was performed using MinMax scaler on the entire dataset.

4 Data Summarization

This section describes the use of statistical methods applied to breakdown, and understand the data.

The dataset was inspected and it was observed to contain 11,430 unique entries after preprocessing. The number of features were reduced to 34. There were 33 features extracted from the URL, and a column indicating whether the entry was that of a legitimate or a phishing URL. On grouping the dataset by the result, it was identified that the dataset was properly balanced, and it contained 50% legitimate and 50% phishing URLs. There were 5,715 legitimate and phishing URLs.

Code used for summarizing this data has been uploaded to the project folder.

5 Data Visualization

After summarizing the data, two types of graphs - a scatter plot and a double bar graph were plotted to better understand how each feature affected the result.

The dataset was divided into two classes - **legitimate** and **phishing** to easily inspect the data. One of the main features of the dataset was the length of URL and

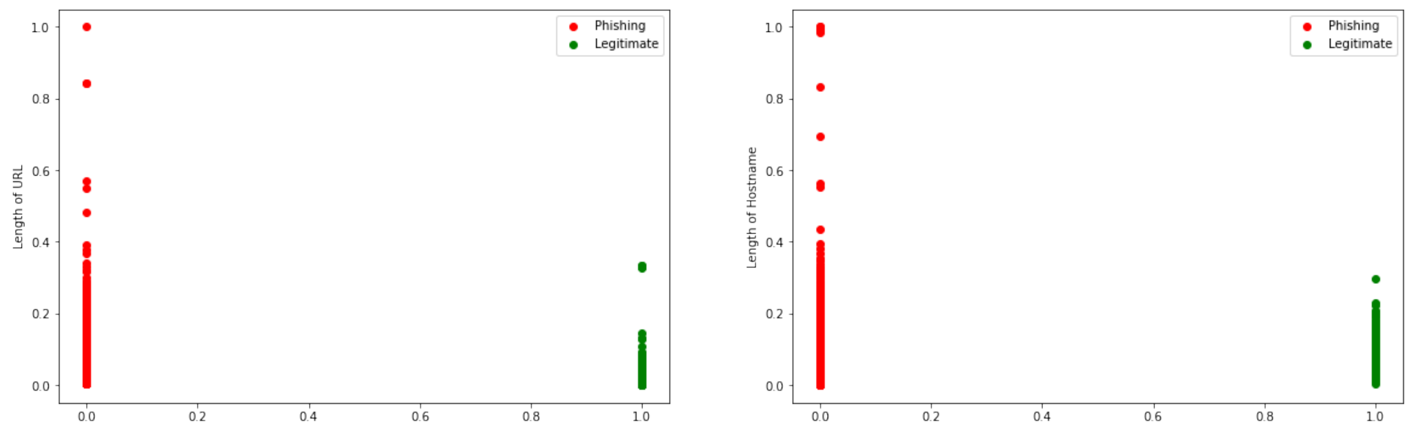


Figure 1: Scatter Plot depecting how length of URL varies in legitimate and phishing URLs.

the length of hostname. Phishing websites are normally said to have long URLs to hide certain parts of the URL. A scatter plot was drawn to see the visibility of this feature in the dataset.

Double bar graphs were also plotted to compare how the remaining features varied for both the classes.

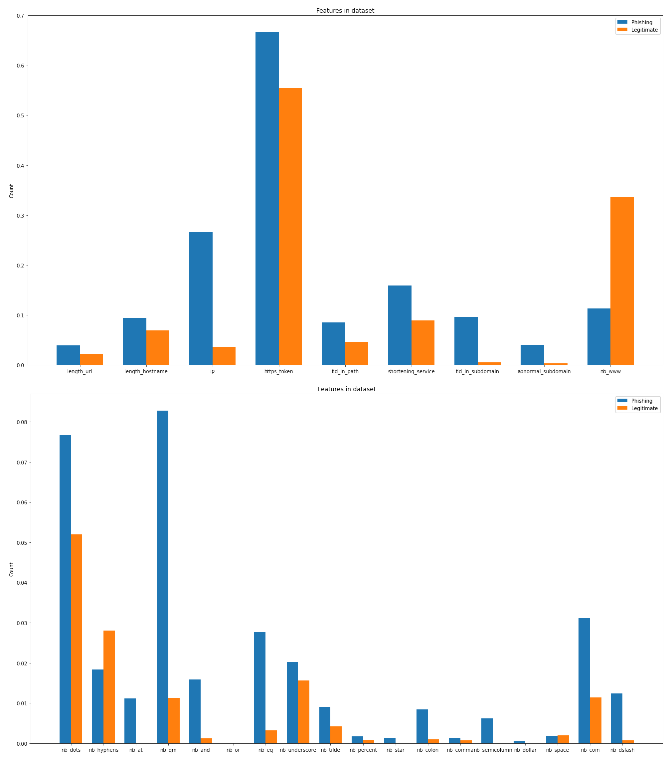


Figure 2: Double bar graphs depecting how features vary in legitimate and phishing URLs.

6 Data Interpretation

Based on 1, it could be observed that lengthier URLs and hostnames are more probabilistic to be phishing URLs.

Based on Fig. III and IV, the following observations could be made:

1. The length of domain names and URLs are longer for phishing URLs.
2. Majority of URLs having an IP address instead of a domain name, were identified to be phishing.
3. Phishing websites were found to be using URL shorteners, and did not have SSL support.
4. Most legitimate websites had www in their URL.
5. The number of phishing websites that had a top-level domain name as a subdomain or as a path was comparatively high.
6. The number of occurrences of special characters such as dots, at, question marks, ampersands, underscores, tildes, stars, colons, commas, etc were found to be more in phishing URLs rather than legitimate URLs . However, the count of hyphens were found to be more in legitimate URLs.

7 Python packages used

The following python packages were used in order to build, train, and compare various machine learning models.

- scikit-learn - It is a machine learning library that provides various classification, regression and clustering algorithms, and related tools to test, train, and validate the generated model. This library was used in the initial stage of the project, in order to find a best suitable algorithm.
- Matplotlib - It is a data visualization library in Python which is used to plot graphs. The library was used in this project to analyze results graphically.
- Pandas - It is a software library written for the Python programming language for data manipulation and analysis

- NumPy - It is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

8 Model Selection

In this section, we choose four different supervised machine learning algorithms, and analyze their performance separately on the dataset. The best algorithm would be selected for implementation.

The algorithms chosen are as follows:

1. K-Nearest Neighbor Algorithm (KNN)
2. Support Vector Machine (SVM)
3. Gaussian Naive Bayes Classifier
4. Decision Tree Algorithm

The results has been summarized in the following section.

8.1 Analysis

The code used to build and compare the models discussed in this section has been uploaded in the project folder.

8.1.1 K-Nearest Neighbor Algorithm (KNN)

The k-nearest neighbors (KNN) algorithm is a supervised machine learning algorithm that can be used to solve both classification and regression problems.

The algorithm was run on the dataset with varying values of k, and the best average accuracy gained was 0.83, with K-Fold Cross Validation, with the number of nearest neighbours as 3.

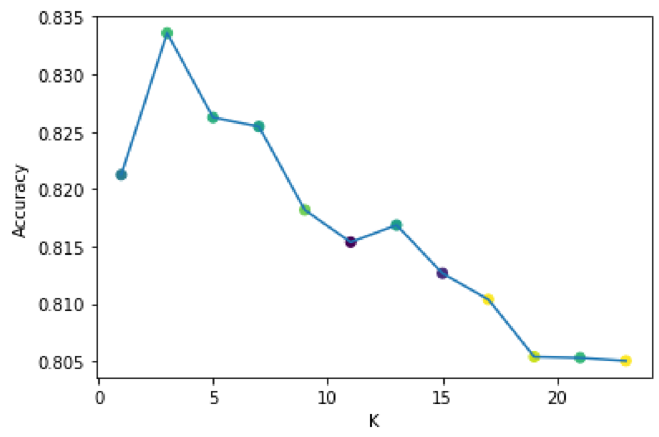


Figure 3: Plot showing accuracy of KNN Algorithm with varying values of K

8.1.2 Support Vector Machine (SVM)

Support vector machines (SVMs) are a set of supervised learning methods used for classification and regression problems. The Support Vector Machine algorithm gave an average accuracy of 0.77 with RBF Kernel. A lowest score of 0.65 was obtained with Sigmoid Kernel.

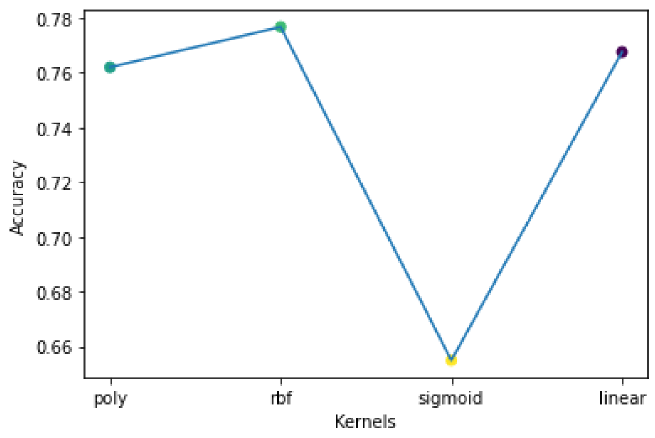


Figure 4: Plot showing accuracy of SVM Algorithm with various kernels.

8.1.3 Gaussian Naive Bayes Classifier

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data.

Gaussian Naive Bayes classifier yielded an average accuracy of 0.61.

8.1.4 Decision Tree Algorithm

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problem. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

Decision Tree Algorithm yielded an average accuracy of 0.82.

8.2 Summary

The results obtained from each algorithm has been summarized in the table below.

Algorithm	Accuracy Obtained
KNN	0.83
SVM	0.77
Gaussian Naive Bayes	0.61
Decision Tree	0.82

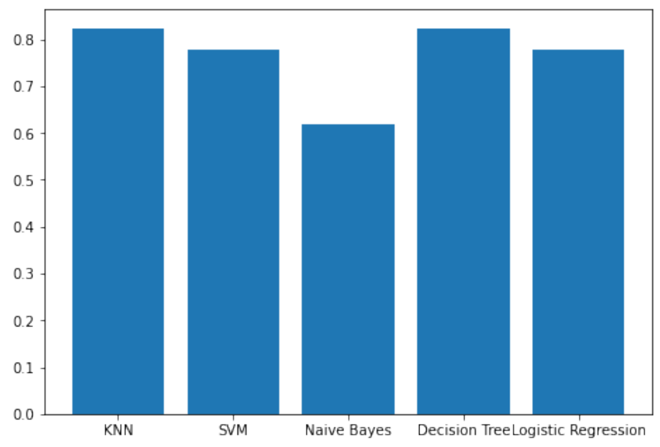


Figure 5: Comparison of various machine learning algorithms on the same dataset.

From the observations stated above, the KNN Algorithm appears as a best choice for this problem. We implemented KNN Algorithm, and the implementation has been discussed in the next section.

9 Implementing KNN

The following are the requirements to implement the KNN Algorithm.

1. Shuffle the dataset in order to reduce the problem of overfitting.
2. Split dataset into test, train and validation datasets.
3. A function to calculate Euclidean Distance.
4. A method to find K nearest neighbours of a given datapoint.
5. A classifier function that classifies a datapoint into phishing or legitimate, on the basis of nearest neighbours.
6. A method to measure accuracy of the implemented algorithm.

The required methods were implemented and has been documented in ipynb files uploaded in the project folder.

9.1 Source Code

9.1.1 Loading dataset

```
# Load the cleaned dataset from Phase - 1

data = pd.read_csv('final_dataset.csv')

# Shuffle the dataset usnig Numpy Random Permutation
shuffle_index = np.random.permutation(data.shape[0])
data = data.iloc[shuffle_index]


# Use 70% of the dataset for training, and remaining 30% for testing.
# In[4]:
train_size = int(data.shape[0]*0.7)

# In[5]:
train_df = data.iloc[:train_size,:]
test_df = data.iloc[train_size:,:]

train = train_df.values
test = test_df.values

y_true = test[:,-1]
```

9.1.2 Euclidean Distance

The euclidean distance betweet two vectors is calculated using the formula :

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

The same is implemented in Python below.

```python
from math import sqrt

def euclidean_distance(x_test, x_train):
    """
    Calculates Euclidean distance between two given vectors.
    Args: Vector A, Vector B
    Returns: Distance between two given vectors.
    """

    distance = 0

    for i in range(len(x_test)-1):
        distance = distance + (x_test[i] - x_train[i])**2

    return sqrt(distance)
```

9.1.3 Finding N nearest neighbors

```python
def find_n_neighbors(x_test, x_train, N):
    """
    Function that finds N nearest neighbors of a vector by calculating euclidean
   distance between them.

    Args:
        x_test : Vector A,
        x_train : Set of vectors
        N : Number of nearest neighbors required

    Returns:
        N Closest vectors to vector A

    """

    distances = []
    data = []

    # For each datapoint in training set,
    # find the distance from X_test vector
    for row in x_train:
        distances.append(euclidean_distance(x_test, row))

    # Sort distances and return N records
    distances = np.array(distances)
    data = np.array(x_train)
    sort_indexes = distances.argsort()
    data = data[sort_indexes]

    return data[:N]
```

9.1.4 The KNN Function

```python
def knn(x_test, x_train, N):
    """
        The KNN Classifier Function
        Arguments:
            x_test : Test point
            x_train : Set of N vectors
            N : no of nearest neighbours
        Returns:
            Class of test point
    """

    positive = 0
    negative = 0

    # Find N nearest neighbours
    # Class of the test point will be
    # class of majority of nearing classes
    neighbors = find_n_neighbors(x_test, x_train, N)

    for i in neighbors:

        if i[-1] == 0:
            negative += 1
        else:
            positive += 1

    if negative > positive:
        return 0

    return 1
```

9.2 Calculating Accuracy

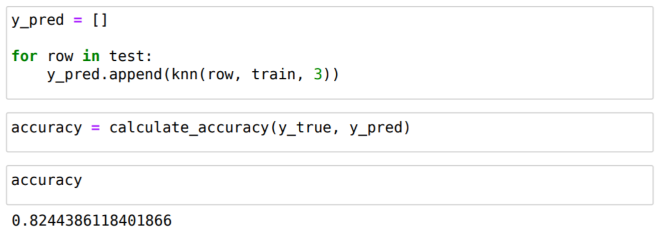
```python
def calculate_accuracy(y_true, y_pred):
    """
    Function to calculate accuracy given predicted classes, and originally correct
    classes.
    Args:
        y_true : Actual true classes
        y_pred : Classes predicted using KNN

    Returns:
        Accuracy
    """
    correct = 0

    for i in range(len(y_true)):
        if y_true[i] == y_pred[i]:
            correct += 1

    accuracy = correct / len(y_true)
    return accuracy
```

9.3 Result



```
y_pred = []

for row in test:
    y_pred.append(knn(row, train, 3))
```

```
accuracy = calculate_accuracy(y_true, y_pred)
```

```
accuracy
```

```
0.8244386118401866
```

Upon running the KNN function with k = 3, an accuracy of 0.82 was obtained on unseen data.

10 Conclusion

A machine learning algorithm (K nearest neighbors classifier) was developed from scratch in order to classify phishing and legitimate URLs.

Link to project folder