# Machine Learning in Cybersecurity and Privacy

## Challenge 2 - Classifying the type of URLs (Malicious or Benign) using KNN Algorithm

By Smit Doshi (001475186)

#### Importing required libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

#### Reading the data from the provided CSV file

In [2]:

```
data = pd.read_csv("Dataset_Challenge2.csv")
```

#### Replacing all N/A values with 0

In [3]:

```
data.fillna(0,inplace=True)
```

#### Importing standard scaler

In [4]:

```
from sklearn.preprocessing import StandardScaler
```

#### Creating Instance of the scaler

In [5]:

```
scaler = StandardScaler()
```

#### Fitting our data into scaler instance

In [6]:

```
scaler.fit(data.drop('Type',axis=1))
```

Out[6]:

```
StandardScaler()
```

#### Transforming our data to standardized data

In [7]:

```
scaled_features = scaler.transform(data.drop('Type',axis=1))
```

#### Converting it into dataframe

In [8]:

```
df_feat = pd.DataFrame(scaled_features,columns=data.columns[:-1])
```

#### Importing required modules

In [9]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

#### Splitting the dataset into training and test sets

#### Here, the training set is 80% of the set and test set is 20% of the original dataset

In [10]:

```
X = df_feat
y = data['Type']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
te=101)
```

#### Instantiating KNN algorithm for k = 3

In [11]:

```
knn = KNeighborsClassifier(n_neighbors=3)
```

#### Fitting the model

In [12]:

```
knn.fit(X_train,y_train)
```

Out[12]:

```
KNeighborsClassifier(n_neighbors=3)
```

#### Predicting the values

In [13]:

```
pred = knn.predict(X_test)
```

#### Importing metrics

In [14]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [15]:

```
print(confusion_matrix(y_test,pred))
print("-----------")
print(classification_report(y_test,pred))
```

```
[[300  15]
 [ 10  32]]
-----------
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       315
           1       0.68      0.76      0.72        42

    accuracy                           0.93       357
   macro avg       0.82      0.86      0.84       357
weighted avg       0.93      0.93      0.93       357
```

#### Checking the performance for variety of Ks to determine the best K (accuracy wise)

In [16]:

```
error_rate =[]
for i in range(1,140):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

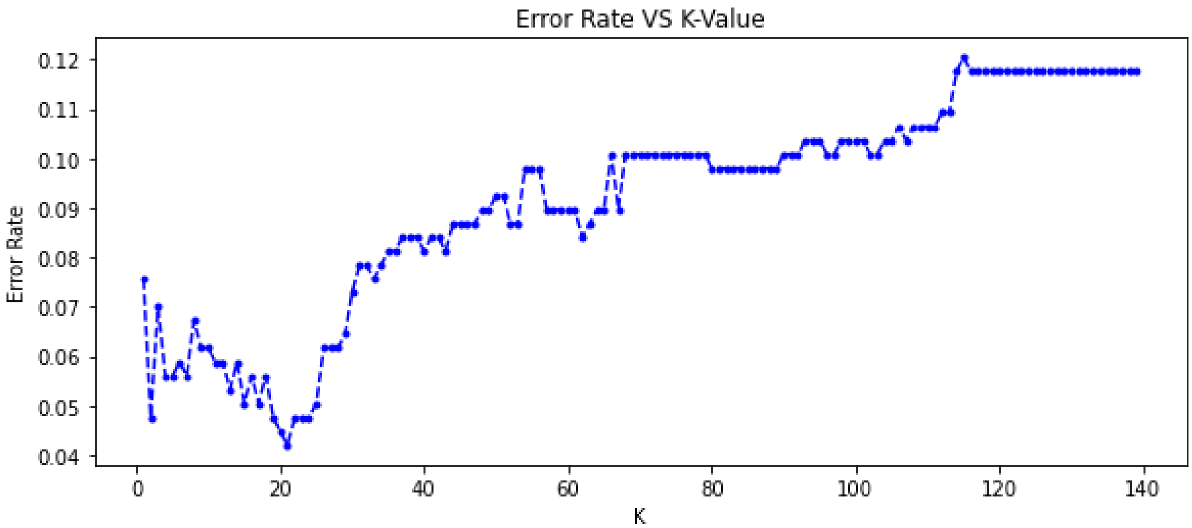
#### Plotting a figure to visualize the performance

In [17]:

```
plt.figure(figsize=(10,4))
plt.plot(range(1,140),error_rate,color='blue',linestyle='--',marker='.')
plt.title("Error Rate VS K-Value")
plt.xlabel("K")
plt.ylabel('Error Rate')
```

Out[17]:

```
Text(0, 0.5, 'Error Rate')
```




**According to performance check for a variety of Ks- I think the model predicts or performs well for K = 20**

**Accuracy = 96 %**

In [18]:

```
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X_train,y_train)
pred= knn.predict(X_test)
print(confusion_matrix(y_test,pred))
print("-----------")
print(classification_report(y_test,pred))
```

```
[[313   2]
 [ 14  28]]
-----------
              precision    recall  f1-score   support

           0       0.96      0.99      0.98       315
           1       0.93      0.67      0.78        42

    accuracy                           0.96       357
   macro avg       0.95      0.83      0.88       357
weighted avg       0.95      0.96      0.95       357
```