The LNM Institute of Information Technology

# IDS Project Report

ML Classification on Internet Firewall Dataset


**Group Members**

Muskan Singla : 20ucc068

Prabhav Jain : 20ucc074

Sourabh Joshi : 20ucc103

Shreya Agarwal : 20ucs186

# Introduction

These days, we are witnessing unprecedented challenges to network security. This indeed confirms that network security has become increasingly important. Firewall logs are important sources of evidence, but they are still difficult to analyze. Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) have emerged as effective in developing robust security measures due to the fact that they have the capability to deal with complex cyberattacks in a timely manner.

**Aim: Classification of Firewall Log Data Using Multi-class Machine Learning Models**

This work aims to tackle the difficulty of analyzing firewall logs using ML and DL by building multi-class ML and DL models that can analyze firewall logs and classify the actions to be taken in response to received sessions as "Allow", "Drop", "Deny", or "Reset-both".

Dataset needs to be explored to find the insights and significant relationships between different attributes and the target variable. Then apply ML classification algorithms on the data to train proper models and get accurate inferences.

# Description of Dataset

This data set was collected from the internet traffic records on a university's firewall.

| | | | | | |
|---|---|---|---|---|---|
| **Data Set Characteristics:** | Multivariate | **Number of Instances:** | 65532 | **Area:** | Computer |
| **Attribute Characteristics:** | N/A | **Number of Attributes:** | 12 | **Date Donated** | 2019-02-04 |
| **Associated Tasks:** | Classification | **Missing Values?** | N/A | **Number of Web Hits:** | 27254 |

Number of Instances: 65532
Number of Attributes: 12

There are 12 features in total. Action feature is used as a class. There are 4 classes in total. **These are allow, deny, drop and reset-both classes**.

**Attributes :**

- Source Port
- Destination Port
- NAT Source Port
- NAT Destination Port
- Bytes
- Bytes Sent
- Bytes Received
- Packets
- Elapsed Time (sec)
- Pkts_sent
- pkts_received
- Action (Target Class)

TABLE I. FEATURES AND DESCRIPTION

| Feature | Description |
|---|---|
| Source Port | Client Source Port |
| Destination Port | Client Destination Port |
| NAT Source Port | Network Address Translation Source Port |
| NAT Destination Port | Network Address Translation Destination Port |
| Elapsed Time (sec) | Elapsed Time for flow |
| Bytes | Total Bytes |
| Bytes Sent | Bytes Sent |
| Bytes Received | Bytes Received |
| Packets | Total Packets |
| pkts_sent | Packets Sent |
| pkts_received | Packets Received |
| Action | Class (allow, deny, drop, reset-both) |

There are 4 classes in the action attribute used as a class

# Importing Requirements

Dataset - https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data

Before importing the dataset, we need to import proper libraries like pandas, numpy, matplotlib and seaborn for data processing and visualization

random library is used for sampling purpose.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

# Importing Dataset

- Reading the data from log2.csv and naming the column manually, and storing it in a variable named ‘data’.

```
data = pd.read_csv('log2.csv',
                names = ["Destination Port","NAT Destination Port","Action","Bytes","Bytes Sent",
                     "Bytes Received","Packets","Elapsed Time (sec)","pkts_sent","pkts_received"])
```

# Exploring the DataSet

- The first 10 rows of the data set

```
data.head(10)
```

| | Source Port | Destination Port | NAT Source Port | NAT Destination Port | Action | Bytes | Bytes Sent | Bytes Received | Packets | Elapsed Time (sec) | pkts_sent | pkts_received |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Source Port | Destination Port | NAT Source Port | NAT Destination Port | Action | Bytes | Bytes Sent | Bytes Received | Packets | Elapsed Time (sec) | pkts_sent | pkts_received |
| 1 | 57222 | 53 | 54587 | 53 | allow | 177 | 94 | 83 | 2 | 30 | 1 | 1 |
| 2 | 56258 | 3389 | 56258 | 3389 | allow | 4768 | 1600 | 3168 | 19 | 17 | 10 | 9 |
| 3 | 6881 | 50321 | 43265 | 50321 | allow | 238 | 118 | 120 | 2 | 1199 | 1 | 1 |
| 4 | 50553 | 3389 | 50553 | 3389 | allow | 3327 | 1438 | 1889 | 15 | 17 | 8 | 7 |
| 5 | 50002 | 443 | 45848 | 443 | allow | 25358 | 6778 | 18580 | 31 | 16 | 13 | 18 |
| 6 | 51465 | 443 | 39975 | 443 | allow | 3961 | 1595 | 2366 | 21 | 16 | 12 | 9 |
| 7 | 60513 | 47094 | 45469 | 47094 | allow | 320 | 140 | 180 | 6 | 7 | 3 | 3 |
| 8 | 50049 | 443 | 21285 | 443 | allow | 7912 | 3269 | 4643 | 23 | 96 | 12 | 11 |
| 9 | 52244 | 58774 | 2211 | 58774 | allow | 70 | 70 | 0 | 1 | 5 | 1 | 0 |

- Dataset contains 65533 rows and 12 columns

```
data.shape
```

```
(65533, 12)
```

- The below code is showing the count of rows, unique value in the respective columns, max occurring value in a respective columns and their frequency

```
data.describe()
```

| | Source Port | Destination Port | NAT Source Port | NAT Destination Port | Action | Bytes | Bytes Sent | Bytes Received | Packets | Elapsed Time (sec) | pkts_sent | pkts_received |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 65533 | 65533 | 65533 | 65533 | 65533 | 65533 | 65533 | 65533 | 65533 | 65533 | 65533 | 65533 |
| unique | 22725 | 3274 | 29153 | 2534 | 5 | 10725 | 6684 | 8815 | 1117 | 916 | 750 | 923 |
| top | 58638 | 53 | 0 | 0 | allow | 70 | 70 | 0 | 1 | 0 | 1 | 0 |
| freq | 840 | 15414 | 28432 | 28432 | 37640 | 10651 | 11015 | 31574 | 29829 | 28265 | 45253 | 31574 |

- There are no missing values in dataset in any of the attributes.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 65533 entries, 0 to 65532
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Source Port           65533 non-null  object
 1   Destination Port      65533 non-null  object
 2   NAT Source Port       65533 non-null  object
 3   NAT Destination Port  65533 non-null  object
 4   Action                65533 non-null  object
 5   Bytes                 65533 non-null  object
 6   Bytes Sent            65533 non-null  object
 7   Bytes Received        65533 non-null  object
 8   Packets               65533 non-null  object
 9   Elapsed Time (sec)    65533 non-null  object
 10  pkts_sent             65533 non-null  object
 11  pkts_received         65533 non-null  object
dtypes: object(12)
memory usage: 6.0+ MB
```

- Other way of finding number of null values

```
data.isna().sum()
```

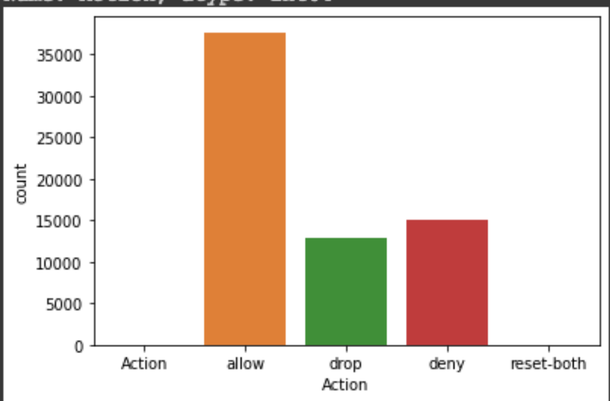
```
Source Port              0
Destination Port         0
NAT Source Port          0
NAT Destination Port     0
Action                   0
Bytes                    0
Bytes Sent               0
Bytes Received           0
Packets                  0
Elapsed Time (sec)       0
pkts_sent                0
pkts_received            0
dtype: int64
```

## Data Visualization

- Using count plot we can know how the data is spread into different categories of the target class

```
print(data['Action'].value_counts())
sns.countplot(x='Action',data=data)
plt.show()
```

```
allow          37640
deny           14987
drop           12851
reset-both        54
Action             1
Name: Action, dtype: int64
```



Observation :

Allow : 37640
Deny : 14987
Drop : 12851
Reset-both : 54

The dataset is without any errors so we can proceed with this dataset.

## Data Analysis

- Finding the correlation of all features with Class by using Spearman Correlation
- Spearman's correlation measures the strength and direction of monotonic association between two variables.
- The Spearman correlation coefficient is defined as the Pearson correlation coefficient between the ranked variables.

```
a=[]
a=data.apply(lambda col:col.corr(data['Action'],method='spearman'),axis=0)
a=a.abs().sort_values(ascending=False)
a
```

```
Action                  1.000000
NAT Destination Port    0.857614
Elapsed Time (sec)      0.855720
NAT Source Port         0.849707
Packets                 0.830663
pkts_received           0.800600
Bytes Received          0.793962
Bytes                   0.750417
Bytes Sent              0.693658
Destination Port        0.546605
pkts_sent               0.541731
Source Port             0.059236
dtype: float64
```

## Preprocessing the data for the model

In order to do any training with the dataset, we need to first preprocess the data. Data preprocessing plays a major role before applying any algorithm.

Since our entire dataset is already numerical except for the target variable , we should only work on target variable.
We can convert the strings to numbers or binary codes in two ways ie. categorical codes and by using dummy values.

As our number of samples are too large , therefore first we will sample our data as follows.

```
n = 65532 #number of records in file
s = 5000 #desired sample size
skip = sorted(random.sample(range(n),n-s))
data = pd.read_csv('log2.csv',
                   names = ["Source Port","Destination Port","NAT Source Port","NAT Destination Port","Action","Bytes","Bytes Sent",
                            "Bytes Received","Packets","Elapsed Time (sec)","pkts_sent","pkts_received"] , skiprows=skip)

data.head(10)

data.shape
```

```
(5001, 12)
```

Here converting the target feature that is ‘Action’ using categorical codes

```
data["Action"]=data["Action"].astype("category").cat.codes
x1=data["Action"]
data.head(5)
```

| | Source Port | Destination Port | NAT Source Port | NAT Destination Port | Action | Bytes | Bytes Sent | Bytes Received | Packets | Elapsed Time (sec) | pkts_sent | pkts_received |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 50049 | 443 | 21285 | 443 | 0 | 7912 | 3269 | 4643 | 23 | 96 | 12 | 11 |
| 1 | 1939 | 53 | 33288 | 53 | 0 | 210 | 78 | 132 | 2 | 30 | 1 | 1 |
| 2 | 34529 | 443 | 30445 | 443 | 0 | 62652 | 30891 | 31761 | 158 | 381 | 67 | 91 |
| 3 | 56172 | 53 | 35826 | 53 | 0 | 253 | 87 | 166 | 4 | 30 | 2 | 2 |
| 4 | 55703 | 445 | 0 | 0 | 2 | 70 | 70 | 0 | 1 | 0 | 1 | 0 |

Checking and storing all other features as it will be used in algorithms.

```
x2 = data.drop('Action',axis=1)
x2.head(5)
```

| | Source Port | Destination Port | NAT Source Port | NAT Destination Port | Bytes | Bytes Sent | Bytes Received | Packets | Elapsed Time (sec) | pkts_sent | pkts_received |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 49164 | 443 | 45916 | 443 | 7292 | 950 | 6342 | 19 | 75 | 9 | 10 |
| 1 | 55723 | 445 | 0 | 0 | 70 | 70 | 0 | 1 | 0 | 1 | 0 |
| 2 | 53985 | 30188 | 0 | 0 | 62 | 62 | 0 | 1 | 0 | 1 | 0 |
| 3 | 50995 | 445 | 0 | 0 | 70 | 70 | 0 | 1 | 0 | 1 | 0 |
| 4 | 48706 | 23393 | 0 | 0 | 60 | 60 | 0 | 1 | 0 | 1 | 0 |

Then we will concatenate both the Processed data into one dataframe.

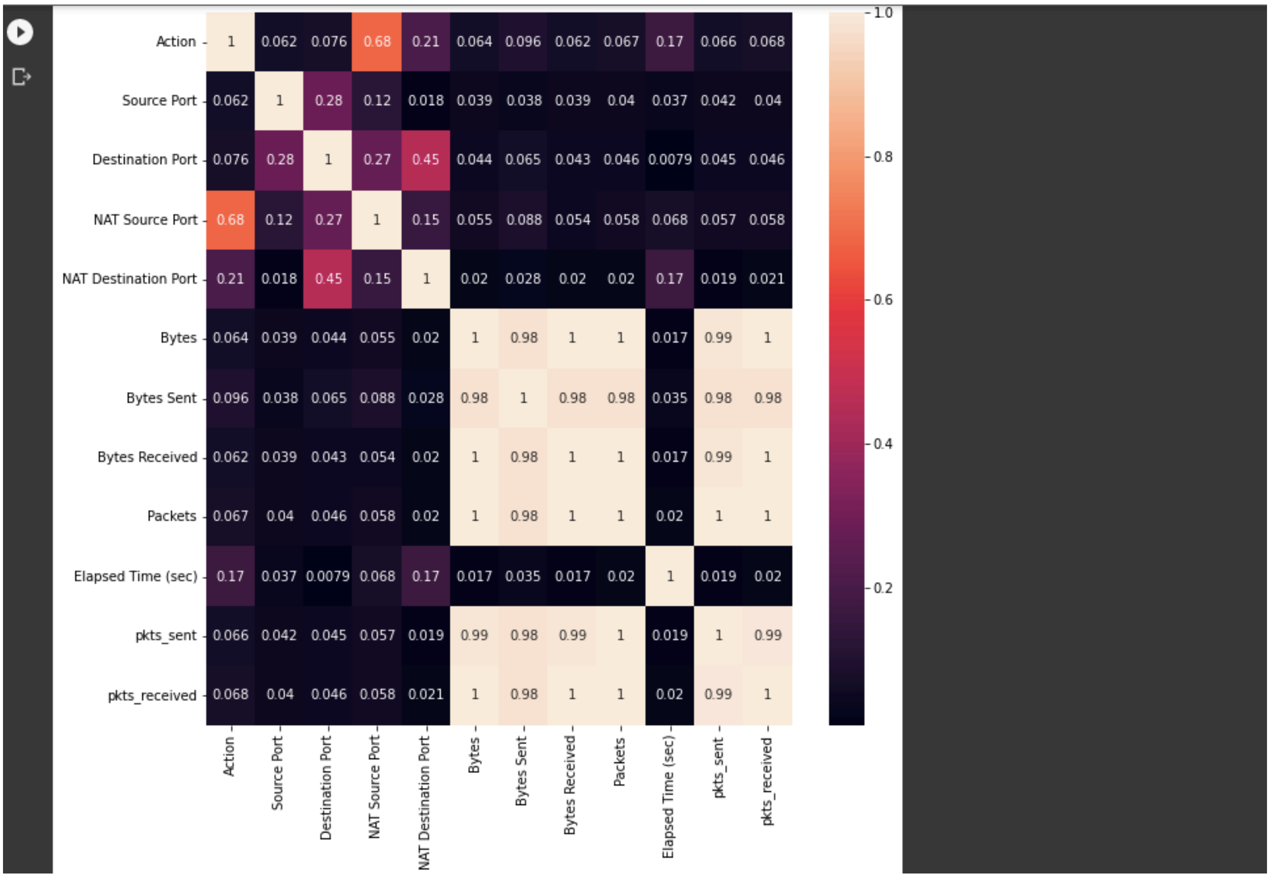
```
#concatenating into one dataframe
x = pd.concat([x1,x2],axis=1)
x.head(5)
```

| | Action | Source Port | Destination Port | NAT Source Port | NAT Destination Port | Bytes | Bytes Sent | Bytes Received | Packets | Elapsed Time (sec) | pkts_sent | pkts_received |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 42412 | 443 | 55628 | 443 | 204 | 70 | 134 | 4 | 26 | 2 | 2 |
| 1 | 0 | 40737 | 443 | 35009 | 443 | 44004 | 3909 | 40095 | 80 | 135 | 38 | 42 |
| 2 | 2 | 55708 | 445 | 0 | 0 | 70 | 70 | 0 | 1 | 0 | 1 | 0 |
| 3 | 1 | 59126 | 57470 | 0 | 0 | 66 | 66 | 0 | 1 | 0 | 1 | 0 |
| 4 | 2 | 50970 | 445 | 0 | 0 | 70 | 70 | 0 | 1 | 0 | 1 | 0 |

# Correlation matrix with Heatmap

A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data.

```
#correlation between attributes
corr=abs(x.corr())
plt.figure(figsize=(10,10))
sns.heatmap(corr,annot=True)
plt.show()
```




Observations:
Many features are highly co-related with each other like
Bytes with Bytes_received, packets ,pkts_sent and pkt_received,
Bytes_received with Bytes,Packets pkts_sent and Pkts_received,
pkts_sent with pkts_received and many more.

# Dividing Data

Now, we are dividing the dataset into two categories - X, which contains all the features except target attribute and Y, which contains the target attribute. It is done as below:

```
target = ["Action"]
X = x.drop(target, axis=1)

target = x[target]
Y = pd.DataFrame(target)
Y = np.array(Y)

X.shape
```

```
(1001, 11)
```

```
target = ["Action"]
X = x.drop(target, axis=1)

target = x[target]
Y = pd.DataFrame(target)
Y = np.array(Y)

X.shape

Y.shape
```

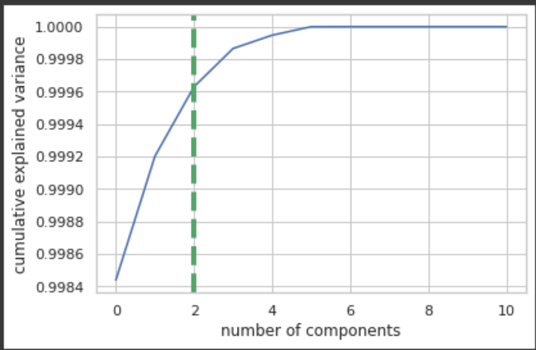
```
(1001, 1)
```

# Principal component Analysis (PCA)

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.

Here is the graph showing variance with respect to number of components used.

```
pca_test = PCA(n_components=11)

xtr = pca_test.fit_transform(X)

xtr.shape

sns.set(style='whitegrid')
plt.plot(np.cumsum(pca_test.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.axvline(linewidth=4, color='g', linestyle = '--', x=2, ymin=0, ymax=1)
display(plt.show())
```




```
None
```

# Splitting the data into training and test data

- We need some data to train and test the model
- Sklearn provides the function that splits the data into training and test using some algorithm

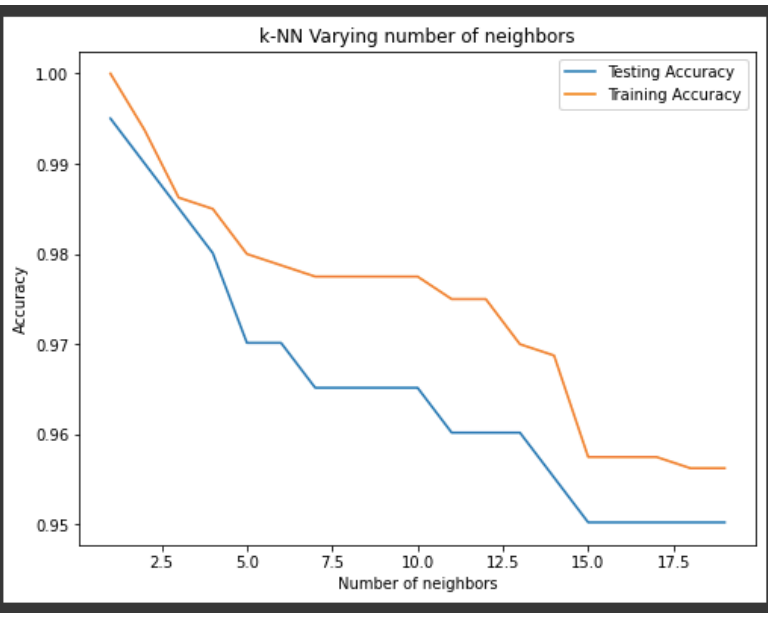
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(xtr,Y,test_size=0.2)
```

# Classifications

## KNN Classification

- K-nearest neighbours (k-NN) is a pattern recognition technique that finds the k closest relatives in future cases using training datasets.
- We calculate to place data under the category of its nearest neighbour while using k-NN in classification.
- If k = 1, it will be assigned to the class closest to 1. A plurality vote of its neighbours classifies K.

```
# KNN Classifier
from sklearn.neighbors import KNeighborsClassifier

neighbors = np.arange(1,20)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
   #Setup a knn classifier with k neighbors
   knn = KNeighborsClassifier(n_neighbors=k)

   #Fit the model
   knn.fit(xtrain, ytrain.ravel())

   #Compute accuracy on the training set
   train_accuracy[i] = knn.score(xtrain, ytrain.ravel())

   #Compute accuracy on the test set
   test_accuracy[i] = knn.score(xtest, ytest.ravel())

#Generate plot
fig = plt.figure(1, figsize=(8,6))
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label= 'Testing Accuracy')
plt.plot(neighbors, train_accuracy, label= 'Training Accuracy')
plt.legend(prop={'size':10})
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```




```
classifier = KNeighborsClassifier(n_neighbors=6)
classifier.fit(xtrain, ytrain.ravel())

from sklearn.metrics import classification_report, confusion_matrix
y_pred = classifier.predict(xtest)
print(confusion_matrix(ytest, y_pred))
print(classification_report(ytest, y_pred))
```

```
[[112   2   1]
 [  0  51   0]
 [  0   0  35]]
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       115
           1       0.96      1.00      0.98        51
           2       0.97      1.00      0.99        35

    accuracy                           0.99       201
   macro avg       0.98      0.99      0.98       201
weighted avg       0.99      0.99      0.99       201
```

```
from sklearn.metrics import accuracy_score

knn_acc = accuracy_score(ytest, y_pred)
knn_acc
```

```
0.9751243781094527
```

# Naive Bayes Classification

- Every pair of features being classified is independent of each other, according to the Naive Bayes Classifier algorithm.
- The feature matrix and the response vector are the two elements of our dataset.
- It can be used in text analysis to classify words or phrases as belonging to a predefined "tag" (classification) or not.

```
#Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
model2 = gnb.fit(xtrain, ytrain)
prediction2 = model2.predict(xtest)

print('Accuracy on training data: {:,.3f}'.format(gnb.score(xtrain,ytrain)))
print('Accuracy on test data: {:,.3f}'.format(gnb.score(xtest,ytest)))
```

```
Accuracy on training data: 0.961
Accuracy on test data: 0.940
```

```
accuracy_score(ytest, prediction2)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning
  y = column_or_1d(y, warn=True)
0.8905472636815921
```

```
print(confusion_matrix(ytest, prediction2))
print(classification_report(ytest, prediction2))
```

```
[[56 30 30]
 [26  7 14]
 [14 10 14]]
              precision    recall  f1-score   support

           0       1.00      0.85      0.92       116
           1       0.74      0.98      0.84        47
           2       0.95      1.00      0.97        38

    accuracy                           0.91       201
   macro avg       0.90      0.94      0.91       201
weighted avg       0.93      0.91      0.91       201
```

# Support Vector Machine Classification

- A support vector machine (SVM) is a type of machine that employs methods to train and classify input within degrees of polarity, going beyond X/Y prediction.
- The SVM algorithm's purpose is to find the optimum line or decision boundary for categorising n-dimensional space into classes so that additional data points can be readily placed in the correct category in the future.
- A hyperplane is the name for the optimal choice boundary.

```
#classification using svm
from sklearn.svm import SVC

svc = SVC()

model3 = svc.fit(xtrain, ytrain)
prediction3 = model3.predict(xtest)

print("Accuracy on training data: {:,.3f}".format(svc.score(xtrain,ytrain)))
print("Accuracy on test data: {:,.3f}".format(svc.score(xtest, ytest)))
```

```
Accuracy on training data: 0.594
Accuracy on test data: 0.547
```

```
accuracy_score(ytest, prediction3)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning:
  y = column_or_1d(y, warn=True)
0.6766169154228856
```

```
print(confusion_matrix(ytest, prediction3))
print(classification_report(ytest, prediction3))
```

```
[[112   0   0]
 [ 44   0   0]
 [ 45   0   0]]
              precision    recall  f1-score   support

           0       0.56      1.00      0.72       112
           1       0.00      0.00      0.00        44
           2       0.00      0.00      0.00        45

    accuracy                           0.56       201
   macro avg       0.19      0.33      0.24       201
weighted avg       0.31      0.56      0.40       201
```

# Random Forest Classification

- As the name implies, a random forest is made up of a huge number of individual decision trees that work together as an ensemble.
- Each tree in the random forest produces a class prediction, and the class with the most votes becomes the prediction of our model.

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()

model4 = rfc.fit(xtrain, ytrain)
prediction4 = model4.predict(xtest)

print('Accuracy on training data: {:,.3f}'.format(rfc.score(xtrain,ytrain)))
print('Accuracy on test data: {:,.3f}'.format(rfc.score(xtest,ytest)))
```

```
<ipython-input-11-15d91e47bd2d>:97: DataConversionWarning: A column-vector y was passed when a
  model4 = rfc.fit(xtrain, ytrain)
Accuracy on training data: 1.000
Accuracy on test data: 0.980
```

```
accuracy_score(ytest, prediction4)
```

```
<ipython-input-12-31556c19d0b7>:97: DataConversionWarning: A column-vector y was passed wher
  model4 = rfc.fit(xtrain, ytrain)
0.9950248756218906
```

```
print(confusion_matrix(ytest, prediction4))
print(classification_report(ytest, prediction4))
```

```
<ipython-input-13-140230a405dc>:97: DataConversionWarning: A column-vector y was passed
  model4 = rfc.fit(xtrain, ytrain)
[[121   0   0]
 [  0  44   1]
 [  0   0  35]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       121
           1       1.00      0.98      0.99        45
           2       0.97      1.00      0.99        35

    accuracy                           1.00       201
   macro avg       0.99      0.99      0.99       201
weighted avg       1.00      1.00      1.00       201
```

# Logistic Regression Classification

- This algorithm is used to predict a binary outcome.
- The binary outcome is determined by analysing independent factors, with the findings falling into one of two groups.
- It is formulated as **P(Y=1 | X) OR P(Y=0 | X)** .
- This can be then used to calculate the probability of the variable as 0 or 1 or on a scale in between.

```
#Logistic Regression Classifier
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

model5 = lr.fit(xtrain, ytrain)
prediction5 = model5.predict(xtest)

print('Accuracy on training data: {:,.3f}'.format(lr.score(xtrain,ytrain)))
print('Accuracy on test data: {:,.3f}'.format(lr.score(xtest,ytest)))
```

```
Accuracy on training data: 0.965
Accuracy on test data: 0.940
```

```
accuracy_score(ytest, prediction5)
```

```
0.9751243781094527
```

```
print(confusion_matrix(ytest, prediction5))
print(classification_report(ytest, prediction5))
```

```
[[94  1  8  0]
 [ 2 45  3  0]
 [ 0  0 47  0]
 [ 0  1  0  0]]
              precision    recall  f1-score   support

           0       0.98      0.91      0.94       103
           1       0.96      0.90      0.93        50
           2       0.81      1.00      0.90        47
           3       0.00      0.00      0.00         1

    accuracy                           0.93       201
   macro avg       0.69      0.70      0.69       201
weighted avg       0.93      0.93      0.92       201
```

**Key Findings :**

- K-Nearest Neighbour Classifier accuracy is : 97.51 %
- Naive Bayes Classifier accuracy is: 89.05 %
- Support Vector Machine Classifier accuracy is: 67.66 %
- Logistic Regression Classifier accuracy is : 97.51 %
- Random Forest Classifier accuracy is : 99.50 %

So **Random Forest Classifier accuracy** is giving the best accuracy on data with a value of 99.50 **%**.

## Model Evaluation

We have then evaluated our model which is using Logistic Regression classifier by calculating the Confusion matrix, Precision, F1- Score, and Recall for all four classes.

```
print(confusion_matrix(ytest, prediction5))
print(classification_report(ytest, prediction5))
```

```
[[94  1  8  0]
 [ 2 45  3  0]
 [ 0  0 47  0]
 [ 0  1  0  0]]
              precision    recall  f1-score   support

           0       0.98      0.91      0.94       103
           1       0.96      0.90      0.93        50
           2       0.81      1.00      0.90        47
           3       0.00      0.00      0.00         1

    accuracy                           0.93       201
   macro avg       0.69      0.70      0.69       201
weighted avg       0.93      0.93      0.92       201
```

**The Confusion Matrix :**

Class 0 : Allow
Class 1 : Deny
Class 2 : Drop
Class 3 : Reset-both

| | Predicted Class | | |
|---|---|---|---|
| Actual Class | | Class = 0 | Class = 1 |
| | Class = 0 | True Positive (**TP**) | False Negative (**FN**) |
| | Class = 1 | False Positive (**FP**) | True Negative (**TN**) |

**Accuracy**
Accuracy is the metric for the % of correct prediction. It's simply the number of successfully anticipated observations divided by the total number of observations.

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$

**Precision**
Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. That is, how many Class 0 predictions did we make out of all the Class 0 predictions. A low false positive rate is related to high precision.

$$Precision = \frac{TP}{TP + FP}$$

**Recall**
Recall is the ratio of correctly predicted positive observations to all observations in the actual class. The term "recall" is frequently used to refer to the sensitivity or "true positive rate" of a test.

$$Recall = \frac{TP}{TP + FN}$$

**F1-score**
F1-score is a harmonic mean of Precision and Recall, and so it gives a combined idea about these two metrics. When Precision equals Recall, it reaches its peak.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

**Precision for all the classes are as follows:**

Class 0 : 0.98
Class 1 : 0.96
Class 2 : 0.81
Class 3 : 0.00

## Inferences

We can see from the above calculations that Precision, Recall, and F1-score for class-0 is higher than any other class. So it can be observed that class-0 is classified in a better and correct way than other classes in this dataset.

## Final Code

The code for this project is written in python using some of the most used libraries such as pandas, numpy, matplotlib, seaborn, plotly on Google Colab.

Which is uploaded on this Github repository
https://github.com/MuskanSingla18/ML-Classification-on-Internet-Firewall-Dataset.git