

Configuration Manual

MSc Internship
MSc in Cyber Security

Sumanth Kumar Alladi
Student ID: X18108377

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland
MSc Project Submission Sheet



School of Computing

Student Name: Sumanth Kumar Alladi

Student ID: X18108377

Programme: MSc in Cyber Security

Year: 2020

Module: Academic Internship

Lecturer: Michael Pantridge

Submission

Due Date: 28/09/2020

Project Title:

Effectively improving the efficiency and performance of an intrusion detection system using hybrid machine learning models

Word Count: 880

Page Count: 7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature:

Date:

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

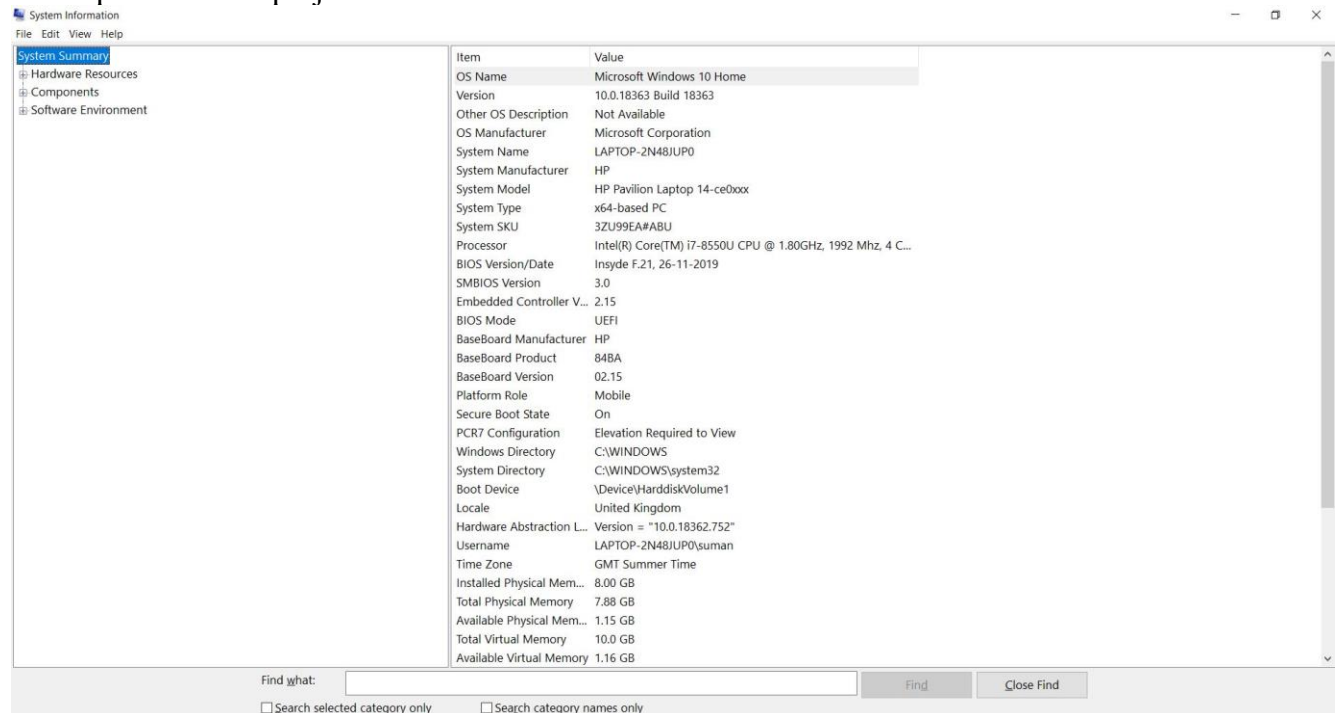
Sumanth Kumar Alladi
Student ID: X18108377

1 Introduction

This configuration manual was created as a guide for the setup of the system and the environment that was used for the development and the execution of this thesis program titled “-----”. The configuration and setup of the both, the hardware side and the software side has been explained below, with the procedures on how to use the program was explained.

2 System Requirements

In the below figure we can observe the hardware configuration of the system that was used to develop and run the project.

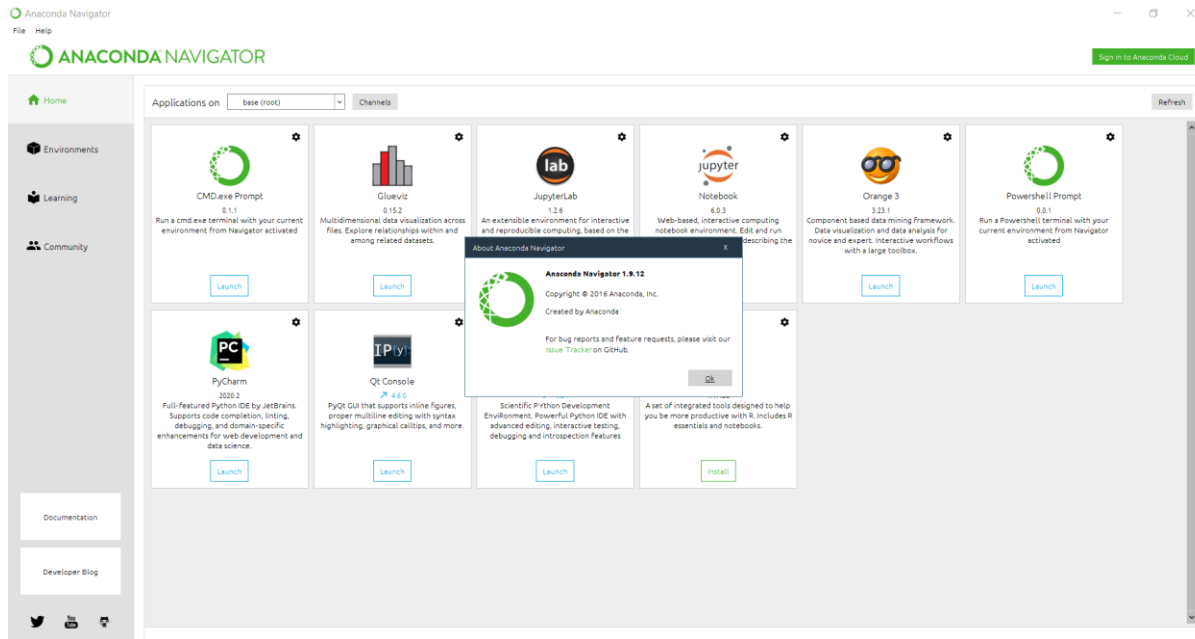


3 Softwares Used

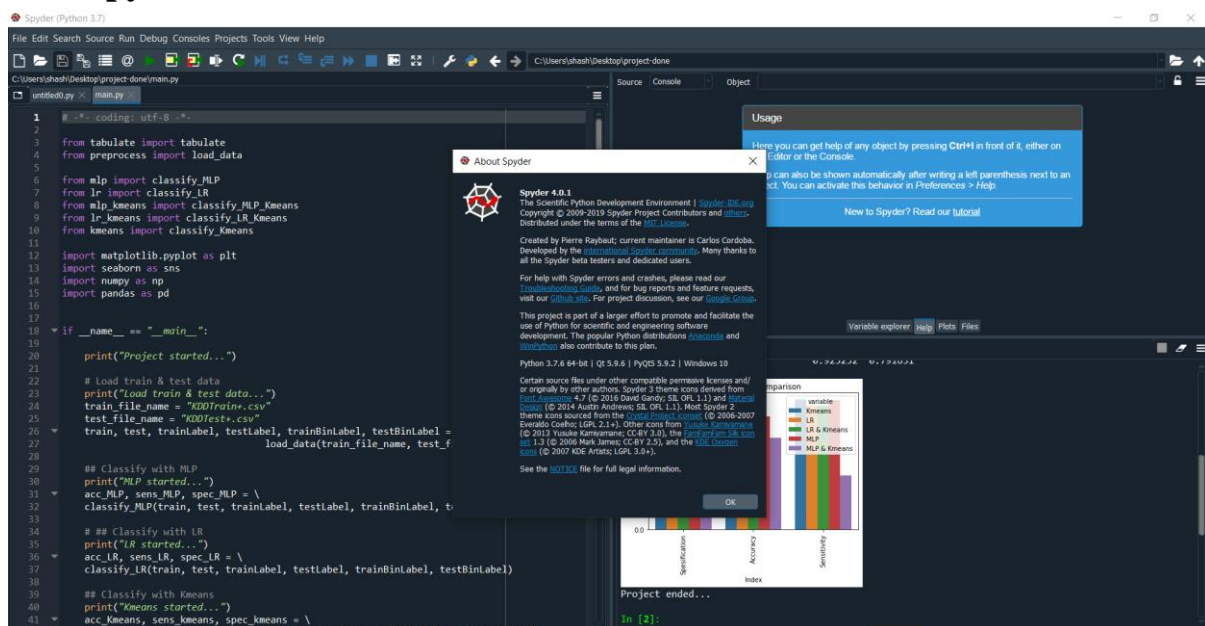
For the development of the project the softwares that were used are:

3.1 Anaconda Navigator

I have installed Anaconda Navigator version 1.9.12 to setup an environment which can use multiple types of IDE environment softwares.



3.2 Spyder



Spyder is an Python Development Environment software which contains a powerful Python editor. This software has been first installed on Anaconda Navigator and then has been launched on it. The version use for this project is 4.0.1.

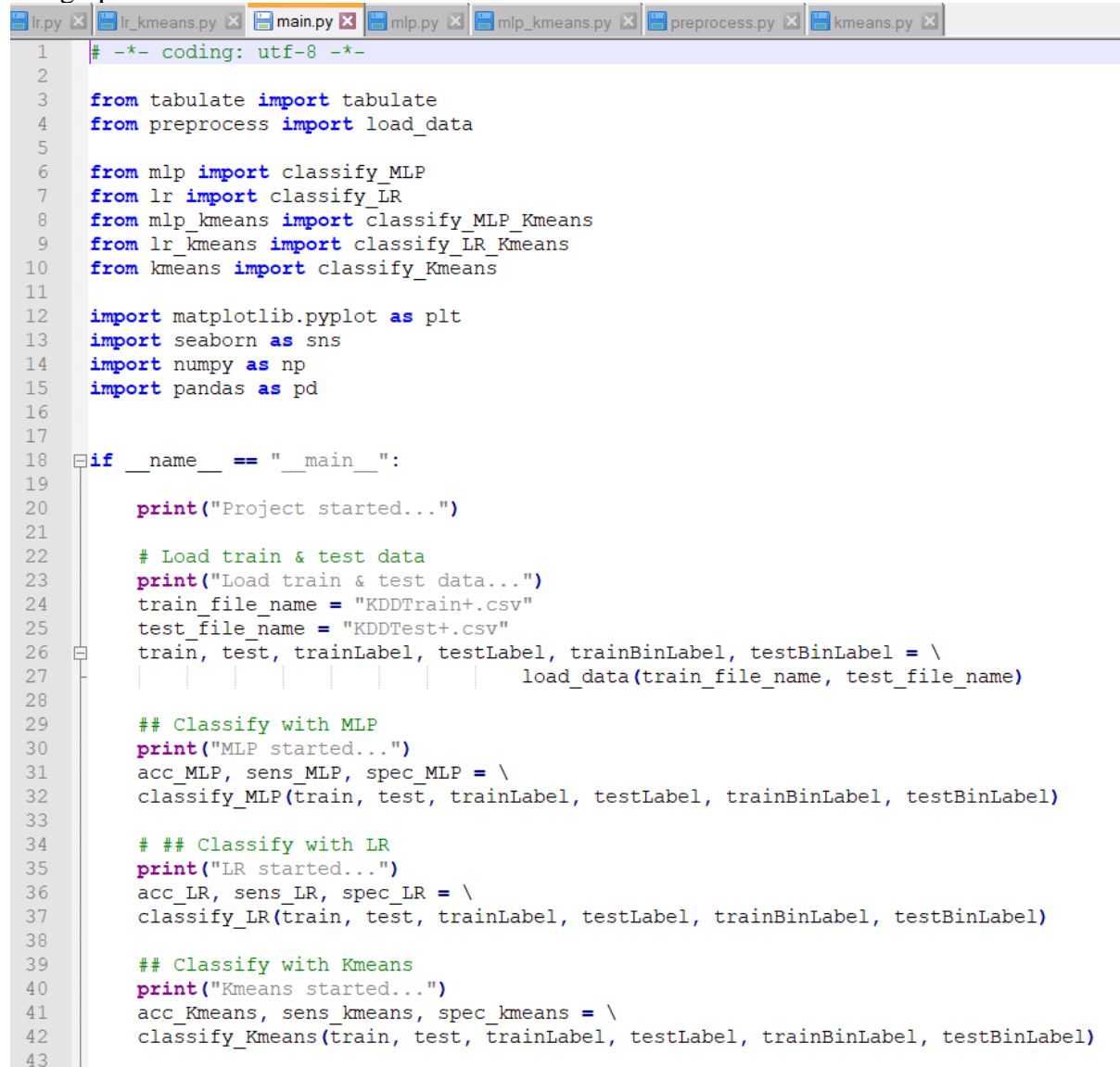
Package named 'tabulate' was installed on the Anaconda Navigator through the command line prompt. This was done so that the graph generated by the project can be executed and viewed. The command used for the installation of the package was "pip install tabulate". After this I was able to run the project by executing the main.py file.

4 Files Executed

The project contains 11 files that needs to be executed, they are as follows:

4.1 Main.py

This is the file that basically executes the whole project. In this the testing and training datasets are sent to preprocess.py file which gets loaded to it. Then the values received from the pre-processing is then sent to the either the hybrid model or the individual model. It depends on which model is getting executed. And lastly, the results are displayed in a tabular and graphical form.



```
1 # -*- coding: utf-8 -*-
2
3 from tabulate import tabulate
4 from preprocess import load_data
5
6 from mlp import classify_MLP
7 from lr import classify_LR
8 from mlp_kmeans import classify_MLP_Kmeans
9 from lr_kmeans import classify_LR_Kmeans
10 from kmeans import classify_Kmeans
11
12 import matplotlib.pyplot as plt
13 import seaborn as sns
14 import numpy as np
15 import pandas as pd
16
17
18 if __name__ == "__main__":
19
20     print("Project started...")
21
22     # Load train & test data
23     print("Load train & test data...")
24     train_file_name = "KDDTrain+.csv"
25     test_file_name = "KDDTest+.csv"
26     train, test, trainLabel, testLabel, trainBinLabel, testBinLabel = \
27         load_data(train_file_name, test_file_name)
28
29     ## Classify with MLP
30     print("MLP started...")
31     acc_MLP, sens_MLP, spec_MLP = \
32         classify_MLP(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel)
33
34     # ## Classify with LR
35     print("LR started...")
36     acc_LR, sens_LR, spec_LR = \
37         classify_LR(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel)
38
39     ## Classify with Kmeans
40     print("Kmeans started...")
41     acc_Kmeans, sens_kmeans, spec_kmeans = \
42         classify_Kmeans(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel)
43
```

4.2 Preprocess.py

Here the feature selection, labelling and data conversion to the nominal values process takes place. First, the file takes all the data from the dataset and maps different types of attacks.

Necessary feature required by the model is considered and all the other data is dropped. This code was written by referring it from the code present on GitHub website.

```

1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4  import numpy as np
5  from sklearn.preprocessing import LabelEncoder
6
7  # Load train and test data
8  def load_data(train_file_name, test_file_name):
9      ## Loads the nsl-kdd train and test data
10     train = pd.read_csv(train_file_name, header=None)
11     test = pd.read_csv(test_file_name, header=None)
12
13
14     ## Reads "Field Names.csv". Use this to set the names of train and test data columns
15     columns = pd.read_csv('Field Names.csv', header=None)
16     columns.columns = ['name', 'type']
17     train.columns = columns['name']
18     test.columns = columns['name']
19
20
21     ## Read Attack Types.csv
22     # Use this to create a mapping from attack types to final labels (Normal, Dos, R2L, Prob, U2R)
23     attackType = pd.read_csv('Attack Types.csv', header=None)
24     attackType.columns = ['Name', 'Type']
25     attackMap = {}
26
27     # Creates attackMap map which contains a mapping between attack type and the final label
28     for i in range(len(attackType)):
29         attackMap[attackType['Name'][i]] = attackType['Type'][i]
30
31
32     ## Add a new variable called 'label' which contains the final label
33     train['label'] = train['attack_type'].map(attackMap)
34     test['label'] = test['attack_type'].map(attackMap)
35
36
37     ## The variable 'label' is stored in different variables
38     # This is required to keep the dependent variable separate from the independent variable
39     trainLabel = train['label']
40     testLabel = test['label']
41     trainBinLabel = (np.array(trainLabel) == 'normal').astype(np.int)
42     testBinLabel = (np.array(testLabel) == 'normal').astype(np.int)
43
44

```

4.3 lr.py

Here, Logistic Regression model has been executed. The components such as accuracy, sensitivity and specification has been calculate using confusion matrix.

```

1  from sklearn.metrics import accuracy_score, confusion_matrix
2  from sklearn.linear_model import LogisticRegression
3
4
5  ## Classify Logistic Regression
6  def classify_LR(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel):
7
8      ## Training RF model
9      model = LogisticRegression(max_iter=200, C=0.1, random_state=123)
10     model.fit(train, trainBinLabel)
11
12
13     ## Prediction of testing data using trained model
14     pred = model.predict(test)
15
16
17     ## Calculating performance statistics
18     # Calculate accuracy
19     acc = accuracy_score(y_pred=pred, y_true=testBinLabel)
20
21     # Calculate confusion matrix
22     con_matrix = confusion_matrix(y_pred=pred, y_true=testBinLabel)
23
24     # Calculate TN, FN, TP, FP
25     TN = con_matrix[0][0]
26     FN = con_matrix[1][0]
27     TP = con_matrix[1][1]
28     FP = con_matrix[0][1]
29
30     # Calculate detection rate and FAR
31     sens = TP / (TP + FN)
32     spec = TN / (TN + FP)
33
34     return acc, sens, spec
35
36

```

4.4 lr_kmeans.py

In this file the hybrid model of Logistic Regression and K-Means model has been executed with the calculation of the performance components similarly done in the previous file.

```
1 from sklearn.metrics import accuracy_score, confusion_matrix
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.cluster import KMeans
4 import numpy as np
5
6
7 def classify_LR_Kmeans(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel):
8
9     ## Training RF model
10    model = LogisticRegression(max_iter=200, C=0.1, random_state=123)
11    model.fit(train, trainBinLabel)
12
13    ## Training Kmeans model
14    kmeans = KMeans(n_clusters=2, random_state=0).fit(test)
15
16    ## Prediction of testing data using trained model
17    pred_kmeans = 1 - kmeans.labels_
18
19    ## Prediction of testing data using trained model
20    pred_lr = model.predict(test)
21
22
23    ## Combine RF & Kmeans
24    pred = np.zeros(len(pred_lr))
25    for i in range(len(pred)):
26        if pred_lr[i] == 1 and pred_kmeans[i] == 1:
27            pred[i] = 1
28
29
30    ## Calculating performance statistics
31    # Calculate accuracy
32    acc = accuracy_score(y_pred=pred, y_true=testBinLabel)
33
34    # Calculate confusion matrix
35    con_matrix = confusion_matrix(y_pred=pred, y_true=testBinLabel)
36
37    # Calculate TN, FN, TP, FP
38    TN = con_matrix[0][0]
39    FN = con_matrix[1][0]
40    TP = con_matrix[1][1]
41    FP = con_matrix[0][1]
42
43    # Calculate Sensitivity and Specificity
```

4.5 mlp.py

Multi-Layer Perception model has been executed to get the performance outputs which has been calculate by using the confusion matrix.

```
1 from sklearn.metrics import accuracy_score, confusion_matrix
2 from sklearn.neural_network import MLPClassifier
3
4
5 ## Classify MLP
6 def classify_MLP(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel):
7
8     ## Training RF model
9     model = MLPClassifier(hidden_layer_sizes=(100, 25), activation='relu', random_state=123)
10    model.fit(train, trainBinLabel)
11
12
13    ## Prediction of testing data using trained model
14    pred = model.predict(test)
15
16
17    ## Calculating performance statistics
18    # Calculate accuracy
19    acc = accuracy_score(y_pred=pred, y_true=testBinLabel)
20
21    # Calculate confusion matrix
22    con_matrix = confusion_matrix(y_pred=pred, y_true=testBinLabel)
23
24    # Calculate TN, FN, TP, FP
25    TN = con_matrix[0][0]
26    FN = con_matrix[1][0]
27    TP = con_matrix[1][1]
28    FP = con_matrix[0][1]
29
30    # Calculate detection rate and FAR
31    sens = TP / (TP + FN)
32    spec = TN / (TN + FP)
33
34    return acc, sens, spec
35
```


4.6 mpl_kmeans.py

The hybrid model of MLP and K-Means has been executed with the calculation of performance components.

```
1 from sklearn.metrics import accuracy_score, confusion_matrix
2 from sklearn.neural_network import MLPClassifier
3 from sklearn.cluster import KMeans
4 import numpy as np
5
6
7 def classify_MLP_Kmeans(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel):
8
9     ## Training RF model
10    model = MLPClassifier(hidden_layer_sizes=(100, 25), activation='relu', random_state=123)
11    model.fit(train, trainBinLabel)
12
13    ## Training Kmeans model
14    kmeans = KMeans(n_clusters=2, random_state=0).fit(train)
15
16    ## Prediction of testing data using trained model
17    pred_kmeans = 1 - kmeans.labels_
18
19    ## Prediction of testing data using trained model
20    pred_lr = model.predict(test)
21
22
23    ## Combine RF & Kmeans
24    pred = np.zeros(len(pred_lr))
25    for i in range(len(pred)):
26        if pred_lr[i] == 1 and pred_kmeans[i] == 1:
27            pred[i] = 1
28
29
30    ## Calculating performance statistics
31    # Calculate accuracy
32    acc = accuracy_score(y_pred=pred, y_true=testBinLabel)
33
34    # Calculate confusion matrix
35    con_matrix = confusion_matrix(y_pred=pred, y_true=testBinLabel)
36
37    # Calculate TN, FN, TP, FP
38    TN = con_matrix[0][0]
39    FN = con_matrix[1][0]
40    TP = con_matrix[1][1]
41    FP = con_matrix[0][1]
```

4.7 kmeans.py

The performance components has been calculate for the K-Means model in this file.

```
1 from sklearn.metrics import accuracy_score, confusion_matrix
2 from sklearn.cluster import KMeans
3
4 def classify_Kmeans(train, test, trainLabel, testLabel, trainBinLabel, testBinLabel):
5
6     ## Clustering Kmeans model
7     kmeans = KMeans(n_clusters=2, random_state=0).fit(test)
8
9
10    ## Prediction of testing data using trained model
11    pred = 1 - kmeans.labels_
12
13
14    ## Calculating performance statistics
15    # Calculate accuracy
16    acc = accuracy_score(y_pred=pred, y_true=testBinLabel)
17
18    # Calculate confusion matrix
19    con_matrix = confusion_matrix(y_pred=pred, y_true=testBinLabel)
20
21    # Calculate TN, FN, TP, FP
22    TN = con_matrix[0][0]
23    FN = con_matrix[1][0]
24    TP = con_matrix[1][1]
25    FP = con_matrix[0][1]
26
27    # Calculate detection rate and FAR
28    sens = TP / (TP + FN)
29    spec = TN / (TN + FP)
30
31    return acc, sens, spec
```

4.8 Attack Types.csv

In this file, different types of attacks have been grouped together into four main attack categories.

4.9 Field Names.csv

This file helps the model in the feature selection process as it contains the required field that must be taken into consideration and all other data can be dropped.

4.10 KDDTest+.csv

20% of NSL-KDD dataset has been taken as a testing data. This file helps the model to test itself to see if its able to detect the anomalies in the network traffic.

4.11 KDDTrain+.csv

This file has been used to train the model. 80% of the data from the NSL-KDD dataset has been taken.