MINI PROJECT - II

On

NETWORK CONGESTION ANALYSIS AND ANOMALY DETECTION

Submitted by

SHANI KUMAR GUPTA
171500308
MUDIT MANGAL
171500194

Department of Computer Engineering & Applications
Institute of Engineering & Technology

GLA University
Mathura- 281406, INDIA

2020

ऋते ज्ञानान्न मुक्तिः

Department of Computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha, Mathura – 281406

Declaration

I hereby declare that the work which is being presented in the Mini Project II “**Network Congestion Analysis and Anomaly Detection”** submitted to GLA University, Mathura is a record of an original work done by us under the guidance of **Mr. Vinay Agarwal**.

Name of Candidate:

Shani Kumar Gupta (171500308)

Mudit Mangal (171500194)

Course: B. Tech (CSE)

Year: 3rd Year

Semester: VIth Semester

Acknowledgement

I thank the almighty for giving me the courage and perseverance in completing the project.
This project itself is acknowledgement for all those people who have given me their heartfelt cooperation in making this project a grand success. I extend my sincere thanks to Mr. Vinay Agrawal, Asst. Professor at GLA University, Mathura for providing valuable guidance at every stage of this project work. I am profoundly grateful towards the unmatched services rendered by him. Lat but not Least, I would like to express to deep sense of gratitude and earnest thanks to my dear parents for their moral support and heartful cooperation in doing main project.

Abstract

Although the discovery and analysis of Network Congestion and the complex communication between the Users and complex data transfer between the network. It is more complex to analysis the network and their behaviour as well as the network will also produce the huge amount of data which is critical to analyse. So the main concern by the time how we can analyse the Network Congestion because it is the biggest problem by the time and most of the users phase the problem due to the congestion in the network so basically in this project we analyse the network data to find out the congestion in the network and based on the traffic in the network we will predict the cost of the required bandwidth to decrease the congestion in the particular network.

Contents

Chapter 1

Introduction

1.1 Motivation

In the real world, Congestion in the Network is the biggest problem phased by the telecom users as well as the service provider. Most of the time we analyse that the network is so congested due to which proper communication is not possible and many times the call is hang off due to the congestion which create the difficulty for the users so we decide to analysis the factor of Network Congestion and after that we will predict the cost of required bandwidth so that the congestion in the network will be reduce and anomaly in the network is one of the cause of network performance.

1.2 Objective

Network Congestion Analysis in which we will analyse the different factors of congestion, how to reduce the congestion in the network as well as analyse the security aspects of the network so that we can enhance the security to protect the data packets. In this project we analyse the data set which consists of date and traffic and we use Arima model to analyse the dataset and find the traffic and based on the traffic we will find out the bandwidth required cost so that service provider can easily reduce the congestion and after that we analyse and detect the anomaly in the network to improve the security aspects.

Chapter 2

Software Requirement

2.1 Jupyter Notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

2.2 MongoDB

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License.

2.3 Brackets

Brackets is a source code editor with a primary focus on web development. Created by Adobe Systems, it is free and open-source software licensed under the MIT License, and is currently maintained on GitHub by Adobe and other open-source developers. It is written in JavaScript, HTML and CSS.

Chapter 3

Technologies Requirement

3.1 Python

Python is an interpreted, high-level, general-purpose programming language created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

3.2 MongoDB

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License

3.3 HTML

Hypertext Markup Language is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets and scripting languages such as JavaScript.

3.4 CSS

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

3.5 **Bootstrap**

Bootstrap is a free and open source front end development framework for the creation of websites and web apps. The Bootstrap framework is built on HTML, CSS, and JavaScript (JS) to facilitate the development of responsive, mobile-first sites and apps.

Responsive design makes it possible for a web page or app to detect the visitor’s screen size and orientation and automatically adapt the display accordingly; the mobile first approach assumes that smartphones, tablets and task-specific mobile apps are employees' primary tools for getting work done and addresses the requirements of those technologies in design.

3.5 **JavaScript**

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

Chapter 4

Network Congestion Analysis

4.1 **Network Traffic Analysis and visualizing the data**

In Network Congestion Analysis Project basically we used Time Series algorithm that is Arima Model by the help of this model we predict the traffic in the network and after finding the traffic we predict the cost based on the required bandwidth which help as to analyse the congestion and the factors of traffic in the network. So, in the series of analysing the dataset we follow some step by the help of Arima model by using Python as a language to analyse so the step is given below-

Time Series Analysis:

Time series data often arise when monitoring industrial processes or tracking corporate business metrics. The essential difference between modelling data via time series methods or using the process monitoring methods discussed earlier in this chapter is the following:
Time series analysis accounts for the fact that data points taken over time may have an internal structure (such as autocorrelation, trend or seasonal variation) that should be accounted for.
This section will give a brief overview of some of the more widely used techniques in the rich and rapidly growing field of time series modelling and analysis.

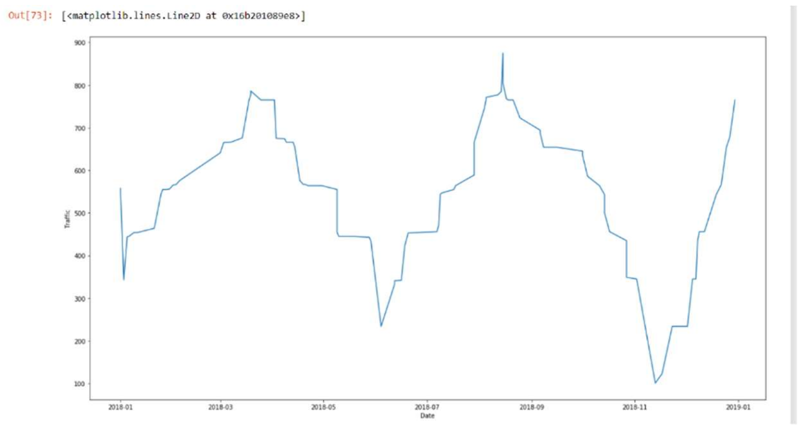
ARIMA MODEL:

ARIMA, short for 'Auto Regressive Integrated Moving Average' is actually, a class of models that 'explains' a given time series based on its own past values, that is, its own lags and the lagged forecast errors, so that equation can be used to forecast future values.

In statistics and econometrics, and in particular in time series analysis, an autoregressive integrated moving average model is a generalization of an autoregressive moving average model. Both of these models are fitted to time series data either to better understand the data or to predict future points in the series.

4.1.1 Finding the Dataset and check the trend in the traffic

Firstly we took the dataset from Kaggle for analysis so in this dataset basically there are two columns first one is date and another one is traffic on the particular network after the we import some library of Python to analyse the dataset and after that we import the dataset with the help to MongoDB in which firstly import the dataset in MongoDB then dataset will be imported in Jupyter Notebook by the help of pymongo library so after that we apply some Python command to find weather the graph follow the particular trend or not so after analysis we got that our data follow the seasonality patter which means the traffic depend upon the season the graph is shown below-

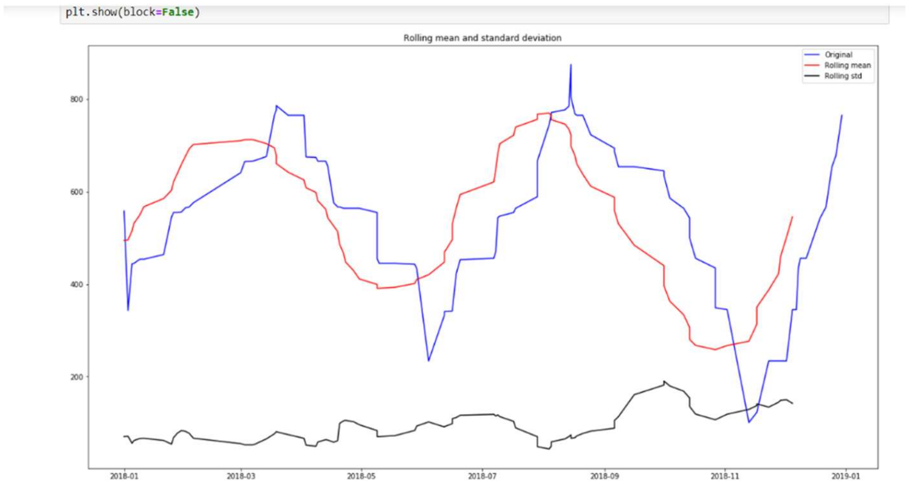


```
Out[73]: [<matplotlib.lines.Line2D at 0x16b201089e8>]
```

Fig 4.1 Relationship between Date and Traffic

4.1.2 Stationarity Checking

In Time Series Analysis firstly we check whether the data is stationary or not because if the data is not stationary we can't use the Arima Model so to check the stationarity we use the different methods to check whether the data is stationary or not so in the series of analysis firstly we use Rolling statistic and we got the graph-

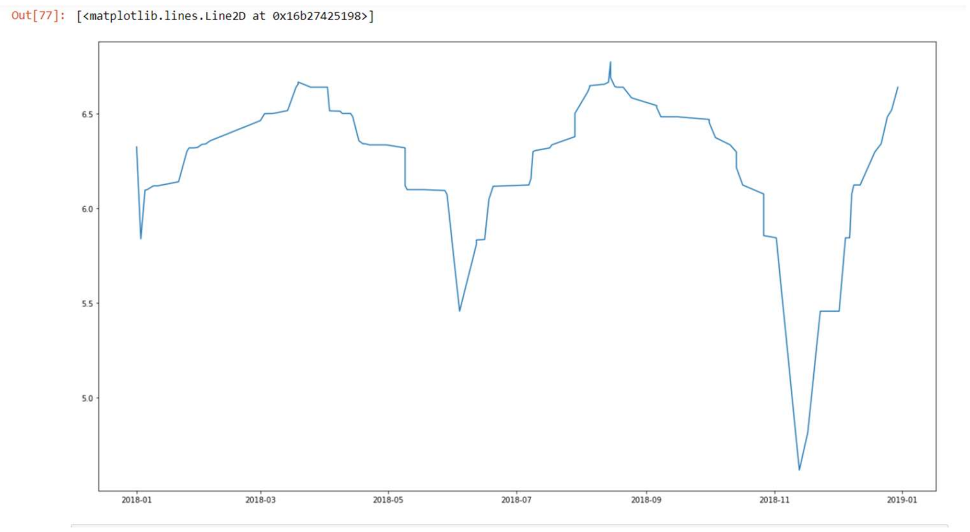


```
plt.show(block=False)
```

Fig 4.2 Rolling Statistics to check stationarity

In this we can see that the data is not stationarity which means we cannot apply the Arima model so after this we apply the different approach to convert it into stationary data.

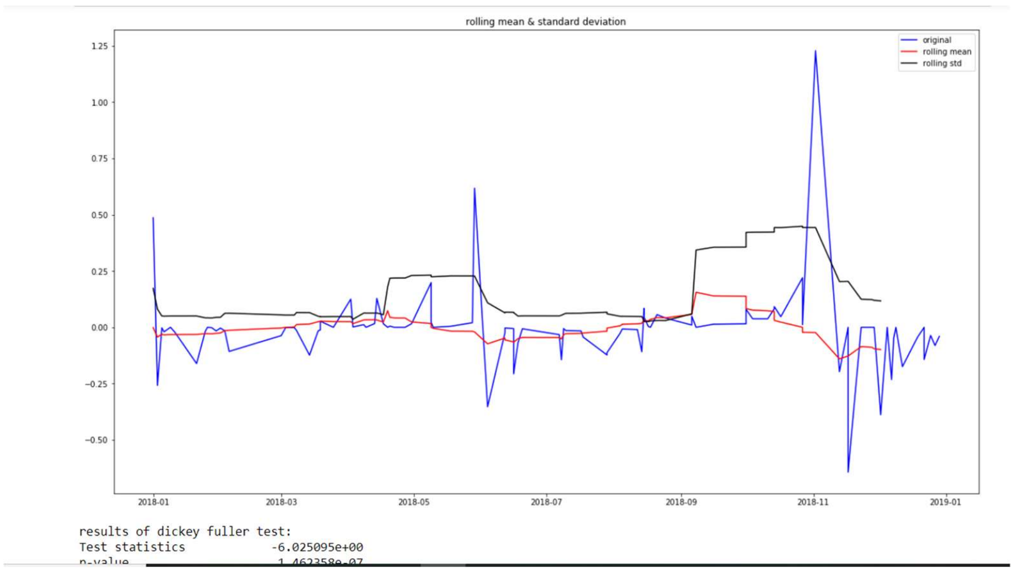
After that we apply the Dicky fuller test to check the data is stationary or not and we got the graph which show that the method is not useful to convert it into stationarity data.



```
Out[77]: [<matplotlib.lines.Line2D at 0x16b27425198>]
```

Fig 4.3 Dickey fuller test to check stationarity

After that we apply the log shifting method to convert into the stationary data so that we can apply the Arima model and find the traffic in the network after apply this approach we got the stationary graph-

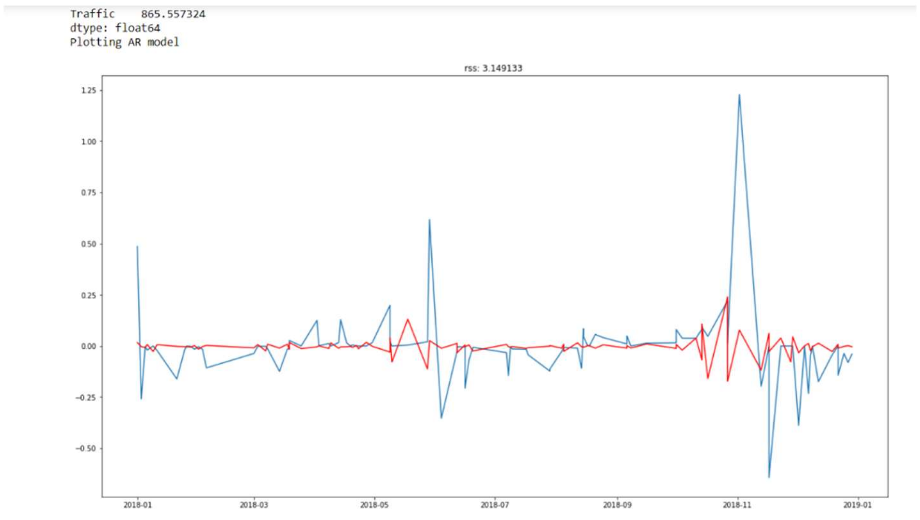


```
results of dickey fuller test:
Test statistics           -6.025095e+00
p-value                    1.462358e-07
```

Fig 4.4 Stationary Graph

4.1.2 Apply Arima Model to get Traffic

So, after the stationarity check we are going to apply the Arima Model to get the traffic in the network so that we can predict the cost on the basic of traffic and we got the following graph-



```
Traffic    865.557324
dtype: float64
Plotting AR model
```

Fig 4.5 Traffic Analysis using Arima Model

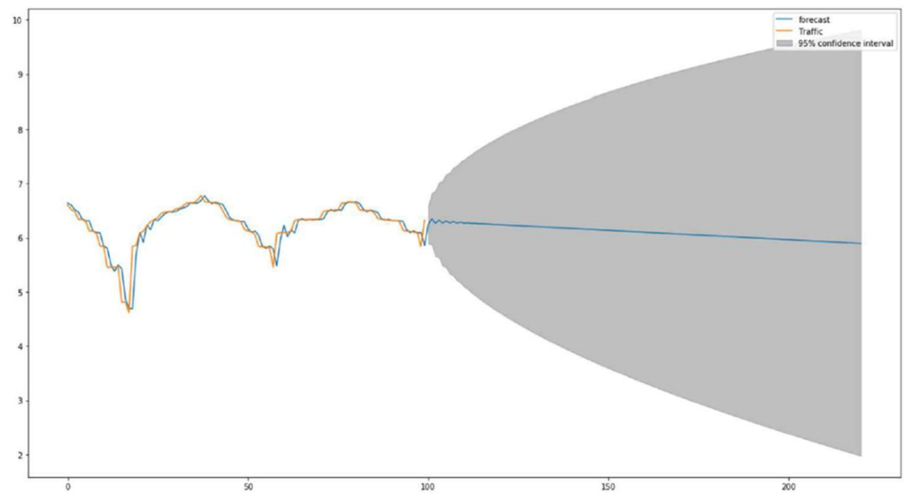


Fig 4.6 Future Traffic Prediction

4.3 **Cost Prediction based on Bandwidth**

So, after using Arima model we got the traffic in the network so if service provider demand for the bandwidth so that they can reduce the congestion in the network so they have to pay the required cost so in this project we predict the cost based on the required bandwidth so we use the following formula to predict the cost which is-

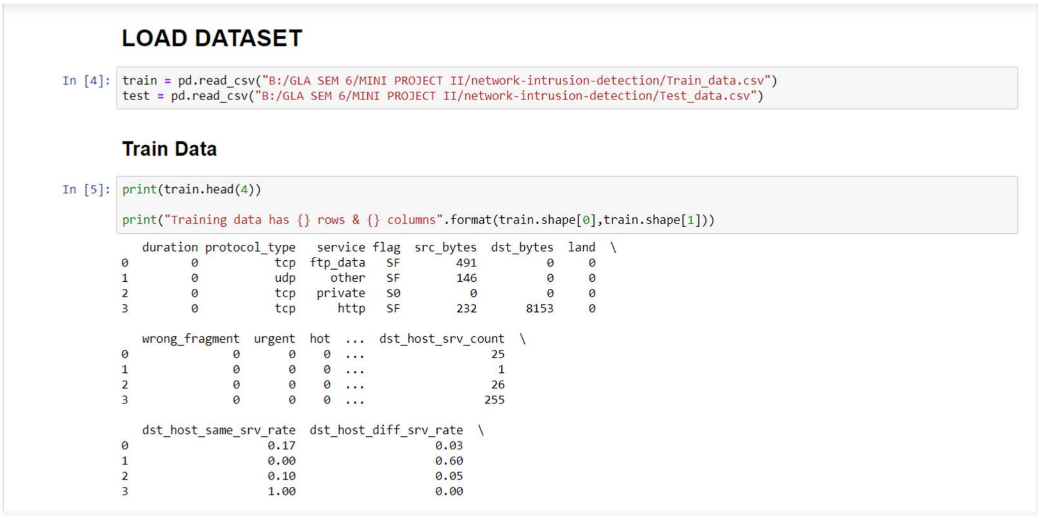
Cost = Traffic * Per bandwidth cost

Chapter 5

Anomaly Detection

5.1 Loading of Dataset and Exploratory Analysis

This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between bad" connections, called intrusions or attacks, and good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.



LOAD DATASET

```
In [4]: train = pd.read_csv("B:/GLA SEM 6/MINI PROJECT II/network-intrusion-detection/Train_data.csv")
        test = pd.read_csv("B:/GLA SEM 6/MINI PROJECT II/network-intrusion-detection/Test_data.csv")
```

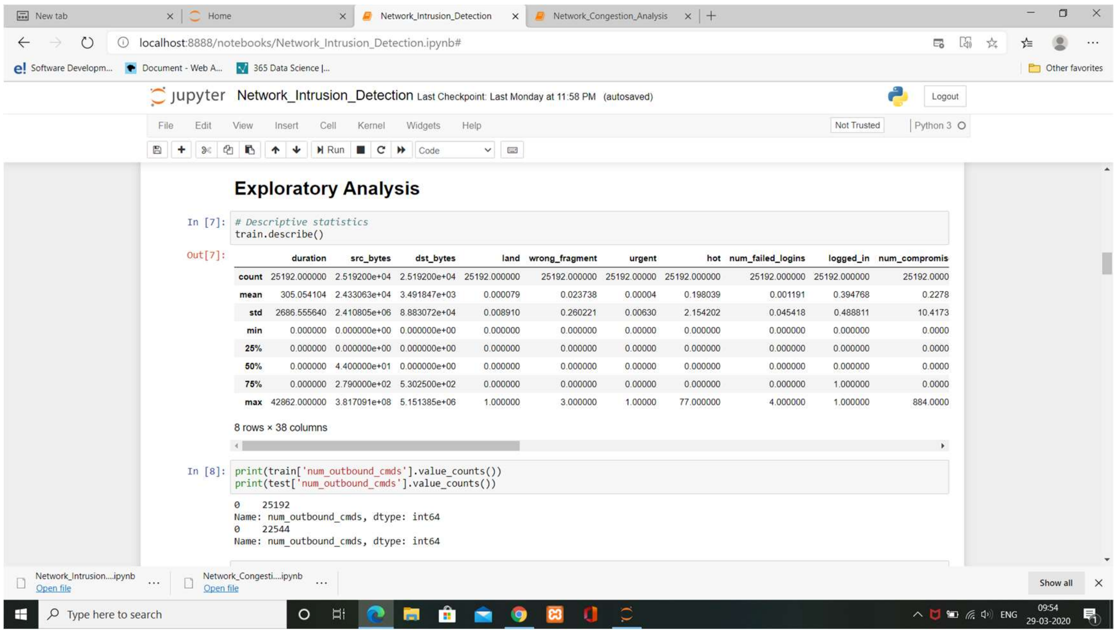
Train Data

```
In [5]: print(train.head(4))

        print("Training data has {} rows & {} columns".format(train.shape[0],train.shape[1]))
```

```
   duration protocol_type   service flag  src_bytes  dst_bytes  land  \
0         0           tcp  ftp_data   SF        491          0     0
1         0           udp     other   SF        146          0     0
2         0           tcp   private   S0          0          0     0
3         0           tcp      http   SF        232       8153     0

   wrong_fragment  urgent  hot  ...  dst_host_srv_count  \
0               0       0    0  ...                  25
1               0       0    0  ...                   1
2               0       0    0  ...                  26
3               0       0    0  ...                 255

   dst_host_same_srv_rate  dst_host_diff_srv_rate  \
0                    0.17                    0.03
1                    0.00                    0.60
2                    0.10                    0.05
3                    1.00                    0.00
```

Fig 5.1 Loading Dataset

Exploratory Analysis:


Network_Intrusion_Detection
localhost:8888/notebooks/Network_Intrusion_Detection.ipynb#
jupyter Network_Intrusion_Detection Last Checkpoint: Last Monday at 11:58 PM (autosaved)

Exploratory Analysis

```
In [7]: # Descriptive statistics
        train.describe()
```

Out[7]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromis
count	25192.000000	2.519200e+04	2.519200e+04	25192.000000	25192.000000	25192.00000	25192.000000	25192.000000	25192.000000	25192.0000
mean	305.054104	2.433063e+04	3.491847e+03	0.000079	0.023738	0.00004	0.198039	0.001191	0.394768	0.2278
std	2686.555640	2.410805e+06	8.883072e+04	0.008910	0.260221	0.00630	2.154202	0.045418	0.488811	10.4173
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
50%	0.000000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
75%	0.000000	2.790000e+02	5.302500e+02	0.000000	0.000000	0.00000	0.000000	0.000000	1.000000	0.0000
max	42862.000000	3.817091e+08	5.151385e+06	1.000000	3.000000	1.00000	77.000000	4.000000	1.000000	884.0000

8 rows × 38 columns

```
In [8]: print(train['num_outbound_cmds'].value_counts())
        print(test['num_outbound_cmds'].value_counts())

0    25192
Name: num_outbound_cmds, dtype: int64
0    22544
Name: num_outbound_cmds, dtype: int64
```

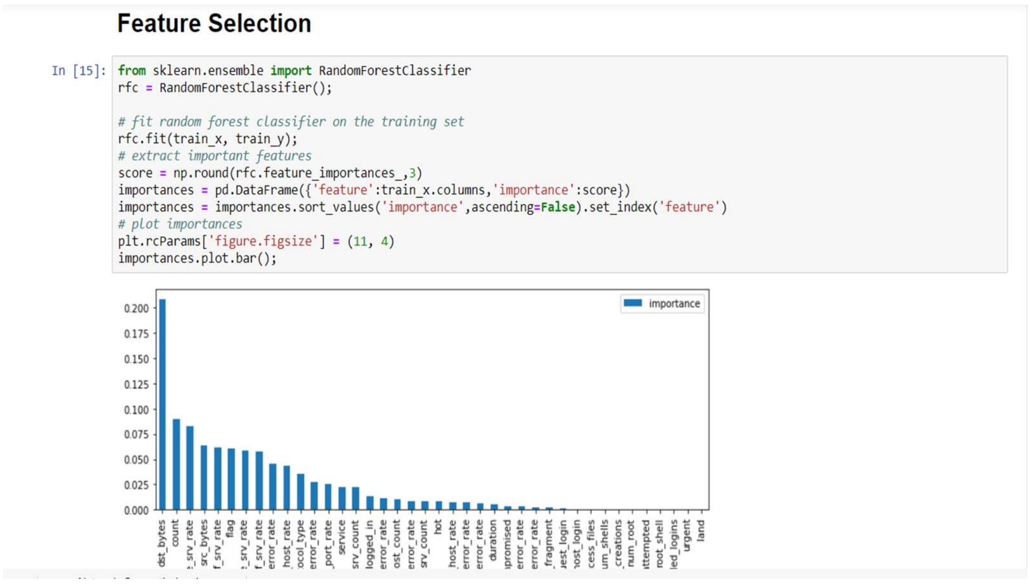
Fig 5.2 Exploratory Analysis

5.2 **Features Selection**

Random forest classifier:

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests create decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

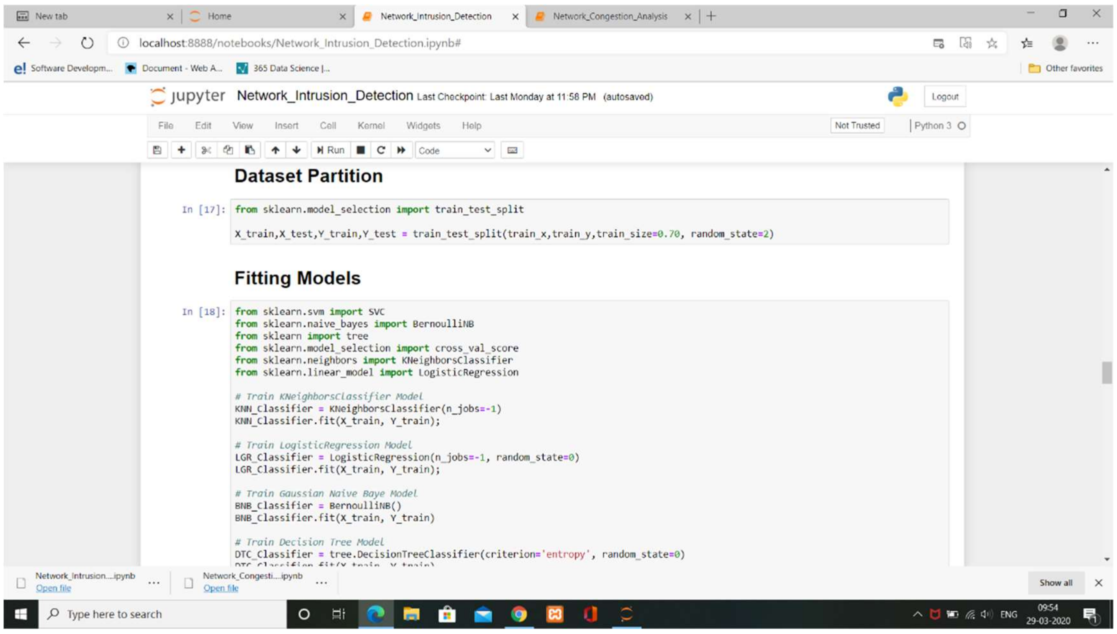

Feature Selection

```
In [15]: from sklearn.ensemble import RandomForestClassifier
         rfc = RandomForestClassifier();

         # fit random forest classifier on the training set
         rfc.fit(train_x, train_y);
         # extract important features
         score = np.round(rfc.feature_importances_,3)
         importances = pd.DataFrame({'feature':train_x.columns,'importance':score})
         importances = importances.sort_values('importance',ascending=False).set_index('feature')
         # plot importances
         plt.rcParams['figure.figsize'] = (11, 4)
         importances.plot.bar();
```

Fig 5.3 Feature Selection

5.3 **Finding Model and Validating Model**

In this analysis we select the different Machine algorithm so that we can find the suitable model for which we can analysis and validate the particular model so that we can get accurate result and detect the anomaly.


New tab
Home
Network_Intrusion_Detection
Network_Congestion_Analysis
localhost:8888/notebooks/Network_Intrusion_Detection.ipynb#
Software Developm...
Document - Web A...
365 Data Science |...
Other favorites
jupyter
Network_Intrusion_Detection Last Checkpoint: Last Monday at 11:58 PM (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
Python 3
Run
Code
Network_Intrusion....ipynb
Open file
Network_Congesti....ipynb
Open file
Show all
Type here to search
ENG
09:54
29-03-2020

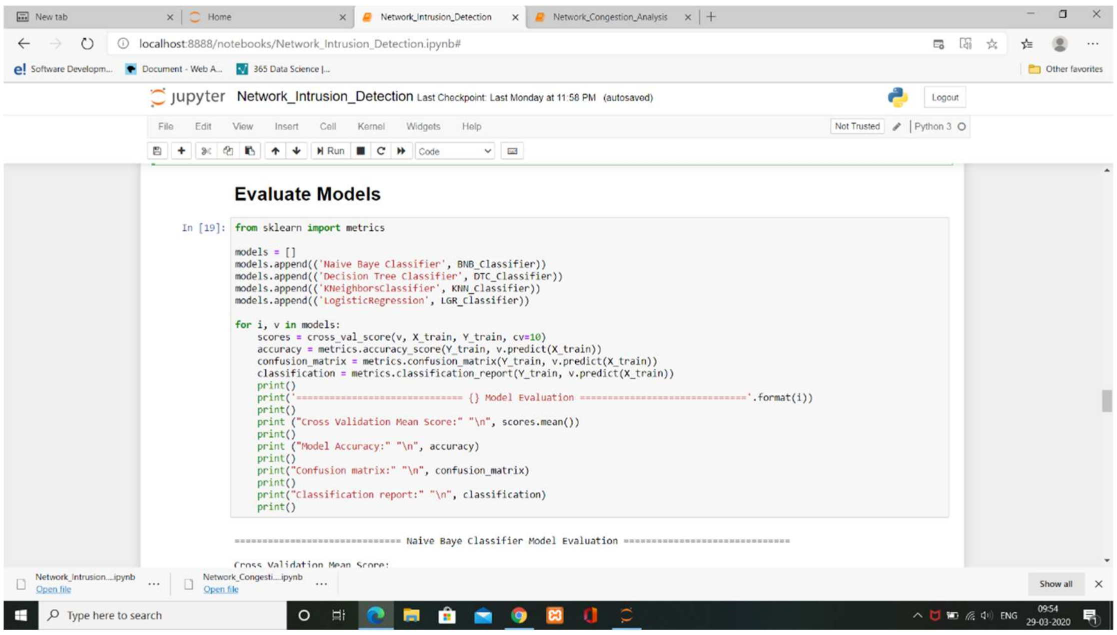
Dataset Partition

```
In [17]: from sklearn.model_selection import train_test_split

         X_train,X_test,Y_train,Y_test = train_test_split(train_x,train_y,train_size=0.70, random_state=2)
```

Fitting Models

```
In [18]: from sklearn.svm import SVC
         from sklearn.naive_bayes import BernoulliNB
         from sklearn import tree
         from sklearn.model_selection import cross_val_score
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.linear_model import LogisticRegression

         # Train KNeighborsClassifier Model
         KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
         KNN_Classifier.fit(X_train, Y_train);

         # Train LogisticRegression Model
         LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
         LGR_Classifier.fit(X_train, Y_train);

         # Train Gaussian Naive Baye Model
         BNB_Classifier = BernoulliNB()
         BNB_Classifier.fit(X_train, Y_train)

         # Train Decision Tree Model
         DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random_state=0)
```

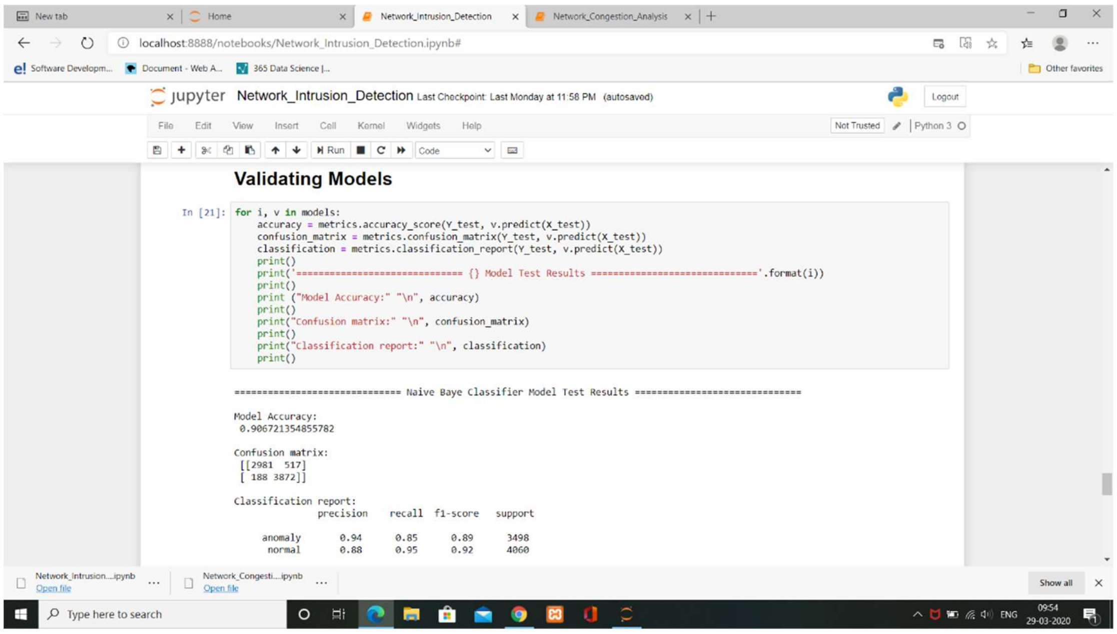
Fig 5.4 Finding Model



Evaluate Models

```
In [19]: from sklearn import metrics

         models = []
         models.append(('Naive Baye Classifier', BNB_Classifier))
         models.append(('Decision Tree Classifier', DTC_Classifier))
         models.append(('KNeighborsClassifier', KNN_Classifier))
         models.append(('LogisticRegression', LGR_Classifier))

         for i, v in models:
             scores = cross_val_score(v, X_train, Y_train, cv=10)
             accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
             confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
             classification = metrics.classification_report(Y_train, v.predict(X_train))
             print()
             print('============================== {} Model Evaluation =============================='.format(i))
             print()
             print ("Cross Validation Mean Score:" "\n", scores.mean())
             print()
             print ("Model Accuracy:" "\n", accuracy)
             print()
             print("Confusion matrix:" "\n", confusion_matrix)
             print()
             print("Classification report:" "\n", classification)
             print()

         ============================== Naive Baye Classifier Model Evaluation ==============================

         Cross Validation Mean Score:
```

Fig 5.5 Evaluate Models



Validating Models

```
In [21]: for i, v in models:
             accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
             confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
             classification = metrics.classification_report(Y_test, v.predict(X_test))
             print()
             print('============================== {} Model Test Results =============================='.format(i))
             print()
             print ("Model Accuracy:" "\n", accuracy)
             print()
             print("Confusion matrix:" "\n", confusion_matrix)
             print()
             print("Classification report:" "\n", classification)
             print()

         ============================== Naive Baye Classifier Model Test Results ==============================

         Model Accuracy:
          0.906721354855782

         Confusion matrix:
          [[2981  517]
          [ 188 3872]]

         Classification report:
                        precision    recall  f1-score   support

              anomaly       0.94      0.85      0.89      3498
               normal       0.88      0.95      0.92      4060
```

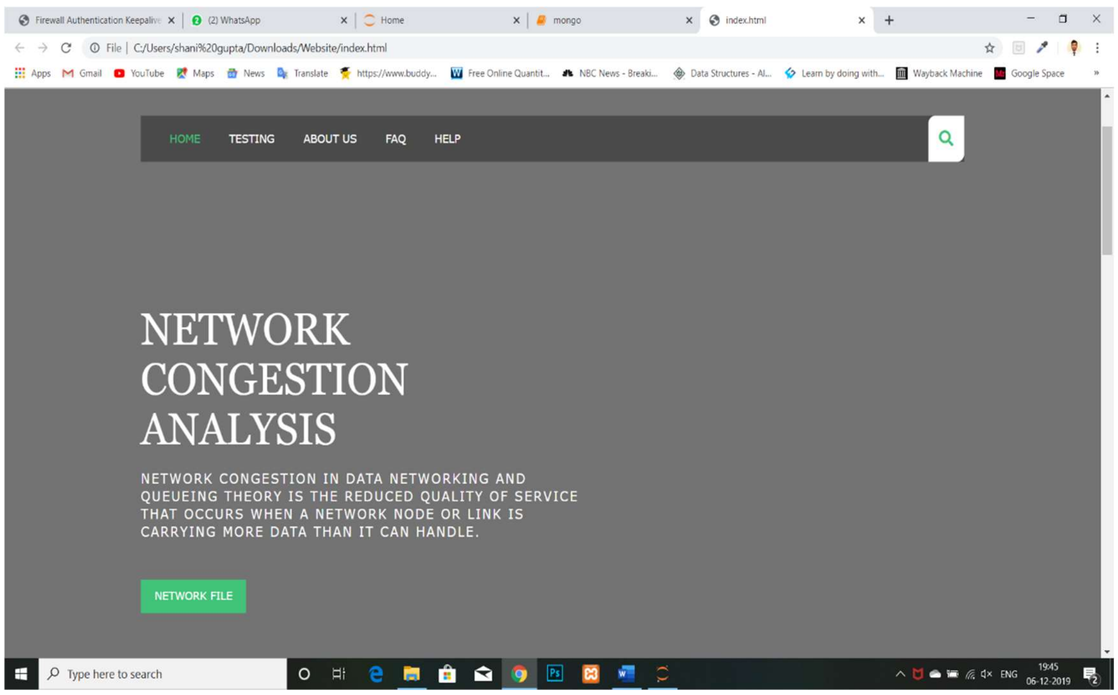
Fig 5.6 Validating Models

Chapter 6

UI Design for Network Traffic Analysis

UI designing is the main part of any project by the help of which we can show the result of our project in attractive way so that it is easy to understand so for this we design the web based portal through which the service provider will upload the traffic data and enter the per bandwidth cost so that through this web portal we can easily predict the cost and show in the visual form.

6.1 Home Page



NETWORK CONGESTION ANALYSIS

NETWORK CONGESTION IN DATA NETWORKING AND QUEUEING THEORY IS THE REDUCED QUALITY OF SERVICE THAT OCCURS WHEN A NETWORK NODE OR LINK IS CARRYING MORE DATA THAN IT CAN HANDLE.

Fig 6.1 Home Page

In the home page we provide the Network File Button so that user can easily go to the testing page to predict the cost for the required bandwidth which help the service provider to analyse it.

6.2 Testing Page

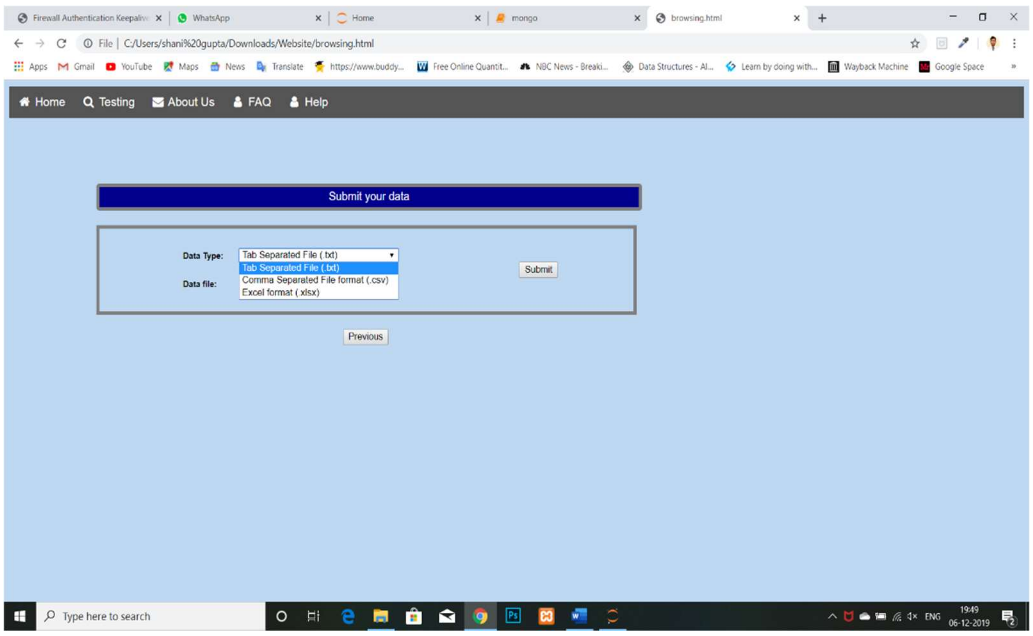


Fig 6.2 Testing Page

Chapter 7

Appendices

7.1 Network_Congestion_Analysis.ipynb

Importing the Python library:

```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize']=20,11
```

Importing the dataset file using Pymongo library:

```
import pymongo
from pymongo import MongoClient
client = MongoClient()
client = MongoClient('mongodb://localhost:27017/')
mydb = client['network']
mycol= mydb['cost']
l=[]
for i in mycol.find():
del i['_id']
l.append(dict(i))
```

Checking the trend in the dataset:

```
plt.xlabel('Date')
plt.ylabel('Traffic')
plt.plot(indexdataset)
```

Checking Stationarity in the data:

```
#rolling statistics
rolmean=indexdataset.rolling(window=12).mean()
```

```
rolstd=indexdataset.rolling(window=12).std()
rolmean.dropna(inplace=True)
rolstd.dropna(inplace=True)
print(rolmean,rolstd)
org=plt.plot(indexdataset,color='blue',label='Original')
mean=plt.plot(rolmean,color='red',label='Rolling mean')
std=plt.plot(rolstd,color='black',label='Rolling std')
plt.legend(loc='best')
plt.title('Rolling mean and standard deviation')
plt.show(block=False)
#dicky fuller test
from statsmodels.tsa.stattools import adfuller
print('results of dickey fuller test: ')
dftest=adfuller(indexdataset['Traffic'],autolag='AIC')
dfoutput=pd.Series(dftest[0:4],index=['Test statistics','p-value','lags used','no.
of observations used'])
for key,value in dftest[4].items():
dfoutput['Critical values (%s)'%key]=value
print(dfoutput)
indexdataset_logscale=np.log(indexdataset)
indexdataset_logscale.dropna(inplace=True)
plt.plot(indexdataset_logscale)
movingaverage=indexdataset_logscale.rolling(window=12).mean()
movingstd=indexdataset_logscale.rolling(window=12).std()
plt.plot(indexdataset_logscale)
plt.plot(movingaverage,color='red')
plt.plot(movingstd,color='black')
from statsmodels.tsa.stattools import adfuller
def test_stationary(timeseries):
#determine rolling statistics
movingaverage=timeseries.rolling(window=12).mean()
movingstd=timeseries.rolling(window=12).std()
#perform rolling statistics
orig=plt.plot(timeseries,color='blue',label='original')
```

```
mean=plt.plot(movingaverage,color='red',label='rolling mean')
std=plt.plot(movingstd,color='black',label='rolling std')
plt.legend(loc='best')
plt.title('rolling mean & standard deviation')
plt.show(block=False)

#perform dicky fuller test
print('results of dickey fuller test: ')
dftest=adfuller(timeseries['Traffic'],autolag='AIC')
dfoutput=pd.Series(dftest[0:4],index=['Test statistics','p-value','lags
used','no. of observations used'])
for key,value in dftest[4].items():
dfoutput['Critical values (%s)'%key]=value
print(dfoutput)
datasetlogshifting.dropna(inplace=True)
test_stationary(datasetlogshifting)
```

Apply Arima Model:

```
from statsmodels.tsa.arima_model import ARIMA
#AR model
datasetlogshifting.dropna(inplace=True)
indexdataset_logscale.dropna(inplace=True)
model=ARIMA(indexdataset_logscale,order=(2,1,1))
results_AR=model.fit(disp=-1)
plt.plot(datasetlogshifting)
actual=datasetlogshifting.max()*indexdataset.max()
predicted=results_AR.fittedvalues.max()*indexdataset.max()
# print(type(actual))
# print(predicted)
traffic=actual-predicted
print(traffic)
plt.plot(results_AR.fittedvalues,color='red')
plt.title('rss: %4f '% sum((results_AR.fittedvaluesdatasetlogshifting['
Traffic'])**2))
```

```
print('Plotting AR model')
```

Output:

```
Traffic 865.557324
dtype: float64
```

7.1 **Network_Intrusion_Detection.ipynb**

Load Dataset:

```
train = pd.read_csv("B:/GLA SEM 6/MINI PROJECT II/network-intrusion-
detection/Train_data.csv")
test = pd.read_csv("B:/GLA SEM 6/MINI PROJECT II/network-intrusion-
detection/Test_data.csv")
```

Feature Selection:

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

# fit random forest classifier on the training set
rfc.fit(train_x, train_y);
# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':train_x.columns,'importance':score})
importances =
importances.sort_values('importance',ascending=False).set_index('feature')
# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();

from sklearn.feature_selection import RFE
import itertools
rfc = RandomForestClassifier()

# create the RFE model and select 10 attributes
rfe = RFE(rfc, n_features_to_select=15)
rfe = rfe.fit(train_x, train_y)
```

```
# summarize the selection of the attributes
feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(),
train_x.columns)]
selected_features = [v for i, v in feature_map if i==True]
selected_features
```

Fitting Models:

```
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train, Y_train);

# Train LogisticRegression Model
LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
LGR_Classifier.fit(X_train, Y_train);

# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy',
random_state=0)
DTC_Classifier.fit(X_train, Y_train)
```

Evaluate Models:

```
from sklearn import metrics
```

```
models = []
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))

for i, v in models:
    scores = cross_val_score(v, X_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train,
v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} Model Evaluation
==============================='.format(i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

Validate Models:

```
for i, v in models:
    accuracy = metrics.accuracy_score(Y_test, v.predict(X_test))
    confusion_matrix = metrics.confusion_matrix(Y_test, v.predict(X_test))
    classification = metrics.classification_report(Y_test, v.predict(X_test))
    print()
    print('============================== {} Model Test Results
=============================='.format(i))
```

```
print()
print ("Model Accuracy:" "\n", accuracy)
print()
print("Confusion matrix:" "\n", confusion_matrix)
print()
print("Classification report:" "\n", classification)
print()
```

Chapter 8

References

To complete this project successfully there are so many online resources are helpful which provide enough information to complete this project so some of the most helpful resources are listed below which provide such a great content to complete this project.

https://www.youtube.com/watch?v=PTxGEA1dFAw

https://www.kaggle.com/what0919/intrusion-detection-classification-by-jinner

https://en.wikipedia.org/wiki/Intrusion_detection_system

https://www.datacamp.com/community/tutorials/random-forests-classifier-python

https://stackoverflow.com/questions/tagged/intrusion-detection

Chapter 9

Conclusion

In this Digital Era everyone is connected through network so it is possible that the huge amount of data in the network channel can create the congestion in the network. So, it is important to analyse the network congestion factors that we can use to reduce the congestion in the network which will also include the security aspects of data packets loss problems. So through using this type of model is easy to identify the traffic in particular month as well as the cost required for the particular bandwidth so that the congestion in network will be less. Network Anomaly is one of the causes of network performance so it’s is import to detect the anomaly in the network to improve the security.