National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Harshitha Deenadayal
Student ID: x18136231

School of Computing
National College of Ireland

Supervisor: Dr. Pierpaolo Dondio

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Harshitha Deenadayal
Student ID:	x18136231
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Pierpaolo Dondio
Submission Due Date:	13/12/2019
Project Title:	Configuration Manual
Word Count:	1176
Page Count:	13

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.

Signature:	
Date:	12th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
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.	☐

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

Harshitha Deenadayal
x18136231

1 Introduction

This manual will explain system set-up including hardware and software requirements and the codes in python and pyspark programming language for the implementation of the research project: 'Distributed Intrusion Detection System for Computer Networks using Hyper-tuned Gradient Boosting algorithm on Spark Framework.'

2 System configuration

2.1 Hardware

Processor: Intel(R) Core(TM) i7-5500U CPU@2.4GHz GPU: NVIDIA GeForce RAM:16GB Storage: 1 TB HDD; Operating system: Windows 10, 64-bit.

2.2 Software for centralized environment

- Centralized environment on linux virtual machine. With Ubuntu 64-bit OS, 2 CPU processors and 11GB RAM.
- Anaconda Navigator (version 3) to launch jupyter notebook on internal explorer.
- Python(v3) programming language for data pre-processing, data mining modelling and visualization, on jupter notebook.

2.3 Software for distributed environment

- Distributed environment set-up using Apache Hadoop and Apache Spark on a linux virtual machine. With Ubuntu 64-bit OS, 2 CPU processors and 11GB RAM.
- Java jdk version 8 and SSH (secure shell) keys are installed on Ubuntu OS.
- Apache Hadoop version 2.9.2 is downloaded from apache website[1], installed and configured. And the path is set between hadoop and java.

[1]https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.9.2/hadoop-2.9.2.tar.gz

Hadoop Installation on Ubuntu

1) Java installation:

```
$sudo apt install openjdk-8-jdk
$sudo apt install rsync
```

2)set the path between Hadoop and java on Ubuntu and set this java file as source

```
$sudo nano /etc/profile
export JAVA_HOME=/usr
$source /etc/profile
```

3) configure SSH keys (secure shell)

-Add a new user account hduser to the hadoop group and set password

```
$sudo addgroup hadoopgroup
$sudo adduser -ingroup hadoopgroup hduser
```

-install ssh

```
$sudo apt-get install ssh
```

-enable ssh

```
$sudo systemctl enable ssh
```

-start ssh

```
$sudo systemctl start ssh
```

4) Genearate keys

```
$su – hduser
$ssh-keygen -t rsa -P "
```

5)download hadoop from ubuntu at hadoop.apache.org

```
$ wget http://ftp.heanet.ie/mirrors/www.apache.org/dist/hadoop/common/hadoop-2.9.2/hadoop-2.9.2.tar.gz
$cd /home/hduser
$sudo mv ./hadoop-2.9.2 /usr/local
$sudo chown -R hduser:hadoopgroup /usr/local/hadoop-2.9.2/
```

6) Hadoop configuration

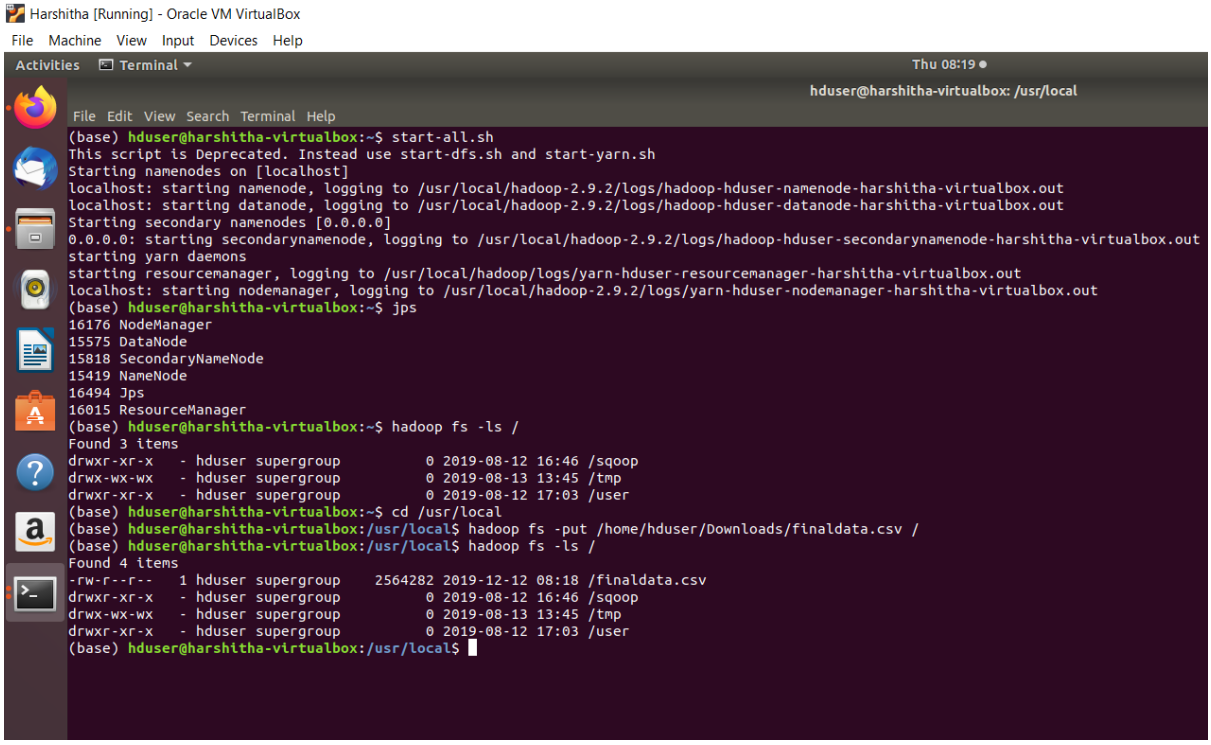
- login to hduser

```
$nano ./.bashrc
```

```
# Hadoop config
export HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
# Native path
export HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_PREFIX/lib/native"
# Java path
export JAVA_HOME="/usr"
# OS path
export PATH=$PATH:$HADOOP_HOME/bin:$JAVA_PATH/bin:$HADOOP_HOME/sbin
```

7) change the directory to,

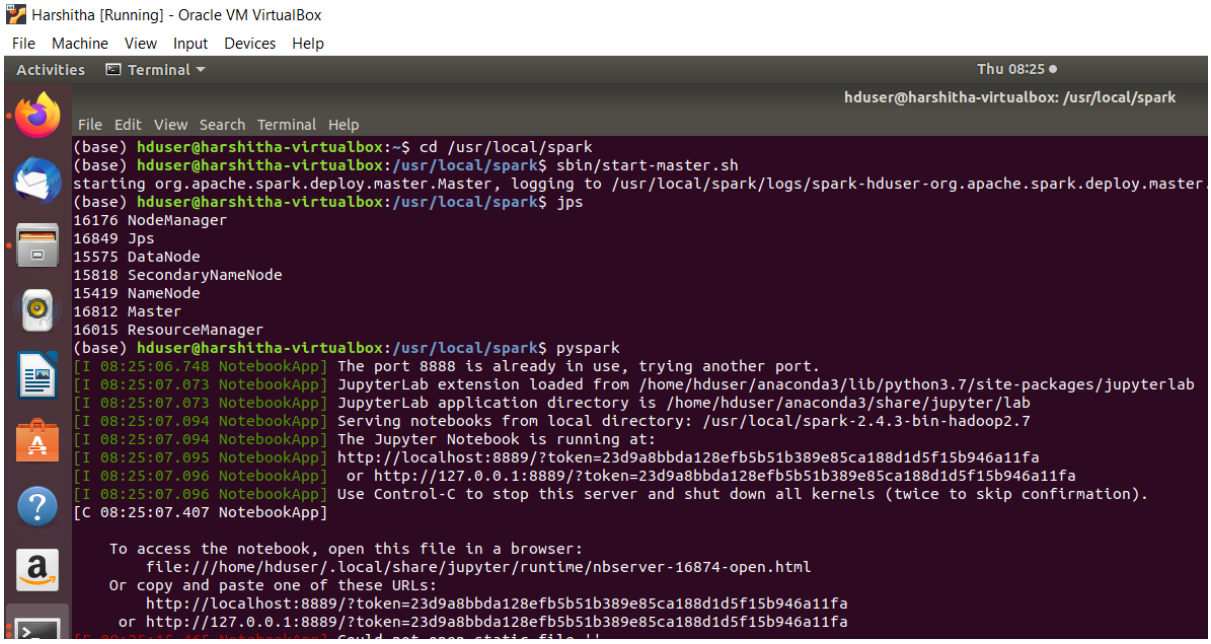
```
        $cd /usr/local/hadoop
        $cd /etc/hadoop
        $sudo nano core-site.xm

<configuration>
<property>
 <name>fs.default.name</name>
   <value>hdfs://localhost:9000</value>
</property>
</configuration>

        $sudo nano hdfs-site.xml

<configuration>
<property>
 <name>dfs.replication</name>
 <value>1</value> </property>
<property>
<name>dfs.name.dir</name>
<value>file:/usr/local/hadoop/hadoopdata/hdfs/namenode</value>
</property>

<property>
<name>dfs.data.dir</name>
<value>file:/usr/local/hadoop/hadoopdata/hdfs/datanode</value>
</property>
</configuration>




        $sudo nano mapred-site.xml
<configuration>
 <property>
  <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
</configuration>

        $$sudo nano yarn-site.xml
<configuration>
 <property>
  <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
 </property>
</configuration>

8)move the main hadoop to,
        $cd ../..
        $cd bin
        $sudo hdfs namenode -format
```



- Apache spark 2.4.3 is downloaded and installed[2].

```
                                        Apache Spark
1) Install Spark under /usr/local
        $ sudo cp ~/Downloads/spark-2.4.3-bin-hadoop2.7.tgz .
        $ sudo tar xzf spark-2.4.3-bin-hadoop2.7.tgz

2) Create a symbolic link named spark
        $ sudo ln -s spark-2.4.3-bin-hadoop2.7 spark

3) Change the ownership of the files
        $ sudo chown -R hduser:hadoopgroup spark

4) Download from website and install Anaconda with Python 2.7 for Linux
        $ sudo bash Anaconda2-2019.03-Linux-x86_64.sh

5) Spark environment variables to the .bashrc file from /home/hduser

export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_PYTHON=/usr/local/anaconda2/bin/python2
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"

6) Source bashrc
        $ source ./.bashrc
```

[2]https://spark.apache.org/releases/spark-release-2-4-3.html



- Anaconda with python 2.7 on spark environment, to launch jupyter notebook on internal explorer.

3 Project Development

The development of this project happens in several stages namely: data pre-processing, feature engineering, modelling, evaluation and cross validation. The experiments are conducted on both centralized and distributed environments to compare the time-taken to train the models. In this research project, data pre-processing stage includes exploratory data analysis and data transformations. Two steps of feature engineering is carried out and they are: Feature reduction and feature selection. Feature reduction is based on association rule mining technique where feature selection is based on variable importance graph plotted using random forest. The dataset is split into train and test samples. Data mining models are build using various algorithms like SVM, decision tree, random forest, gradient boosting and an ensemble of all these four models. Parameters of all algorithms are hyper tuned to estimate the best fit to achieve best accuracy. All the models are trained with best-fit estimators using train dataset and validated on unknown test instances. A confusion matrix is plotted for measuring the performance of each model. Using the pre-defined function of python accuracy, precision, recall and F1 score are calculated. Finally, all the models are cross validated to ensure there are no over-fitting issues.

The above explained project development process is carried out similarly on distributed (apache hadoop and apache spark) environment as well and time-taken(in-minutes) to train the models are compared with centralized environment without hadoop and spark set-up.

4 Code in Python

4.1 Libraries

In [ ]:
```
import numpy as np                                  #to performs operations on array
import pandas as pd                                 #for data manipulations
import seaborn as sns                               #visualiation library for statistical graph
from pprint import pprint                           #provides data structures in a input interpreter form
import statistics                                   #for calculation

from sklearn.preprocessing import LabelEncoder                          #for converting cat variables into numeric
from sklearn.multioutput import MultiOutputClassifier                       #for hybrid model
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold,StratifiedKFold      #for cross validation
from sklearn.model_selection import GridSearchCV                                 #for hyper-parameter tuning

from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier

from sklearn.metrics import confusion_matrix                                      #for model evaluation
from sklearn.metrics import precision_score,recall_score,f1_score
from sklearn.metrics import accuracy_score
```

In [ ]:
```
%%time
#Visualization
import matplotlib.pyplot as plt

#Config for plotting graph
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 50
fig_size[1] = 20
plt.rcParams["figure.figsize"] = fig_size
plt.rcParams.update({'font.size': 50})
```

In the above lines of codes, all the required libraries are imported and parameters for visualization graph are set[3].

4.2 Data pre-processing

In [ ]:
```
#import the dataset
dataset = pd.read_csv('/home/hduser/Downloads/finaldata.csv')
```

In [ ]:
```
dataset.head()                          #display top 5 rows
```

In [ ]:
```
dataset.shape                           #display dimension of dataframe
```

In [ ]:
```
dataset['class'].value_counts()         #checking class imbalance
```

In [ ]:
```
dataset.dtypes                          #verify the data types
```

In [ ]:
```
#Transformation: convert all categorical attributes to numeric.
encoding = LabelEncoder()                                           #init the encoder

out1=encoding.fit_transform(dataset['class'])
dataset['num_class'] = np.array(out1)                               #save encoded outcome to new col

out2=encoding.fit_transform(dataset['protocol_type'])
dataset['num_proto'] = np.array(out2)

out3=encoding.fit_transform(dataset['service'])
dataset['num_service'] = np.array(out3)

out4=encoding.fit_transform(dataset['flag'])
dataset['num_flag'] = np.array(out4)
```

In [ ]:
```
dataset= dataset.drop(['service','protocol_type', 'flag'], axis=1)   #remove categorical varaibles
```

In the above lines of code, dataset is imported. Some Data exploration is performed, followed by transformation of all categorical variables to numeric.

[3]https://docs.python.org/3/library/

4.3 Feature Reduction

```
In [ ]: #Feature reduction: remove high correlated independent attributes

        # Create correlation matrix
        corr_matrix = dataset.corr().abs()

        # Select upper triangle of correlation matrix
        upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

        # Find features with correlation greater than 0.7
        to_drop = [column for column in upper.columns if any(upper[column] > 0.7)]

        # Drop features
        dataset.drop(to_drop, axis=1, inplace=True)

In [ ]: to_drop                        #following attributes are dropped

In [ ]: dataset.shape                   #verify the dimension of the dataset

In [ ]: dataset.corr()                  #checking the correlation between independent variables again

In [ ]: sns.heatmap(dataset.corr())     #verify correlation again
```

In the above lines of codes, dataset is checked for high correlated independent variables and a heat map is plotted for visualization. Highly correlated variables are then dropped.

4.4 Feature Selection

```
In [ ]: #Feature selection: check for highly significant attributes for predicting.

        x = dataset.drop(['num_class', 'class'], axis=1)
        y = dataset[['num_class']]

        clf = RandomForestClassifier(random_state=0, n_jobs=-1)   # Create decision tree classifer object
        model = clf.fit(x, y)                                     # Train model
        importances = model.feature_importances_                  # Calculate feature importances
        indices = np.argsort(importances)[::-1]                   # Sort feature importances in descending order

        # Rearrange feature names so they match the sorted feature importances
        names = [dataset.columns[i] for i in indices]

        # Create plot
        plt.figure()
        plt.title("Feature Importance")                    # Create plot title
        plt.bar(range(x.shape[1]), importances[indices])

        # Add feature names as x-axis labels
        plt.xticks(range(x.shape[1]), names,rotation=90)
        plt.show()

In [ ]: z = dataset.iloc[0:22544,[ 2,  1, 28, 20, 17,  8,  0, 27, 25,  6, 19]]  #select top 10

In [ ]: dataset_num = z                                                          #copy the filtered dataset

In [ ]: dataset_num.shape                                                        #verify the dimension
        dataset_num.head()
```

In the above lines of codes, a variable importance graph is plotted using random forest algorithm. Based on the significance of variables in predicting class of unknown instances, only top 10 significant variables are selected for training data mining models.

4.5 Support Vector Machine

In [ ]:
```
#SVC with hyper parameter tuning

svc_param = {'C':[1,10,100,1000],'gamma':[1,0.1,0.01,0.001], 'kernel':['rbf']} #select a range of hyper-parameters
svc_grid = GridSearchCV(SVC(),svc_param,n_jobs=-1,refit = True, verbose=2, cv = 10) #run all iterations
svc_grid.fit(predictors_train,result_train)                                          #fit the model
svc_best = svc_grid.best_estimator_                                                  #select the best fit
svc_best.fit(predictors_train,result_train)                                         #train the model with best fit parameters
svc_pred = svc_best.predict(predictors_test)                                         #validated using test data
```

In [ ]:
```
svc_confusion_matrix =confusion_matrix(result_test,svc_pred)         #draw a confusion to find TP,TN,FP and FN
print (svc_confusion_matrix)
sns.heatmap(svc_confusion_matrix)

plt.rcParams["figure.figsize"] = fig_size
plt.rcParams.update({'font.size': 50})
plt.title("SVC Confusion Matrix")
print ("Precision score "+str(precision_score(result_test,svc_pred)))  #find precision, recall, f1 score and Accuracy
print ("Recall score "+str(recall_score(result_test,svc_pred)))
print ("F1 score "+str(f1_score(result_test,svc_pred)))
print ("Accuracy "+str(accuracy_score(result_test,svc_pred)))

svck10 = cross_val_score(svc_best, predictors_test, result_test, cv=10)  # 10-fold cross-validation
svc_avg10 = statistics.mean(svck10)
print(svck10)
print(svc_avg10)
```

In the above lines of code, hyper parameters of SVC algorithm are tuned. 'rbf' kernel was selected since dataset was non-linear. SVC model is trained with train dataset with number of iterations equal to all combinations of hyper parameters and best fit parameters is selected using 'estimator' function. It is then, validated using test dataset. Performance of this model is calculated using confusion matrix. Metric such as precision, recall, f1 score and accuracy are used for performance evaluation.

4.6 Decision Tree

In [ ]:
```
#decision tree classifier

dt = DecisionTreeClassifier()                                                    #base classifier
sample_split_range = list(range(2, 50))                                          #set the parameter
dt_param = dict(min_samples_split=sample_split_range)
dt_grid = GridSearchCV(dt, dt_param, cv=10,n_jobs=-1,verbose=2, scoring='accuracy')  #run all the iterations
dt_grid.fit(predictors_train, result_train)                                      #fit the model
dt_best = dt_grid.best_estimator_                                          #select the best set of hyper-parameters
dt_best.fit(predictors_train,result_train)                                 #train the model
dt_pred = dt_best.predict(predictors_test)                                 #test the model
```

In [ ]:
```
confusion_matrix_dt =confusion_matrix(result_test,dt_pred)
print (confusion_matrix_dt)
sns.heatmap(confusion_matrix_dt)
plt.title("Decision Tree Confusion Matrix")
print ("Precision "+str(precision_score(result_test,dt_pred)))
print ("Recall "+str(recall_score(result_test,dt_pred)))
print ("F1 score "+str(f1_score(result_test,dt_pred)))
print ("Accuracy "+str(accuracy_score(result_test,dt_pred)))

dtk10 = cross_val_score(dt_best, predictors_test, result_test, cv=10)  # 10-fold cross-validation
dt_avg10 = statistics.mean(dtk10)
print(dtk10)
print(dt_avg10)
```

In the above lines of code, hyper parameters of decision tree algorithm are tuned. Decision tree model is trained with train dataset with number of iterations equal to all combinations of hyper parameters and best fit parameters is selected using 'estimator' function. It is then, validated using test dataset. Performance of this model is calculated using confusion matrix. Metric such as precision, recall, f1 score and accuracy are used for performance evaluation. A 10-fold cross-validation of the accuracy of the model is performed to ensure there is no over-fitting.

4.7 Random Forest

In [ ]:
```
#RandormForest classifier with and without hyper parameter tuning

rf = RandomForestClassifier()              #create base rf classifier
pprint(rf.get_params())                    #check the default parameters
rf_param = {                               #tune the parameters
    'bootstrap': [True],                   #sampling are drawn with replacement
    'max_depth': [80, 90, 100, 110],       #number of internal node
    'max_features': [2, 3],                 #number of attributes considered for splitting
    'min_samples_leaf': [3, 4, 5],          #number of splits at leaf nodes
    'min_samples_split': [8, 10, 12],       #number of splits at each internal node
    'n_estimators': [100, 200, 300, 1000]
}
rf_grid = GridSearchCV(estimator = rf, param_grid = rf_param,
                       cv = 10, n_jobs = -1, verbose = 2)    #iterations all combination of hyper-parameter
rf_grid.fit(predictors_train, result_train)                 #fit the model
rf_best = rf_grid.best_estimator_                           # find the best estimator
rf_final = rf_best.fit(predictors_train,result_train)       ##train the model with best fit estimators
rf_pred = rf_final.predict(predictors_test)                #validate the model using test data
```

In [ ]:
```
rf_confusion_matrix =confusion_matrix(result_test,rf_pred)
print (rf_confusion_matrix)
sns.heatmap(rf_confusion_matrix)
plt.title("Random Forest Confusion Matrix")
print ("Precision score "+str(precision_score(result_test,rf_pred)))
print ("Recall score "+str(recall_score(result_test,rf_pred)))
print ("F1 score "+str(f1_score(result_test,rf_pred)))
print ("Accuracy "+str(accuracy_score(result_test,rf_pred)))

rfk10 = cross_val_score(rf_best, predictors_test, result_test, cv=10)  # 10-fold cross-validation
rf_avg10 = statistics.mean(rfk10)
print(rfk10)
print(rf_avg10)
```

In the above lines of code, hyper parameters of random forest algorithm are tuned. Random forest model is trained with train dataset with number of iterations equal to all combinations of hyper parameters and best fit parameters is selected using 'estimator' function. It is then, validated using test dataset. Performance of this model is calculated using confusion matrix. Metric such as precision, recall, f1 score and accuracy are used for performance evaluation. A 10-fold cross-validation of the accuracy of the model is performed to ensure there is no over-fitting.

4.8 Gradient Boosting

In [ ]:
```
#firstly tune the parameter for gradient boosting

min_samples_leaf= [20, 50,100,150]                          #number of splits at leaf nodes
learning_rate= [0.5, 0.4, 0.3, 0.2, 0.1, 0.05, 0.02, 0.01]  #contribution of each tree towards learning
max_depth= [4, 6, 8]                                        #number of internal node

tune1=[]                                                    #plotting graph of accuracy vs learning rate
for item in learning_rate:
    plot1 = GradientBoostingClassifier(learning_rate=item)
    plot1.fit(predictors_train,result_train)
    tune1.append(plot1.score(predictors_test,result_test))

plt.plot(learning_rate,tune1)
plt.title("Learning rate")
```

In [ ]:
```
tune2=[]                                                    # plotting graph of accuracy vs max-depth
for item in max_depth:
    graph2 = GradientBoostingClassifier(max_depth=item)
    graph2.fit(predictors_train,result_train)
    tune2.append(graph2.score(predictors_test,result_test))
    print(item)
plt.plot(max_depth,tune2)
plt.title("Max depth")
```

In [ ]:
```
tune3=[]                                                    # plotting graph of accuracy vs min_samples_leaf
for item in min_samples_leaf:
    graph3 = GradientBoostingClassifier(min_samples_leaf=item)
    graph3.fit(predictors_train,result_train)
    tune3.append(graph3.score(predictors_test,result_test))
    print(item)
plt.plot(min_samples_leaf,tune3)
plt.title("Min samples leaf")
```

Above lines of codes, plots Accuracy vs. learning-rate, Accuracy vs. max-depth and Accuracy vs. min-samples-leaf graphs to determine the best values of hyper-parameters for high accuracy.

In [ ]:
```
#Gradient Boosting

gb_param = {'learning_rate': [0.1],
              'max_depth': [6],
              'min_samples_leaf': [20],
              #'max_features': [1.0, 0.3, 0.1]     #number of features considered for split.
              }

gb_grid = GridSearchCV(GradientBoostingClassifier(), gb_param,verbose=2, cv=10, n_jobs=-1)  #set the parameter range
gb_grid.fit(predictors_train, result_train)                                                  #fit the model
gb_best = gb_grid.best_estimator_                                                            #select the best fit
gb_best.fit(predictors_train,result_train)                                                 #train the model with best fit
gb_pred = gb_best.predict(predictors_test)                                                      #validate using test data
```

In [ ]:
```
gb_confusion_matrix =confusion_matrix(result_test,gb_pred)
print (gb_confusion_matrix)
sns.heatmap(rf_confusion_matrix)
plt.title("Gradient Boosting Confusion Matrix")
print ("Precision score "+str(precision_score(result_test,gb_pred)))
print ("Recall score "+str(recall_score(result_test,gb_pred)))
print ("F1 score "+str(f1_score(result_test,gb_pred)))
print ("Accuracy "+str(accuracy_score(result_test,gb_pred)))

gbk10 = cross_val_score(gb_best, predictors_test, result_test, cv=10)  # 10-fold cross-validation
gb_avg10 = statistics.mean(gbk10)
print(gbk10)
print(gb_avg10)
```

In the above lines of code, best fit values which are obtained based on graphs plotted are set for hyper-parameters. Random forest model is trained with train dataset with number of iterations equal to all combinations of hyper parameters and best fit parameters is selected using 'estimator' function. It is then, validated using test dataset. Performance of this model is calculated using confusion matrix. Metric such as precision, recall, f1 score and accuracy are used for performance evaluation. A 10-fold cross-validation of the accuracy of the model is performed to ensure there is no over-fitting.

4.9 Hybrid

In [ ]:
```
from sklearn.ensemble import VotingClassifier
hybrid_classifier = MultiOutputClassifier(
                        VotingClassifier(estimators=
                                         [('svc',svc_best),('dt',dt_best),('rf',rf_best),('xgb',gb_best)],voting='hard'))
hybrid_classifier.fit(predictors_train,result_train)   #fit the hybrid model
pred_hybrid=hybrid_classifier.predict(predictors_test)  #validate the hybrid
```

In [ ]:
```
model1=[]
for item in pred_hybrid:
    model1.append(item[0])         #add hybrid model in the table
```

In [ ]:
```
confusion_matrix_hybrid =confusion_matrix(np.array(result_test['num_class']),np.array(model1))
print (confusion_matrix_hybrid)
print ("Precision "+str(precision_score(np.array(result_test['num_class']),np.array(model1))))
print ("Recall "+str(recall_score(np.array(result_test['num_class']),np.array(model1))))
print ("F1 score "+str(f1_score(np.array(result_test['num_class']),np.array(model1))))
print ("Accuracy "+str(accuracy_score(np.array(result_test['num_class']),np.array(model1))))

sns.heatmap(confusion_matrix_hybrid)
plt.title("Hybrid Model Confusion Matrix")
```

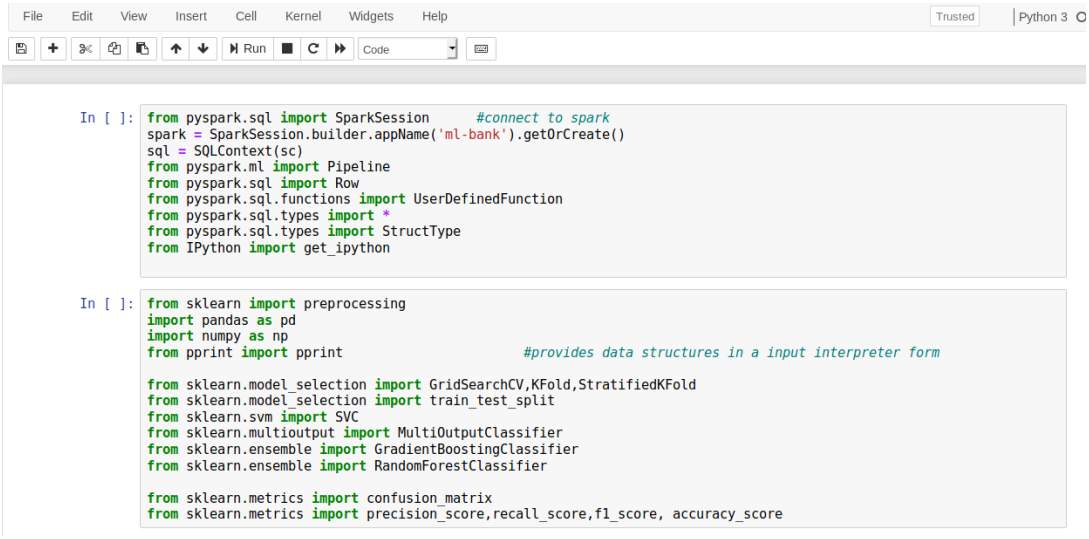
In [ ]:
```
## mse
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(result_test,pred_hybrid)
mse
```

In [ ]:
```
# cross validated the hybrid model to test for over-fitting with 3 and 10 folds
k10 = cross_val_score(hybrid_classifier, predictors_test, result_test, cv=10)
avg10 = statistics.mean(k10)
print(avg10)
```

Using the above lines of codes, an ensemble model of all four models is trained to enhance the overall performance of data mining model. This model is then cross-validated with 10 folds.

5 Code in Pyspark

Code written in pyspark for distributed environment (Apache Hadoop and Apache Spark) is similar to code written python, except for minor changes.



In the above lines of codes, a connection to spark is established first. Then, all the required libraries are loaded.

In the above lines of code, dataset is imported from HDFS. Columns names transformed by converted into lowercase, removing special characters and white spaces. All

the categorical variables in dataset are converted into numeric.
And the rest of the codes are similar to code written in python and are based on feature engineering performed earlier. The main purpose of experiment is to measure the time taken for training the models.

In [ ]:
```
#feature reduction by dropping highly correlated col
dataset = dataset.drop(['su_attempted','num_root','num_file_creations','srv_serror_rate',
                        'srv_rerror_rate','dst_host_srv_count','dst_host_same_srv_rate',
                        'dst_host_serror_rate','dst_host_srv_serror_rate','dst_host_rerror_rate',
                        'dst_host_srv_rerror_rate'], axis=1)
```

In [ ]:
```
#Feature selection: selecting top 10 significant varaibles
dataset = dataset[['dst_bytes', 'src_bytes', 'same_srv_rate', 'srv_count',
       'logged_in', 'duration', 'class', 'dst_host_same_src_port_rate', 'hot',
       'rerror_rate']]
```

In [ ]:
```
dataset.shape
```

In [ ]:
```
##creating predictors and outcomes
predictors = dataset[['dst_bytes', 'src_bytes', 'same_srv_rate', 'srv_count',
       'logged_in', 'duration',  'dst_host_same_src_port_rate', 'hot',
       'rerror_rate']]

result = dataset[['class']]
```

In [ ]:
```
#splitting into train and test
predictors_train,predictors_test,result_train,result_test = train_test_split(predictors,result,test_size=.2)
```

In [ ]:
```
#SVC with hyper parameter tuning

svc_param = {'C':[1,10,100,1000],'gamma':[1,0.1,0.01,0.001], 'kernel':['rbf']} #select a range of hyper-parameters
svc_grid = GridSearchCV(SVC(),svc_param,n_jobs=-1,refit = True, verbose=2, cv=10)  #run all iterations
svc_grid.fit(predictors_train,result_train)                                        #fit the model
svc_best = svc_grid.best_estimator_                                                #select the best fit
svc_best.fit(predictors_train,result_train)                                      #train model with best fit parameters
svc_pred = svc_best.predict(predictors_test)                                       #validated using test data

svc_confusion_matrix =confusion_matrix(result_test,svc_pred)         #draw a confusion to find TP,TN,FP and FN
print (svc_confusion_matrix)
print ("Precision score "+str(precision_score(result_test,svc_pred)))  #find precision, recall, f1 score, Accuracy
print ("Recall score "+str(recall_score(result_test,svc_pred)))
print ("F1 score "+str(f1_score(result_test,svc_pred)))
print ("Accuracy "+str(accuracy_score(result_test,svc_pred)))
```

In [ ]:
```
svc_confusion_matrix =confusion_matrix(result_test,svc_pred)         #draw a confusion to find TP,TN,FP and FN
print (svc_confusion_matrix)
print ("Precision score "+str(precision_score(result_test,svc_pred)))  #find precision, recall, f1 score, Accuracy
print ("Recall score "+str(recall_score(result_test,svc_pred)))
print ("F1 score "+str(f1_score(result_test,svc_pred)))
print ("Accuracy "+str(accuracy_score(result_test,svc_pred)))
```

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
#decision tree classifier

dt = DecisionTreeClassifier()                                                  #base classifier
sample_split_range = list(range(2, 50))                                        #set the parameter
dt_param = dict(min_samples_split=sample_split_range)
dt_grid = GridSearchCV(dt, dt_param, cv=10,n_jobs=-1,verbose=2, scoring='accuracy')  #run all the iterations
dt_grid.fit(predictors_train, result_train)                                    #fit the model
dt_best = dt_grid.best_estimator_                                  #select the best set of hyper-parameters for learning
dt_best.fit(predictors_train,result_train)                                     #train the model
dt_pred = dt_best.predict(predictors_test)                                     #test the model

confusion_matrix_dt =confusion_matrix(result_test,dt_pred)
print (confusion_matrix_dt)
print ("Precision "+str(precision_score(result_test,dt_pred)))
print ("Recall "+str(recall_score(result_test,dt_pred)))
print ("F1 score "+str(f1_score(result_test,dt_pred)))
print ("Accuracy "+str(accuracy_score(result_test,dt_pred)))
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from pprint import pprint                    #provides data structures in a input interpreter form
#RandormForest classifier with and without hyper parameter tuning


rf = RandomForestClassifier()              #create base rf classifier
pprint(rf.get_params())                    #check the default parameters
rf_param = {                               #tune the parameters
    'bootstrap': [True],                   #sampling are drawn with replacement
    'max_depth': [80, 90, 100, 110],       #number of internal node
    'max_features': [2, 3],                 #number of attributes considered for splitting
    'min_samples_leaf': [3, 4, 5],          #number of splits at leaf nodes
    'min_samples_split': [8, 10, 12],       #number of splits at each internal node
    'n_estimators': [100, 200, 300, 1000]
}

rf_grid = GridSearchCV(estimator = rf, param_grid = rf_param,
                       cv = 10, n_jobs = -1, verbose = 2)    #iterations all combination of hyper-parameter
rf_grid.fit(predictors_train, result_train)                  #fit the model
rf_best = rf_grid.best_estimator_                            # find the best estimator
rf_final = rf_best.fit(predictors_train,result_train)        ##train the model with best fit estimators
rf_pred = rf_final.predict(predictors_test)                 #validate the model using test data


print ("Precision score "+str(precision_score(result_test,rf_pred)))
print ("Recall score "+str(recall_score(result_test,rf_pred)))
print ("F1 score "+str(f1_score(result_test,rf_pred)))
print ("Accuracy "+str(accuracy_score(result_test,rf_pred)))
```

In [ ]:
```
#Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier

gb_param = {'learning_rate': [0.1],
             'max_depth': [6],
             'min_samples_leaf': [20],
             #'max_features': [1.0, 0.3, 0.1]     #number of features considered for split.
             }

gb_grid = GridSearchCV(GradientBoostingClassifier(), gb_param,verbose=2, cv=10, n_jobs=-1)   #set the parameter range
gb_grid.fit(predictors_train, result_train)                                                  #fit the model
gb_best = gb_grid.best_estimator_                                                            #select the best fit
gb_best.fit(predictors_train,result_train)                                          #train the model with best fit
gb_pred = gb_best.predict(predictors_test)                                          #validate using test data

confusion_matrix_gb =confusion_matrix(result_test,gb_pred)
print (confusion_matrix_gb)
print ("Precision "+str(precision_score(result_test,gb_pred)))
print ("Recall "+str(recall_score(result_test,gb_pred)))
print ("F1 score "+str(f1_score(result_test,gb_pred)))
print ("Accuracy "+str(accuracy_score(result_test,gb_pred)))
```

In [ ]:
```
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import VotingClassifier
hybrid_classifier = MultiOutputClassifier(VotingClassifier(estimators=
                                        [('svc',svc_best),('dt',dt_best),('rf',rf_best),
                                         ('xgb',gb_best)],voting='hard'))
```

In [ ]:
```
hybrid_classifier.fit(predictors_train,result_train)   #fit the hybrid model
pred_hybrid=hybrid_classifier.predict(predictors_test)  #validate the hybrid
```

In [ ]:
```
model1=[]
for item in pred_hybrid:
    model1.append(item[0])    #add hybrid model in the table
```

In [ ]:
```

confusion_matrix_hybrid =confusion_matrix(np.array(result_test['class']),np.array(model1))
print (confusion_matrix_hybrid)
print ("Precision "+str(precision_score(np.array(result_test['class']),np.array(model1))))
print ("Recall "+str(recall_score(np.array(result_test['class']),np.array(model1))))
print ("F1 score "+str(f1_score(np.array(result_test['class']),np.array(model1))))
print ("Accuracy "+str(accuracy_score(np.array(result_test['class']),np.array(model1))))
```