National College of Ireland

Configuration Manual for Sepsis Prediction using Machine Learning and Deep Learning Algorithms

MSc Research Project
Data Analytics

Terrance Thomas
Student ID: X18184928

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Terrance Thomas
Student ID:	X18184928
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	28/09/2020
Project Title:	Configuration Manual for Sepsis Prediction using Machine Learning and Deep Learning Algorithms
Word Count:	8496
Page Count:	15

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:	28th September 2020

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 for Sepsis Prediction using Machine Learning and Deep Learning Algorithms

Terrance Thomas
X18184928

1 Introduction

The Configuration Manual gives a detailed explanation of the research project setup. This includes the configuration of the system, the source of the data set, the software or scripting language used for programming, the description of libraries, the packages used and the code outputs.

This document also gives results of various tests done in the research. This report also includes relevant information, graphs, and output metrics that is a part of implementation but was not added to the report due to size constraints. The reason for adding this information in the Manual is because it is worth considering for relevance and discussion of the research.

2 Experimental Environment Specification

2.1 System Configuration or Specifications

The project was executed on a local system. The system configuration is as follows:

- **Intel i7 8th generation processor**
- **1TB hard drive**
- **16 GB RAM**
- **8 GB of Nvidia GTX 1080 graphic card**
- **Operating system: Windows 10**

2.2 Technical Specifications

2.2.1 Software

- **Python:** The software version of 3.7 has been used for the research. Most of the activities like data collection, cleaning, pre-processing, exploratory analysis, modelling and evaluation is done using the software
- **Anaconda-3 Jupyter Notebook:** Python 3.7 is supported by Anaconda-3, and is a scripting platform. Jupyter Notebook is a platform used to write the code addition to its potential to combine code and share it.

- **Microsoft Excel 2003:** Data is stored in .psv format which are pipe separated value files. There are 40,336 patient records/files. Each files contains hourly data of the patients.

2.2.2 Libraries

The following libraries were used for the research:

```
import pandas as pd
from glob import glob
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import pickle
import os
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import time
from pylab import rcParams
import six
import sys
#sys.modules['sklearn.externals.six'] = six
import mlrose
from imblearn.under_sampling import NearMiss
import missingno as msno
from sklearn.metrics import classification_report, auc,
    precision_recall_curve, roc_curve
from sklearn.metrics import f1_score, precision_score,
    recall_score, accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from collections import Counter
from keras import backend as K
```

The above is the list of libraries imported for the research project. These libraries are used for multiple things like creating graphs, for data manipulation, to create performance metrics, for data analysis and other purpose tasks[1][2][3][4][5][6][7][8][9][10].

3 Data set source

The data is used from a online challenge which started in 2019. The data is part of an online challenge and is openly available on https://physionet.org/content/challenge-2019/1.0.0/ and clicking on link gives access to the data provided Name, Affiliation and email id is given. This redirects to a page to provide the data but the link seems to be a broken link.An alternate link is provided on the broken link which can be used to access the data. The link is https://archive.physionet.org/users/shared/challenge-2019/ which can be used directly and is already bypassed or after providing the information and then clicked on the link still it takes to the data of the challenge which can be downloaded.

Figure 1 shows few columns in the data set when it is loaded in python via a dataframe. Data set consist of 40,336 patients of 2 different hospitals in USA. NaN represents missing or null values of the hourly data of the patients since they have been admitted into the ICU.

	HR	O2Sat	Temp	SBP	MAP	DBP	Resp	EtCO2	BaseExcess	HCO3	...	Fibrinogen	Platelets	Age	Gender	Unit1	Unit2	HospAdmTime	ICULOS
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	83.14	0	NaN	NaN	-0.03	1
1	97.0	95.0	NaN	98.0	75.33	NaN	19.0	NaN	NaN	NaN	...	NaN	NaN	83.14	0	NaN	NaN	-0.03	2
2	89.0	99.0	NaN	122.0	86.00	NaN	22.0	NaN	NaN	NaN	...	NaN	NaN	83.14	0	NaN	NaN	-0.03	3
3	90.0	95.0	NaN	NaN	NaN	NaN	30.0	NaN	24.0	NaN	...	NaN	NaN	83.14	0	NaN	NaN	-0.03	4
4	103.0	88.5	NaN	122.0	91.33	NaN	24.5	NaN	NaN	NaN	...	NaN	NaN	83.14	0	NaN	NaN	-0.03	5
5	110.0	91.0	NaN	NaN	NaN	NaN	22.0	NaN	NaN	NaN	...	NaN	NaN	83.14	0	NaN	NaN	-0.03	6
6	108.0	92.0	36.11	123.0	77.00	NaN	29.0	NaN	NaN	NaN	...	NaN	NaN	83.14	0	NaN	NaN	-0.03	7

Figure 1: Data when loaded in python

4 Data Exploratory Analysis and Pre-Processing

This section contains Data Exploratory Analysis (EDA) and Pre-Processing. This includes codes and images which are important part of the research but were not included in the report.

[1] https://keras.io/
[2] https://pandas.pydata.org/
[3] https://scikit-learn.org/stable/
[4] https://www.tensorflow.org/
[5] https://numpy.org/
[6] https://plotly.com/
[7] https://docs.python.org/3/library/
[8] https://pypi.org/project/mlrose/
[9] https://pypi.org/project/missingno/
[10] https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under$_s ampling.NearMiss.html$

```
stock1 = sorted(glob('D:/Masters Data/RIC/Sepsis/training_setA/
    training/p0*.psv'))
#List of psv files in the folder training_setA/training
stock1

[
'D:/Masters Data/RIC/Sepsis/training_setA/training\\p000001.psv',
'D:/Masters Data/RIC/Sepsis/training_setA/training\\p000002.psv',
'D:/Masters Data/RIC/Sepsis/training_setA/training\\p000003.psv',
'D:/Masters Data/RIC/Sepsis/training_setA/training\\p000004.psv',
'D:/Masters Data/RIC/Sepsis/training_setA/training\\p000005.psv',
'D:/Masters Data/RIC/Sepsis/training_setA/training\\p000006.psv',
'D:/Masters Data/RIC/Sepsis/training_setA/training\\p000007.psv',


#adding all the psv files in the folder to a single dataframe
    using the read_csv and delimiter seperator "|"

df_A =
    pd.concat((pd.read_csv(file,sep='|').assign(filename = file)
                for file in stock1), ignore_index = True)
```

The code and output shows how the data file path are added stored in a variable and then how the files are loaded in a data frame df_A. Data set B is loaded in a similar way to df_B and then both the data set are combined into a single data frame df_F.

Figure 2 shows the correlation among all the variables in the data set.

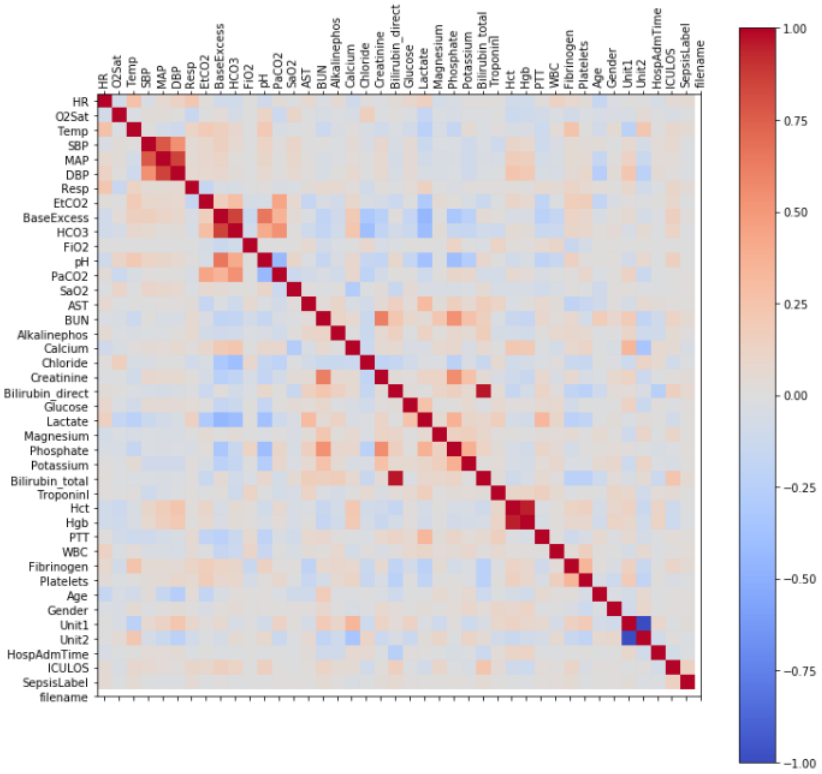

Figure 2: Correlation matrix of all columns

The below code shows all the variables which were dropped due to the high number of missing values.

```
df_F.drop(['Temp','EtCO2','BaseExcess','HCO3','FiO2',
           'pH','PaCO2','SaO2','AST','BUN','Alkalinephos',
           'Calcium','Chloride','Creatinine','Bilirubin_direct',
           'Glucose','Lactate','Magnesium','Phosphate',
           'Potassium','Bilirubin_total','TroponinI',
           'Hct','Hgb','PTT','WBC','Fibrinogen','Platelets',
           'Unit1','Unit2'], axis=1, inplace=True)
```

The code below gives the detailed code of the imputation done on the data.

```
#imputing using median

HR_median = df_F20['HR'].median()
df_F20['HR'].fillna(HR_median,inplace=True)

O2Sat_median = df_F20['O2Sat'].median()
df_F20['O2Sat'].fillna(O2Sat_median,inplace=True)


SBP_median = df_F20['SBP'].median()
df_F20['SBP'].fillna(SBP_median,inplace=True)


MAP_median = df_F20['MAP'].median()
df_F20['MAP'].fillna(MAP_median,inplace=True)


DBP_median = df_F20['DBP'].median()
df_F20['DBP'].fillna(DBP_median,inplace=True)


Resp_median = df_F20['Resp'].median()
df_F20['Resp'].fillna(Resp_median,inplace=True)

HospAdmTime_median = df_F20['HospAdmTime'].median()
df_F20['HospAdmTime'].fillna(HospAdmTime_median,inplace=True)
```

The below code is the code for the feature extraction/encoding done on the data set.

```
#Categorizing patient based on age to old, infant and Child/adult

def featuer_engineer_age(df_F10):
    df_F10.loc[df_F10['Age'] >=65, 'custom_age'] = 2
    df_F10.loc[df_F10['Age'] <1, 'custom_age'] = 0
    df_F10.loc[(df_F10['Age'] >=1) & (df_F10['Age'] <65),
            'custom_age'] = 1
    return df_F10
```

```
# O2Stat for a healthy adult is between 90 and 100 for healthy
human beings. Create a new categorical variable custom_o2stat
with levels normal, abnormal and missing

def feature_engineer_o2stat(df_F10):
    df_F10.loc[(df_F10['O2Sat'] >= 90) & (df_F10['O2Sat'] < 100),
            'custom_o2stat'] = 1
    df_F10.loc[(df_F10['O2Sat'] < 90) & (df_F10['O2Sat'] >= 0),
            'custom_o2stat'] = 0

    df_F10['custom_o2stat'].fillna(np.nan, inplace=True)
    return df_F3


#SBP stands for Systolic blood pressure, It is the upper number
while measuring Blood pressure. DBP stands for Diastolic blood
pressure , It is the lower number while measuring Blood pressure.
Using both these columns to categorize blood pressure as low,
normal, elevated , high and missing

def feature_engineer_blood_pressure(df_F10):
    df_F10.loc[(df_F10['SBP'] <90) & (df_F10['DBP'] <60),
    'custom_bp'] = 0

    df_F10.loc[(df_F10['SBP'].between(90,120, inclusive=True)) &
            (df_F10['DBP'].between(60,80, inclusive=True)),
            'custom_bp'] = 1


    df_F10.loc[(df_F10['SBP'].between(120,140, inclusive=True)) &
            (df_F10['DBP'].between(80,90, inclusive=True)),
            'custom_bp'] = 2


    df_F10.loc[(df_F10['SBP'] > 140 ) &
            (df_F10['DBP'] > 90 ), 'custom_bp'] = 3

    df_F10['custom_bp'].fillna(np.nan, inplace=True)
    return df_F10


#Respiration rate for healthy adults is between 12 and 20.
Categorizing respiratory rate as normal and abnormal based
on thresholds.
```

```
def feature_engineer_resp_rate(df_F10):
    df_F10.loc[(df_F10['Resp'].between(30,60)) & (df_F10['Age']
            <1),
            'custom_resp'] = 1
    df_F10.loc[((df_F10['Resp'] < 30) | (df_F10['Resp'] > 60)) &
             (df_F10['Age'] <1) ,'custom_resp'] = 0


    df_F10.loc[(df_F10['Resp'].between(24,40)) & (df_F10['Age'].
    between(1,3)),
            'custom_resp'] = 1
    df_F10.loc[((df_F10['Resp'] < 24) | (df_F10['Resp'] > 40)) &
             (df_F10['Age'].between(1,3)) ,'custom_resp'] = 0


    df_F10.loc[(df_F10['Resp'].between(22,34)) & (df_F10['Age'].
    between(3,6)),
            'custom_resp'] = 'normal'
    df_F10.loc[((df_F10['Resp'] < 22) | (df_F10['Resp'] > 34)) &
             (df_F10['Age'].between(3,6)) ,'custom_resp'] =
             'abnormal'


    df_F10.loc[(df_F10['Resp'].between(18,30)) & (df_F10['Age'].
    between(6,12)),
            'custom_resp'] = 1
    df_F10.loc[((df_F10['Resp'] < 18) | (df_F10['Resp'] > 30)) &
             (df_F10['Age'].between(6,12)) ,'custom_resp'] = 0


    df_F10.loc[(df_F10['Resp'].between(12,20)) & (df_F10['Age']
    >12),
            'custom_resp'] = 1
    df_F10.loc[((df_F10['Resp'] < 12) | (df_F10['Resp'] > 20)) &
    (df_F10['Age'] >12),
            'custom_resp'] = 0

    df_F10['custom_resp'].fillna(np.nan, inplace=True)


    return df_F10


# MAP for a healthy adult is between 70 and 100 for healthy human
beings. Create a new categorical variable custom_map with levels
low, normal and high

def feature_engineer_map(df_F10):
```

```
    df_F10.loc[(df_F10['MAP'] >= 69),
            'custom_map'] = 0
    df_F10.loc[(df_F10['MAP'] <= 70) & (df_F10['MAP'] >= 100),
            'custom_map'] = 1
    df_F10.loc[(df_F10['MAP'] < 100),
            'custom_map'] = 2

    df_F10['custom_map'].fillna(np.nan, inplace=True)
    return df_F10
```

The below code shows number of data that was missing from the feature extractions. The missing values had to be imputed again.

```
#checking null values of the new column

df_F10.isnull().sum()

Gender                  0
HospAdmTime             0
ICULOS                  0
SepsisLabel             0
custom_map              0
custom_bp         1151790
custom_hr               0
custom_o2stat      326414
custom_age              0
custom_resp             0
dtype: int64
```

The below code shows how NearMiss has been used to transform the imbalanced data set into a balanced data set which now matches the number of dependent variable.

```
# Implementing Undersampling for Handling Imbalanced
nm = NearMiss(random_state=42)
X_res,y_res=nm.fit_sample(X,Y)

#printing the change in the shape after under sampling

print('Original dataset shape {}'.format(Counter(Y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))

Original dataset shape Counter({0: 1524294, 1: 27916})
Resampled dataset shape Counter({0: 27916, 1: 27916})
```

The code for Variable importance is as follows:

```
plt.figure(figsize=(20,10))

model = RandomForestClassifier(random_state=1211)
model.fit(x_train,y_train)
# print(model.feature_importances_) #use inbuilt class
```

```
    feature_importances of tree based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_,
    index=X.columns)
feat_importances.nlargest(37).plot(kind='barh')
plt.show()
```

Figure 3 explains the variable importance of all the variables used in the data set after cleaning. It shows that HR is the most important variable while gender is the least important variable

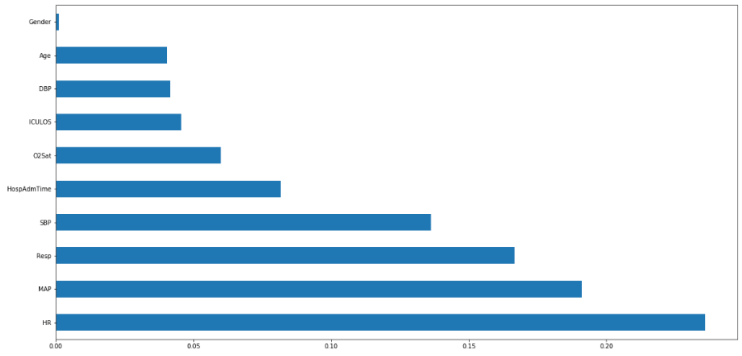

Figure 3: Variable importance

5 Models Implementation and Evaluation

This section shows all the model code and their output in terms of performance

5.1 Long Short Term Memory

The code for LSTM is as follows:

```
model = Sequential()
model.add(LSTM(256, input_shape=(1,10), return_sequences = True))
model.add(LSTM(256))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adam',
metrics=['accuracy',f1_m,precision_m, recall_m])
model.summary()


Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 1, 256)            273408
_________________________________________________________________
lstm_4 (LSTM)                (None, 256)               525312
```

```
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
=================================================================
Total params: 831,745
Trainable params: 831,745
Non-trainable params: 0



history = model.fit(X_train,y_train, epochs=45, batch_size=150,
verbose=1, validation_data=(X_test,y_test), shuffle=True)

Train on 41874 samples, validate on 13958 samples
Epoch 1/45
41874/41874 [==============================] - 7s 158us/step -
loss: 0.0529 - accuracy: 0.9348 - f1_m: 0.9284 -
precision_m: 0.9716 - recall_m: 0.8957 - val_loss: 0.0356 -
val_accuracy: 0.9559 - val_f1_m: 0.9536 -
val_precision_m: 0.9940 - val_recall_m: 0.9170
Epoch 2/45
41874/41874 [==============================] - 6s 136us/step -
loss: 0.0347 - accuracy: 0.9600 - f1_m: 0.9584 -
precision_m: 0.9855 - recall_m: 0.9340 - val_loss: 0.0240 -
val_accuracy: 0.9761 - val_f1_m: 0.9752 -
val_precision_m: 0.9911 - val_recall_m: 0.9602
```

5.2 Random Forest

The code for RF is given below

```
best = RandomForestClassifier(random_state=1211, oob_score=True,
max_depth=15, n_estimators=100, class_weight='balanced_subsample')
best.fit(x_train,y_train)

y_pred_best = best.predict(x_test)
y_pred_proba_best = best.predict_proba(x_test)[:,1]
precision_best, recall_best, thresholds_best =
precision_recall_curve(y_test, y_pred_proba_best)
auc_best = auc(recall_best, precision_best)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_best)
auroc = auc(fpr, tpr)

print(classification_report(y_test, y_pred_best))
print('F1 score:', f1_score(y_test,y_pred_best))
print('Precision:', precision_score(y_test,y_pred_best))
```

```
print('Recall:', recall_score(y_test,y_pred_best))
#print('AUPRC:', auc_best)
#print('AUC:', auroc)
print('Accuracy:', accuracy_score(y_test,y_pred_best))


 precision    recall  f1-score   support

           0       0.97      1.00      0.98      6930
           1       1.00      0.97      0.98      7028

    accuracy                           0.98     13958
   macro avg       0.98      0.98      0.98     13958
weighted avg       0.98      0.98      0.98     13958

F1 score: 0.9814319774582762
Precision: 0.9969176574196389
Recall: 0.9664200341491178
Accuracy: 0.9815876200028657
```

5.3 Decision Tree

The code for DT is given below

```
clf_dtc = DecisionTreeClassifier(criterion='entropy',
max_depth=5,random_state=0)
clf_dtc.fit(x_train, y_train.ravel())
report_performance(clf_dtc)
#roc_curves(clf_dtc)
accuracy(clf_dtc)
f1(clf_dtc)
precision(clf_dtc)
recall(clf_dtc)


Confusion Matrix:
[[6868   62]
 [ 290 6738]]


Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      6930
           1       0.99      0.96      0.97      7028

    accuracy                           0.97     13958
   macro avg       0.98      0.97      0.97     13958
```

```
weighted avg       0.98      0.97      0.97     13958

Acuuracy Of the Model:  0.9747814873191002


F1 score Of the Model:  0.974544402661267


Precision  Of the Model:  0.9908823529411764


Recall  Of the Model:  0.9587364826408651
```

5.4 Extreme Gradient Boosting

The code for XGBoost is given below

```
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [1,2,3,4,5]
        }

clf_xgb = XGBClassifier(random_state=0)
clf_xgb.fit(x_train, y_train.ravel())
report_performance(clf_xgb)
#roc_curves(clf_xgb)
accuracy(clf_xgb)
f1(clf_xgb)
precision(clf_xgb)
recall(clf_xgb)


Confusion Matrix:
[[6903   27]
 [ 213 6815]]


Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      6930
```

```
           1       1.00      0.97      0.98      7028

    accuracy                           0.98     13958
   macro avg       0.98      0.98      0.98     13958
weighted avg       0.98      0.98      0.98     13958


Acuuracy Of the Model:  0.9828055595357501



F1 score Of the Model:  0.9826964671953857



Precision  Of the Model:  0.996053785442853



Recall  Of the Model:  0.9696926579396699
```

5.5 Adaptive Boosting

The code for AdaBoost is given below

```
clfs =[AdaBoostClassifier()]

for model in clfs:
    print(str(model).split('(')[0],": ")
    model.fit(x_train,y_train.ravel())
    X = pd.DataFrame(x_train)
    report_performance(model)
    #roc_curves(model)
    accuracy(model)
    f1(model)
    precision(model)
    recall(model)


AdaBoostClassifier :


Confusion Matrix:
[[6876   54]
 [ 265 6763]]


Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      6930
           1       0.99      0.96      0.98      7028

    accuracy                           0.98     13958
   macro avg       0.98      0.98      0.98     13958
weighted avg       0.98      0.98      0.98     13958


Acuuracy Of the Model:  0.9771457228829346



F1 score Of the Model:  0.9769591910436981



Precision  Of the Model:  0.9920786269620068



Recall  Of the Model:  0.9622936824132043
```

5.6 K Nearest Neighbors

The code for KNN is given below

```
clfs =[KNeighborsClassifier()]

for model in clfs:
    print(str(model).split('(')[0],": ")
    model.fit(x_train,y_train.ravel())
    X = pd.DataFrame(x_train)
    report_performance(model)
    #roc_curves(model)
    accuracy(model)
    f1(model)
    precision(model)
    recall(model)

KNeighborsClassifier :


Confusion Matrix:
[[6888   42]
 [ 339 6689]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      6930
           1       0.99      0.95      0.97      7028

    accuracy                           0.97     13958
   macro avg       0.97      0.97      0.97     13958
weighted avg       0.97      0.97      0.97     13958


Acuuracy Of the Model:  0.9727038257630033



F1 score Of the Model:  0.9723090340867795



Precision  Of the Model:  0.9937602139355222



Recall  Of the Model:  0.9517643710870802
```