National College of Ireland

Configuration Manual

MSc Research Project
MSc. Data Analytics

Shantanu Deshpande
Student ID: x18125514

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shantanu Deshpande
Student ID:	x18125514
Programme:	MSc. in Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Vladimir Milosavljevic
Submission Due Date:	12/12/2019
Project Title:	Corporate Bankruptcy Prediction using Machine Learning Techniques
Word Count:	1052
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:	11th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

Corporate Bankruptcy Prediction using Machine Learning Techniques

Shantanu Deshpande
X18125514
MSc. Data Analytics
National College of Ireland

1. Introduction

The purpose of this document is to give a brief information about the steps involved in implementing this project. The aim of this project was to evaluate the performance of a classifier in predicting corporate bankruptcy using a novel data pre-processing technique involving a feature selection strategy and a resampling strategy. The second objective was to compare the performance with few other classifiers and evaluate which model gave better prediction performance. The tools and techniques that were used for achieving the specified objectives have been mentioned in the remaining part of this manual.

2. System Specifications

The system configuration on which this research project has been carried out is mentioned below:

- Operating System: Windows 10 Home
- Installed Memory (RAM): 8.0 GB
- Hard Drive: 1024 GB HDD
- Processor: Intel® Core™ i5-3337U CPU @ 1.80GHz

3. Tools/Technologies

For implementing this project, Python programming language has been used whereas Jupyter Notebook has been used as the Integrated Development Environment (IDE). The visualizations have been performed on Tableau. The specific versions of respective platform/language is mentioned below:

- Python 3.7.2
- Jupyter Notebook Server v. **6.0.2**
- Tableau Desktop v. 2018.2.0 64-bit

4. Environment Setup

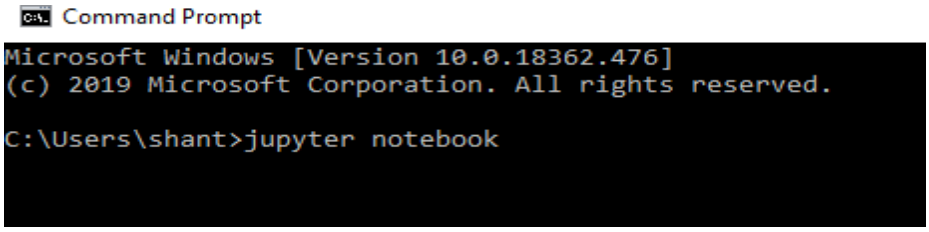
The initial and foremost step in the execution of the project is the installation of the required platform and languages.

- Python has been downloaded and installed using the following link. [1]

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		02a75015f7cd845e27b85192bb0ca4cb	22897802	SIG
XZ compressed source tarball	Source release		df6ec36011808205beda239c72f947cb	17042320	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	d8ff07973bc9c009de80c269fd7efcca	34405674	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	0fc95e9f6d6b4881f3b499da338a9a80	27766090	SIG
Windows help file	Windows		941b7d6279c0d4060a927a65dcab88c4	8092167	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	f81568590bef56e5997e63b434664d58	7025085	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	ff258093f0b3953c886192dec9f52763	26140976	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	8de2335249d84fe1eeb61ec25858bd82	1362888	SIG
Windows x86 embeddable zip file	Windows		26881045297dc1883a1d61baffeecaf0	6533256	SIG
Windows x86 executable installer	Windows		38156b62c0cbcb03bfddeb86e66c3a0f	25365744	SIG
Windows x86 web-based installer	Windows		1e6c626514b72e21008f8cd53f945f10	1324648	SIG

- Jupyter has been installed by following the installation guide given on following link[2]. To start the Jupyter Notebook, we need to enter following command in CMD prompt.



- After execution of complete project, the results were visualized using a visualization software, Tableau, the steps for downloading and installation is given in following link.[3]

5. Data Collection

Data for this project has been collected from UCI Machine Learning Repository[4] which is a public repository. The data description is given as below:

[1] https://www.python.org/downloads/release/python-372/
[2] https://jupyter.org/install
[3] https://www.tableau.com/products/desktop/download
[4] https://archive.ics.uci.edu/ml/datasets/Polish+companies+bankruptcy+data

Polish companies bankruptcy data Data Set

Download: Data Folder, Data Set Description

Abstract: The dataset is about bankruptcy prediction of Polish companies.The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013.

Data Set Characteristics:	Multivariate	Number of Instances:	10503	Area:	Business
Attribute Characteristics:	Real	Number of Attributes:	64	Date Donated	2016-04-11
Associated Tasks:	Classification	Missing Values?	Yes	Number of Web Hits:	88596

Source:

Creator: Sebastian Tomczak
-- Department of Operations Research, WrocÅ‚aw University of Science and Technology, wybrzeÅ¼e WyspiaÅ„skiego 27, 50-370, WrocÅ‚aw, Poland

Donor: Sebastian Tomczak (sebastian.tomczak '@' pwr.edu.pl), Maciej Zieba (maciej.zieba '@' pwr.edu.pl), Jakub M. Tomczak (jakub.tomczak '@' pwr.edu.pl), Tel. (+48) 71 320 44 53

Data Set Information:

The dataset is about bankruptcy prediction of Polish companies. The data was collected from Emerging Markets Information Service (EMIS, [Web Link]), which is a database containing information on emerging markets around the world. The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013.
Basing on the collected data five classification cases were distinguished, that depends on the forecasting period:
- 1stYear â€" the data contains financial rates from 1st year of the forecasting period and corresponding class label that indicates bankruptcy status after 5 years. The data contains 7027 instances (financial statements), 271 represents bankrupted companies, 6756 firms that did not bankrupt in the forecasting period.
- 2ndYear â€" the data contains financial rates from 2nd year of the forecasting period and corresponding class label that indicates bankruptcy status after 4 years. The data contains 10173 instances (financial statements), 400 represents bankrupted companies, 9773 firms that did not bankrupt in the forecasting period.
- 3rdYear â€" the data contains financial rates from 3rd year of the forecasting period and corresponding class label that indicates bankruptcy status after 3 years. The data contains 10503 instances (financial statements), 495 represents bankrupted companies, 10008 firms that did not bankrupt in the forecasting period.
- 4thYear â€" the data contains financial rates from 4th year of the forecasting period and corresponding class label that indicates bankruptcy status after 2 years. The data contains 9792 instances (financial statements), 515 represents bankrupted companies, 9277 firms that did not bankrupt in the forecasting period.
- 5thYear â€" the data contains financial rates from 5th year of the forecasting period and corresponding class label that indicates bankruptcy status after 1 year. The data contains 5910 instances (financial statements), 410 represents bankrupted companies, 5500 firms that did not bankrupt in the forecasting period.

- The dataset contained 64 financial attributes and a dichotomous categorical variable depicting the class of company whether bankrupt or non-bankrupt.

Attribute Information:

X1 net profit / total assets
X2 total liabilities / total assets
X3 working capital / total assets
X4 current assets / short-term liabilities
X5 [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365
X6 retained earnings / total assets
X7 EBIT / total assets
X8 book value of equity / total liabilities
X9 sales / total assets
X10 equity / total assets
X11 (gross profit + extraordinary items + financial expenses) / total assets
X12 gross profit / short-term liabilities
X13 (gross profit + depreciation) / sales
X14 (gross profit + interest) / total assets
X15 (total liabilities * 365) / (gross profit + depreciation)
X16 (gross profit + depreciation) / total liabilities
X17 total assets / total liabilities
X18 gross profit / total assets
X19 gross profit / sales
X20 (inventory * 365) / sales
X21 sales (n) / sales (n-1)
X22 profit on operating activities / total assets
X23 net profit / sales
X24 gross profit (in 3 years) / total assets
X25 (equity - share capital) / total assets
X26 (net profit + depreciation) / total liabilities

X27 profit on operating activities / financial expenses
X28 working capital / fixed assets
X29 logarithm of total assets
X30 (total liabilities - cash) / sales
X31 (gross profit + interest) / sales
X32 (current liabilities * 365) / cost of products sold
X33 operating expenses / short-term liabilities
X34 operating expenses / total liabilities
X35 profit on sales / total assets
X36 total sales / total assets
X37 (current assets - inventories) / long-term liabilities
X38 constant capital / total assets
X39 profit on sales / sales
X40 (current assets - inventory - receivables) / short-term liabilities
X41 total liabilities / ((profit on operating activities + depreciation) * (12/365))
X42 profit on operating activities / sales
X43 rotation receivables + inventory turnover in days
X44 (receivables * 365) / sales
X45 net profit / inventory
X46 (current assets - inventory) / short-term liabilities
X47 (inventory * 365) / cost of products sold
X48 EBITDA (profit on operating activities - depreciation) / total assets
X49 EBITDA (profit on operating activities - depreciation) / sales
X50 current assets / total liabilities
X51 short-term liabilities / total assets
X52 (short-term liabilities * 365) / cost of products sold)
X53 equity / fixed assets
X54 constant capital / fixed assets
X55 working capital
X56 (sales - cost of products sold) / sales
X57 (current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
X58 total costs /total sales
X59 long-term liabilities / equity
X60 sales / inventory
X61 sales / receivables
X62 (short-term liabilities *365) / sales
X63 sales / short-term liabilities
X64 sales / fixed assets

6. Implementation:

The complete code for this project is available on the following GitHub repository:

https://github.com/shantanudeshpande94/bankruptcy-prediction

The various processes involved in the implementation of this project are discussed step by step below-

A] Data Preparation and Storage

The data folder contained individual files that have been separated in five classification cases based on the forecasting period.

- All the files were in an ARFF format and to use it for further study, an openly available Python code[5] has been used to convert it into CSV format.
- For this study, the data file '5th Year' has been used that contained financial attributes from 5th year and the class label showing bankruptcy status after a year.
- We set the seed first for our code here as this will preserve the data samples and the results that will be obtained.
- The data file was then stored in CSV format on Github and then imported in GitHub using the below code.

```
In [1]: #importing necessary packages
        import scipy as sp
        import numpy as np
        import pandas as pd
        import pandas_profiling as pp
        from sklearn import datasets, linear_model
        from sklearn.model_selection import train_test_split
        from imblearn.metrics import classification_report_imbalanced
        from matplotlib import pyplot as plt
        import random

        #set seed
        random.seed(420)

        #loading csv into dataframe
        url = 'https://raw.githubusercontent.com/shantanudeshpande94/bankruptcy-prediction/master/5year.csv'
        bankruptcy_df = pd.read_csv(url)
```

B] Exploratory Data Analysis

Before beginning with our pre-processing stage, it is important to understand the dataset so that

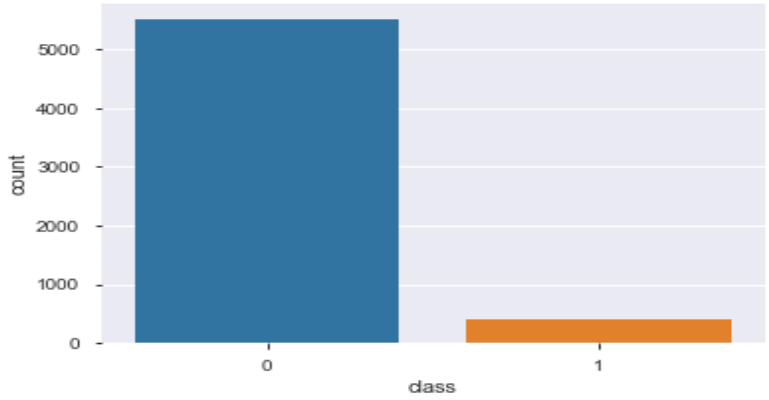
[5] https://github.com/haloboy777/arfftocsv

we are aware of the further steps to be undertaken as part of cleaning and preprocessing.

- To check the class distribution, a bar graph was plotted which gives a clear picture about the number of instances in both the classes. The **seaborn** package was imported for this.

```
import seaborn as sns
sns.countplot(bankruptcy_df['class'])
```

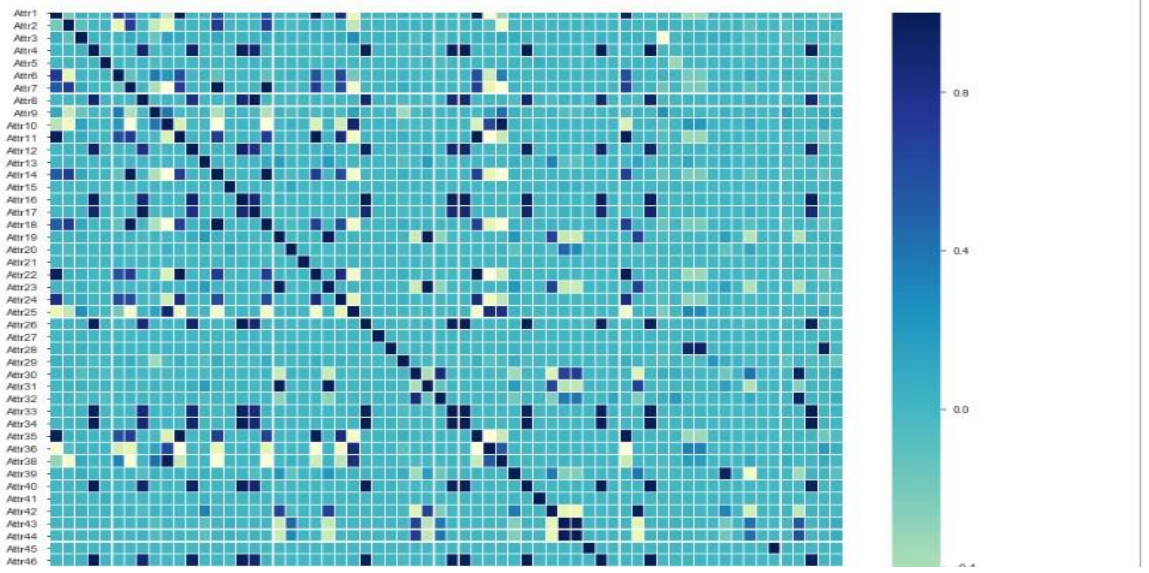
```
5]: <matplotlib.axes._subplots.AxesSubplot at 0x23c3093c4a8>
```



- As the dataset consisted of 64 variables, there can be a possibility that few of these variables may be correlated with each other. This may reduce the performance of the model and also increase the computational time and resources. Thus, the multicollinearity test was performed using the following code-

```
In [15]: #correlation analysis

corrmat = bankruptcy_df.corr()

corrmat.to_csv("correlation-analysis-5.csv")

#plotting correlation graph

import seaborn as sns

f, ax = plt.subplots(figsize =(15, 15))
sns.heatmap(corrmat, ax = ax, cmap ="YlGnBu", linewidths = 0.1)
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x2738a7b3cf8>
```



- Further, individual variables were explored in more detail for missing values and skewness using **pandas_profiling** package. The code for the same is mentioned below-

```
In [ ]: import pandas_profiling as pp

        #check profile report of the dataset

        profile = bankruptcy_df.profile_report(title='Pandas Profiling Report')
        profile.to_file(output_file="output.html")
```

C] Data Cleaning

Before proceeding with the model building phase, we must clean our data and perform transformations so that the model can give optimum performance. The various steps involved in this activity include converting datatype of variables, removal of duplicate rows, dropping variables with high multicollinearity, removing columns with large number of missing values and imputing missing values with mean in rest of the columns etc. The code for each step is shown in the below images-

- Replace special characters ('?') with NA values

```
#to replace '?' with NA values

bankruptcy_df.replace({'?': None},inplace =True)
```

- Converting all variables to numeric datatype

```
#converting from object to float64

bankruptcy_df['Attr1'] = pd.to_numeric(bankruptcy_df['Attr1'])
bankruptcy_df['Attr2'] = pd.to_numeric(bankruptcy_df['Attr2'])
bankruptcy_df['Attr3'] = pd.to_numeric(bankruptcy_df['Attr3'])
bankruptcy_df['Attr4'] = pd.to_numeric(bankruptcy_df['Attr4'])
bankruptcy_df['Attr5'] = pd.to_numeric(bankruptcy_df['Attr5'])
bankruptcy_df['Attr6'] = pd.to_numeric(bankruptcy_df['Attr6'])
bankruptcy_df['Attr7'] = pd.to_numeric(bankruptcy_df['Attr7'])
bankruptcy_df['Attr8'] = pd.to_numeric(bankruptcy_df['Attr8'])
bankruptcy_df['Attr9'] = pd.to_numeric(bankruptcy_df['Attr9'])
bankruptcy_df['Attr10'] = pd.to_numeric(bankruptcy_df['Attr10'])
bankruptcy_df['Attr11'] = pd.to_numeric(bankruptcy_df['Attr11'])
bankruptcy_df['Attr12'] = pd.to_numeric(bankruptcy_df['Attr12'])
bankruptcy_df['Attr13'] = pd.to_numeric(bankruptcy_df['Attr13'])
bankruptcy_df['Attr14'] = pd.to_numeric(bankruptcy_df['Attr14'])
bankruptcy_df['Attr15'] = pd.to_numeric(bankruptcy_df['Attr15'])
bankruptcy_df['Attr16'] = pd.to_numeric(bankruptcy_df['Attr16'])
bankruptcy_df['Attr17'] = pd.to_numeric(bankruptcy_df['Attr17'])
bankruptcy_df['Attr18'] = pd.to_numeric(bankruptcy_df['Attr18'])
bankruptcy_df['Attr19'] = pd.to_numeric(bankruptcy_df['Attr19'])
bankruptcy_df['Attr20'] = pd.to_numeric(bankruptcy_df['Attr20'])
bankruptcy_df['Attr21'] = pd.to_numeric(bankruptcy_df['Attr21'])
bankruptcy_df['Attr22'] = pd.to_numeric(bankruptcy_df['Attr22'])
```

- Dropping duplicate rows, imputing missing values by mean, removing columns with high multicollinearity

```
# dropping duplicate values

bankruptcy_df.drop_duplicates(keep=False,inplace=True)
```

```
#delete because of large number of missing values

del bankruptcy_df['Attr37']
```

```
#imputing missing values by mean

bankruptcy_df.fillna(bankruptcy_df.mean(), inplace=True)

#number of missing values

bankruptcy_df.isna().sum()
```

...

```
#delete because of high correlation

bankruptcy_df = bankruptcy_df.drop(['Attr11','Attr16','Attr18','Attr22','Attr23','Attr26','Attr31','Attr33',
       'Attr34','Attr35','Attr38','Attr4','Attr40','Attr44','Attr46','Attr48','Attr49','Attr50','Attr54',
       'Attr56','Attr60','Attr62','Attr63','Attr64','Attr7','Attr8'], axis = 1)
```

- **Feature Selection using Random Forest:**

  One of the novel approach we have followed is the use of Random forest feature selection technique for reducing the number of features. The packages used for implementing this are **RandomForestClassifier** and **SelectFromModel**. The code is mentioned below-

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
```

```
# Create a list of feature names
feature_labels = ['Attr1','Attr2','Attr3','Attr5','Attr6','Attr9','Attr10',
                  'Attr12','Attr13','Attr14','Attr15','Attr17','Attr19','Attr20',
                  'Attr21','Attr24','Attr25','Attr27','Attr28','Attr29','Attr30',
                  'Attr32','Attr36','Attr39',
                  'Attr41','Attr42','Attr43','Attr45','Attr47',
                  'Attr51','Attr52','Attr53','Attr55','Attr57','Attr58','Attr59',
                   'Attr61']
```

```
#Create X from the features
X = bankruptcy_df[feature_labels].values

# Create y from output
y = bankruptcy_df['class'].values.ravel()

# Create a random forest classifier
rf_clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)

# Train the classifier
rf_clf.fit(X, y)

# Print the name and gini importance of each feature
for feature in zip(feature_labels, rf_clf.feature_importances_):
    print(feature)
```

...

```
# Create a selector object that will use the random forest classifier to identify
# features that have an importance of more than 0.03
sfm = SelectFromModel(rf_clf, threshold=0.03)

# Train the selector
sfm.fit(X, y)
```

```
#Print the names of the most important features

for feature_list_index in sfm.get_support(indices=True):
    print(feature_labels[feature_list_index])
```

```
X_imp_features = sfm.transform(X)
```

```
#converting X_filtered to Dataframe

X_imp_features = pd.DataFrame(data=X_imp_features,
          index=np.arange(1, 5791),
          columns=np.arange(1, 8))
```

D] Modelling

This is the most important phase of the data mining process. To address the issue of class imbalance, a rarely used SMOTEENN resampling technique has been used on 4 different classifiers each time with Stratified K fold cross validation. To evaluate the improvement in results, same models are implemented without SMOTEENN resampling. The packages and codes are mentioned below one by one-

- **SMOTEENN + Random Forest Classifier**
  Packages used:
  - StratifiedKFold
  - SMOTEENN
  - Cross_val_score
  - recall_score
  - accuracy_score
  - confusion_matrix
  - RandomForestClassifier

```
from imblearn.combine import SMOTEENN
from sklearn.model_selection import cross_val_score
from sklearn.metrics import recall_score
```

```
from sklearn.model_selection import StratifiedKFold
from imblearn.combine import SMOTEENN
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

t_n = []
f_p = []
f_n = []
t_p = []
cv = StratifiedKFold(n_splits=5)
for train_idx, test_idx, in cv.split(X_imp_features, y):
    X_train, y_train = X[train_idx], y[train_idx]
    X_test, y_test = X[test_idx], y[test_idx]
    X_train, y_train = SMOTEENN().fit_resample(X_train, y_train)
    clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)
    clf.fit(X_train, y_train)
    y_rf_pred = clf.predict(X_test)
    accuracy_score(y_test, y_rf_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_rf_pred).ravel()
    t_n.append(tn)
    f_p.append(fp)
    f_n.append(fn)
    t_p.append(tp)
    print("True Negatives: ",tn)
    print("False Positives: ",fp)
    print("False Negatives: ",fn)
    print("True Positives: ",tp)
return np.array(t_n)
return np.array(f_p)
return np.array(f_n)
return np.array(t_p)
```

```
Avg_tn = round(sum(t_n)/len(t_n))
Avg_fp = round(sum(f_p)/len(f_p))
Avg_fn = round(sum(f_n)/len(f_n))
Avg_tp = round(sum(t_p)/len(t_p))
```

```
import math

#Accuracy
Accuracy_rf = (Avg_tp+Avg_tn)/(Avg_tp+Avg_tn+Avg_fp+Avg_fn)
print("Accuracy {:0.2f}".format(Accuracy_rf))

#Specificity
Specificity_rf = Avg_tn/(Avg_tn+Avg_fp)
print("Specificity {:0.2f}".format(Specificity_rf))

#Recall
Recall_rf = Avg_tp/(Avg_tp+Avg_fn)
print("Recall / Sensitivity {:0.2f}".format(Recall_rf))

#Precision
Precision_rf = Avg_tp/(Avg_tp + Avg_fp)

#GMean
GM_rf = math.sqrt(Specificity_rf*Recall_rf)
print("Geometric Mean Score {:0.2f}".format(GM_rf))
```

```
Accuracy 0.89
Specificity 0.90
Recall / Sensitivity 0.66
Geometric Mean Score 0.77
```

- **SMOTEENN + Decision Tree**
  Packages Used:
  - DecisionTreeClassifier

```
from sklearn.model_selection import StratifiedKFold
from imblearn.combine import SMOTEENN
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier

tn_dt = []
fp_dt = []
fn_dt = []
tp_dt = []
cv = StratifiedKFold(n_splits=5)
for train_idx_dt, test_idx_dt, in cv.split(X_imp_features, y):
    X_train_dt, y_train_dt = X[train_idx_dt], y[train_idx_dt]
    X_test_dt, y_test_dt = X[test_idx_dt], y[test_idx_dt]
    X_train_dt, y_train_dt = SMOTEENN().fit_resample(X_train_dt, y_train_dt)

    dt = DecisionTreeClassifier()
    dt.fit(X_train_dt, y_train_dt)
    y_dt_pred = dt.predict(X_test_dt)
    accuracy_score(y_test_dt, y_dt_pred)
    tn, fp, fn, tp = confusion_matrix(y_test_dt, y_dt_pred).ravel()
    tn_dt.append(tn)
    fp_dt.append(fp)
    fn_dt.append(fn)
    tp_dt.append(tp)
    print("True Negatives: ",tn)
    print("False Positives: ",fp)
    print("False Negatives: ",fn)
    print("True Positives: ",tp)
return np.array(tn_dt)
return np.array(fp_dt)
return np.array(fn_dt)
return np.array(tp_dt)
```

- **SMOTEENN + K Nearest Neighbour**

  Packages Used:

  - Neighbours
  - KNeighborsClassifier

```
from sklearn import neighbors
from sklearn.neighbors import KNeighborsClassifier

tn_knn = []
fp_knn = []
fn_knn = []
tp_knn = []
cv = StratifiedKFold(n_splits=5)
for train_idx_knn, test_idx_knn, in cv.split(X_imp_features, y):
    X_train_knn, y_train_knn = X[train_idx_knn], y[train_idx_knn]
    X_test_knn, y_test_knn = X[test_idx_knn], y[test_idx_knn]
    X_train_knn, y_train_knn = SMOTEENN().fit_resample(X_train_knn, y_train_knn)

    clf_knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
    clf_knn.fit(X_train_knn, y_train_knn)

    y_knn_pred = clf_knn.predict(X_test_knn)
    accuracy_score(y_test_knn, y_knn_pred)
    tn, fp, fn, tp = confusion_matrix(y_test_knn, y_knn_pred).ravel()
    tn_knn.append(tn)
    fp_knn.append(fp)
    fn_knn.append(fn)
    tp_knn.append(tp)
    print("True Negatives: ",tn)
    print("False Positives: ",fp)
    print("False Negatives: ",fn)
    print("True Positives: ",tp)
return np.array(tn_knn)
return np.array(fp_knn)
return np.array(fn_knn)
return np.array(tp_knn)
```

- **SMOTEENN + AdaBoost**
  Packages Used:
    - AdaBoostClassifier
    - LabelEncoder

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder

tn_ada = []
fp_ada = []
fn_ada = []
tp_ada = []
cv = StratifiedKFold(n_splits=5)
for train_idx_ada, test_idx_ada, in cv.split(X_imp_features, y):
    X_train_ada, y_train_ada = X[train_idx_ada], y[train_idx_ada]
    X_test_ada, y_test_ada = X[test_idx_ada], y[test_idx_ada]
    X_train_ada, y_train_ada = SMOTEENN().fit_resample(X_train_ada, y_train_ada)

    ada_classifier = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200)
    ada_classifier.fit(X_train_ada, y_train_ada)

    y_ada_pred = ada_classifier.predict(X_test_ada)
    accuracy_score(y_test_ada, y_ada_pred)
    tn, fp, fn, tp = confusion_matrix(y_test_ada, y_ada_pred).ravel()
    tn_ada.append(tn)
    fp_ada.append(fp)
    fn_ada.append(fn)
    tp_ada.append(tp)
    print("True Negatives: ",tn)
    print("False Positives: ",fp)
    print("False Negatives: ",fn)
    print("True Positives: ",tp)
return np.array(tn_ada)
return np.array(fp_ada)
return np.array(fn_ada)
return np.array(tp_ada)
```

- Similarly, these models were built without performing SMOTEENN. The code for one of the model, Random Forest, is shown below. Similar type of code was written for other classifiers as well without SMOTEENN.

```
from sklearn.model_selection import StratifiedKFold
from imblearn.combine import SMOTEENN
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

t_n = []
f_p = []
f_n = []
t_p = []
cv = StratifiedKFold(n_splits=5)
for train_idx, test_idx, in cv.split(X_imp_features, y):
    X_train, y_train = X[train_idx], y[train_idx]
    X_test, y_test = X[test_idx], y[test_idx]
    #X_train, y_train = SMOTEENN().fit_resample(X_train, y_train)
    clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)
    clf.fit(X_train, y_train)
    y_rf_pred = clf.predict(X_test)
    accuracy_score(y_test, y_rf_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_rf_pred).ravel()
    t_n.append(tn)
    f_p.append(fp)
    f_n.append(fn)
    t_p.append(tp)
    print("True Negatives: ",tn)
    print("False Positives: ",fp)
    print("False Negatives: ",fn)
    print("True Positives: ",tp)
return np.array(t_n)
return np.array(f_p)
return np.array(f_n)
return np.array(t_p)
```

Further to the implementation of all the models, the performance of these models was evaluated based on different metrics. These metrics were chosen based on literature review and were compared with one another visually by plotting a bar graph. The code for which is given below-

- Accuracy Plot

```
Accuracy_comparison = {'Random Forest': Accuracy_rf, 'Decision Tree': Accuracy_dt, 'KNN': Accuracy_knn, 'Adaboost': Accuracy
```

```
Accuracy_comparison
```

```
9]: {'Random Forest': 0.8861087144089732,
     'Decision Tree': 0.8556611927398444,
     'KNN': 0.7409326424870466,
     'Adaboost': 0.8807260155574762}
```

```
plt.bar(range(len(Accuracy_comparison)), list(Accuracy_comparison.values()), align='center')
plt.xticks(range(len(Accuracy_comparison)), list(Accuracy_comparison.keys()))

plt.show()
```

- Sensitivity Plot

```
Sensitivity_comparison = {'Random Forest': Recall_rf, 'Decision Tree': Recall_dt, 'KNN': Recall_knn, 'Adaboost': Recall_ada}
```

```
Sensitivity_comparison
```

```
2]: {'Random Forest': 0.6585365853658537,
     'Decision Tree': 0.654320987654321,
     'KNN': 0.6049382716049383,
     'Adaboost': 0.7283950617283951}
```

```
plt.bar(range(len(Sensitivity_comparison)), list(Sensitivity_comparison.values()), align='center')
plt.xticks(range(len(Sensitivity_comparison)), list(Sensitivity_comparison.keys()))

plt.show()
```

- Specificity Plot

```
Specificity_comparison = {'Random Forest': Specificity_rf, 'Decision Tree': Specificity_dt, 'KNN': Specificity_knn, 'Adaboost
```

```
Specificity_comparison
```

```
]: {'Random Forest': 0.903435468895079,
    'Decision Tree': 0.870817843866171,
    'KNN': 0.7511606313834726,
    'Adaboost': 0.8921933085501859}
```

```
plt.bar(range(len(Specificity_comparison)), list(Specificity_comparison.values()), align='center')
plt.xticks(range(len(Specificity_comparison)), list(Specificity_comparison.keys()))

plt.show()
```

- Geometric Mean of Specificity & Sensitivity Plot

```
Gmean_comparison = {'Random Forest': GM_rf, 'Decision Tree': GM_dt, 'KNN': GM_knn, 'Adaboost': GM_ada}
```

```
Gmean_comparison
```

```
]: {'Random Forest': 0.7713269791628996,
    'Decision Tree': 0.7548472637994519,
    'KNN': 0.6740962943428721,
    'Adaboost': 0.8061446520662862}
```

```
plt.bar(range(len(Gmean_comparison)), list(Gmean_comparison.values()), align='center')
plt.xticks(range(len(Gmean_comparison)), list(Gmean_comparison.keys()))

plt.show()
```

7. Conclusion

Through the information mentioned in the above sections, the complete implementation process of this project has been explained in a concise, detailed and sequential manner. The necessary packages that were required have also been mentioned wherever they were used and the entire code has been published on GitHub repository, the link for which is mentioned in the Implementation section.