Configuration Manual

MSc Research Project
Programme Name

Shakoor Ahmad Bhat
Student ID: x19236280

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shakoor Ahmad Bhat
Student ID:	x19236280
Programme:	Programme Name
Year:	2018
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	20/12/2018
Project Title:	Configuration Manual
Word Count:	1731
Page Count:	23

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:	14th August 2021

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

Shakoor Ahmad Bhat
x19236280

1 Introduction

The motive of this report is to provide details of the process followed during the coding phase of the research project. Hardware and software configurations are defined to reproduce the research in future. This contains the programming and employment stages for glossy code execution and the steps taken for executing the code.

2 System Configuration

2.1 Hardware Configuration

The hardware description and specification is shown in Figure 1 on which the code is executed:

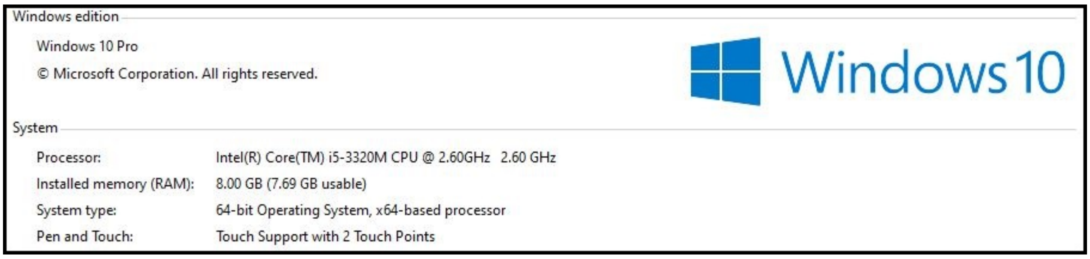


Figure 1: Hardware configuration of the system

2.2 Software Configuration

This section provides the details of the software and its specifications.

2.2.1 Anaconda - Jupyter Notebook:

Anaconda is a open source[1], Anaconda is a Python and R distribution (prebuilt and preconfigured collection of packages) that is commonly used for data science. Anaconda Navigator is a GUI tool that is included in the Anaconda distribution and makes it easy to configure, install, and launch tools such as Jupyter Notebook. It can downloaded from the official website of Anaconda[2]. The download options for Windows, MacOS and Linux

[1] https://www.anaconda.com/
[2] https://www.anaconda.com/products/individual

is shown in Figure 2

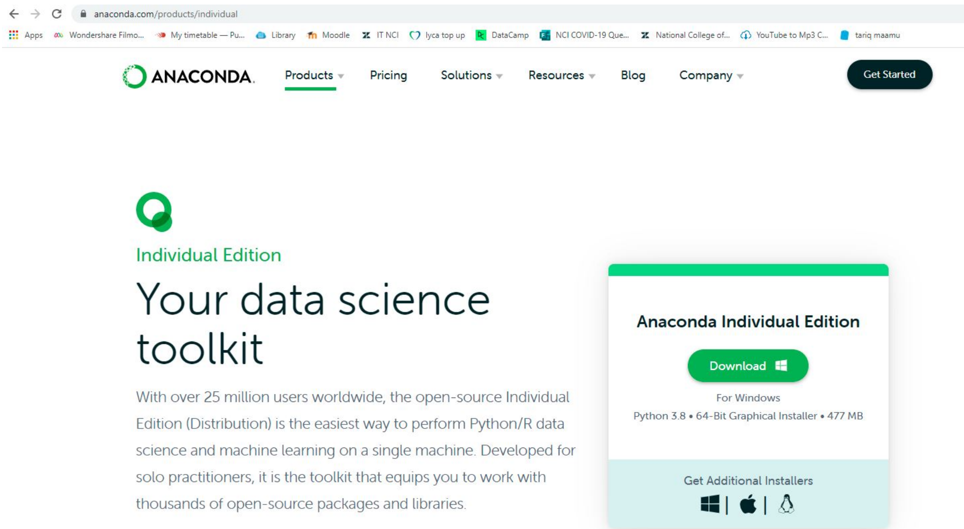


Figure 2: Anaconda Installer Download Page

After installing the anaconda, the home page of Anaconda Navigator will display different Integrated Development Environment (IDE) Figure 3. Jupiter Notebook IDE is launched for code execution and development of various models using Python version 3.

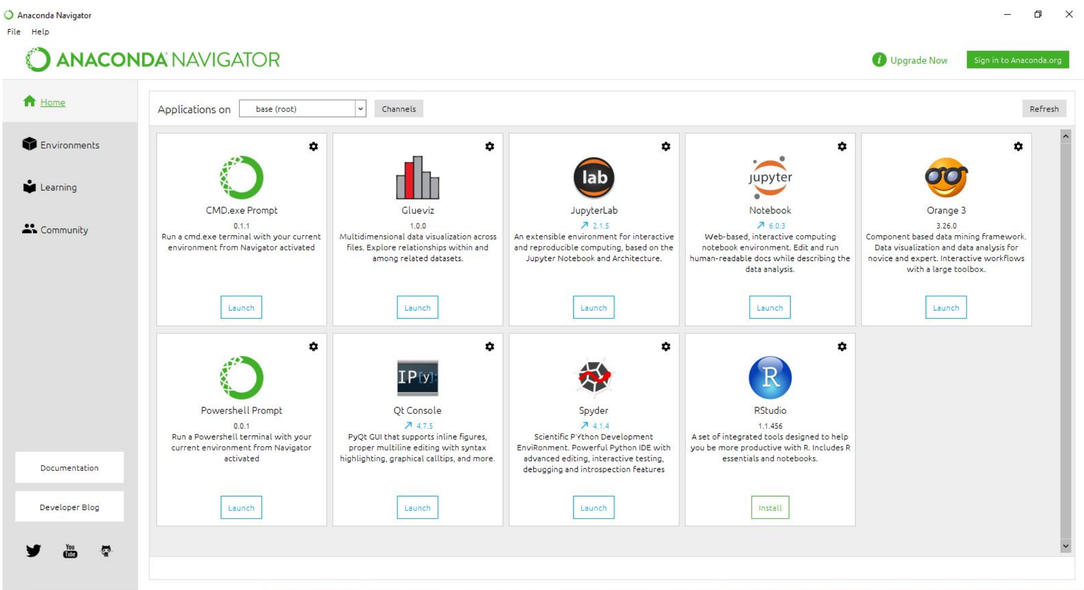


Figure 3: Anaconda Navigator Home Page

2.2.2 Other Softwares

For report documentation we used Overleaf, Figure 4 shows the overleaf home page for report documentation.

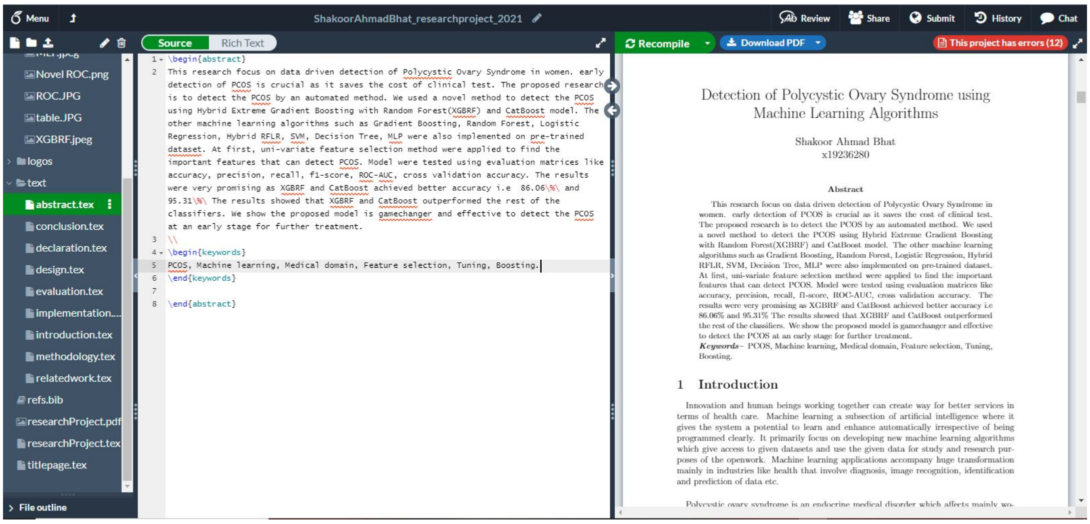


Figure 4: Overleaf Project

The data visualization is done by using Scikit-learn package[3] in Python as shown in Figure 5. The line chart shows the comparison of all models based on ROC Curve plot.

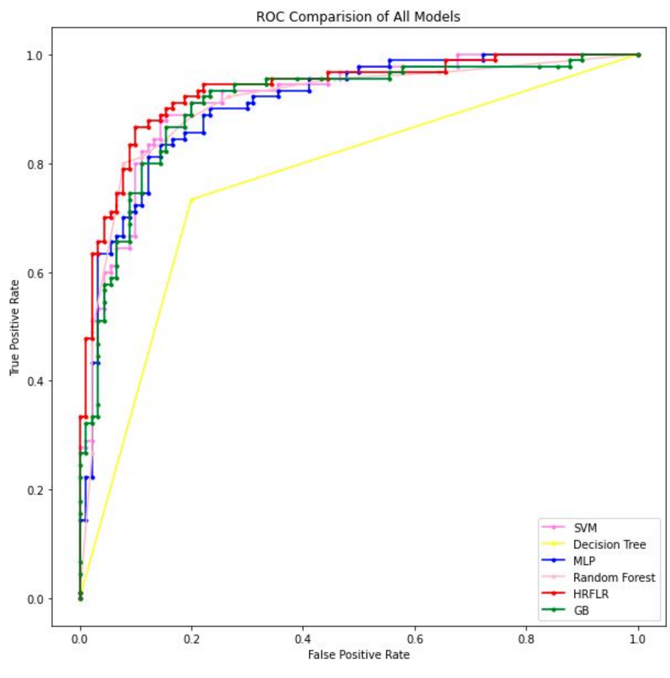


Figure 5: Data visualization of all models based on ROC Curve plot

[3]https://scikit-learn.org/0.24/visualizations.html

3 Data Preperation

The dataset was taken from Kaggle repository[4] as shown in Figure 6. The dataset has one folder and a csv file is provided with category (0 as Normal women and 1 as PCOS women).

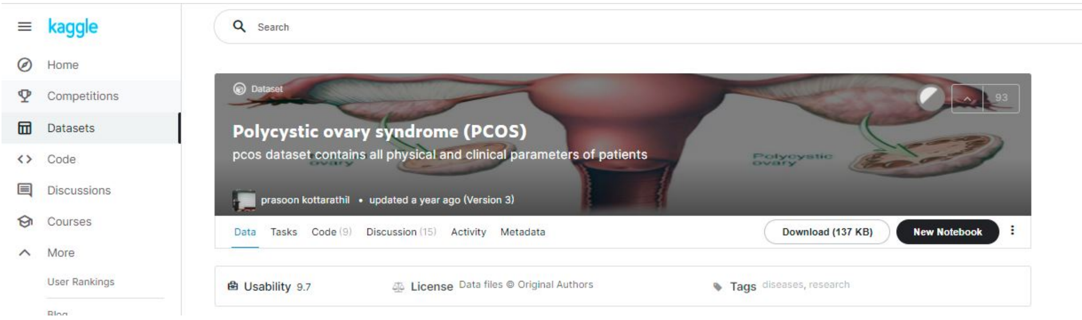


Figure 6: Polycystic ovary syndrome (PCOS) dataset for research project

After this dataset was loaded into the python by using this code as shown in Figure 7.

Loading Dataset

```
######################## Data Selection ###################################
# Reading the data from csv
dataset = pd.read_csv("C:\\Users\\2Pac\\Downloads\\PCOSNEW.csv")
```

Figure 7: Loading the PCOS datset

Then we imported all the libraries as shown in Figure 8.

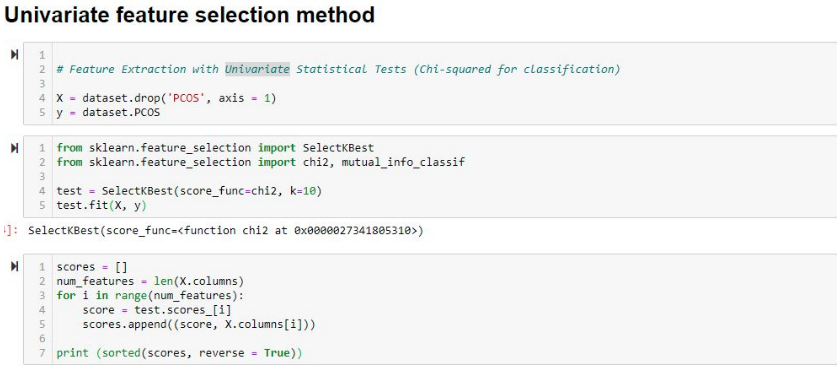
```
from __future__ import print_function
import lime
import lime.lime_tabular
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from time import time
from collections import Counter
from sklearn.inspection import permutation_importance
import sklearn.datasets
import sklearn.ensemble
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import model_selection
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

## Libraries for Upsampling and splitting of dataset into train and test
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

## Libraries for Checking various model performances like Confusion Matrix,Accuracy_Score etc
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

Figure 8: Importing Libraries

Further, after checking the missing values in our dataset by using isna().sum() function.

[4] https://www.kaggle.com/prasoonkottarathil/polycystic-ovary-syndrome-pcos

Univariate Feature selection method[5] were implemented to select the top 10 features by using SelectKbest and Chi2 packages which will help us to detect the PCOS, code shown in Figure 9.


Univariate feature selection method

```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for classification)

X = dataset.drop('PCOS', axis = 1)
y = dataset.PCOS
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif

test = SelectKBest(score_func=chi2, k=10)
test.fit(X, y)
```

```
]: SelectKBest(score_func=<function chi2 at 0x0000027341805310>)
```

```
scores = []
num_features = len(X.columns)
for i in range(num_features):
    score = test.scores_[i]
    scores.append((score, X.columns[i]))

print (sorted(scores, reverse = True))
```

Figure 9: Implementation of Univariate Feature Selection Method

4 Data Transformation

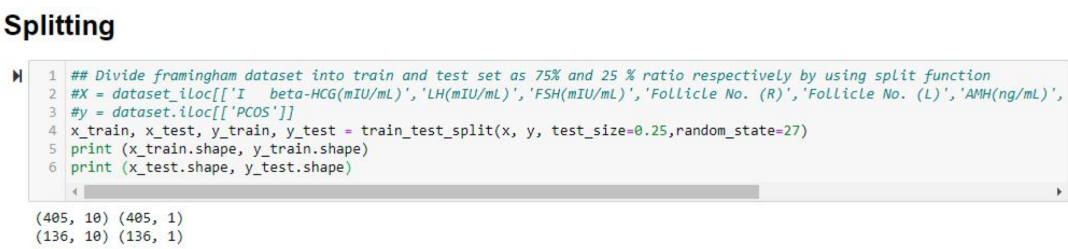
After the data pre-processing data, new dataframe is created based on top 10 features as shown in Figure 10.


New Dataframe based on top 10 features

```
#Making a new dataframe after implementing feature selection
x = dataset[['Avg. F size (R) (mm)','FSH(mIU/mL)','Follicle No. (R)','Follicle No. (L)','AMH(ng/mL)','FSH/LH','Cycle(R/I
y = dataset[["PCOS"]]
```

Figure 10: New Dataframe based on top 10 features

After making a new dataframe, the data is split into train and test having test size as 0.25 and random state as 27 as shown in Figure 11.


Splitting

```
## Divide framingham dataset into train and test set as 75% and 25 % ratio respectively by using split function
#X = dataset_iloc[['I   beta-HCG(mIU/mL)','LH(mIU/mL)','FSH(mIU/mL)','Follicle No. (R)','Follicle No. (L)','AMH(ng/mL)',
#y = dataset.iloc[['PCOS']]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,random_state=27)
print (x_train.shape, y_train.shape)
print (x_test.shape, y_test.shape)
```

```
(405, 10) (405, 1)
(136, 10) (136, 1)
```

Figure 11: Spliiting of dataset into train and test set

Furthermore, SMOTE was used to solve the imbalance problem by randomly increasing

[5]https://github.com/solegalli/feature-selection-for-machine-learning/blob/master/05-Filter-Statistical-Tests/05.3-Univariate-selection.ipynb

minority class examples by replicating them. It was used on training and testing set separately as shown in Figure 12.

SMOTE

```
## Upsampling the Training set

sm = SMOTE(random_state=23, sampling_strategy='minority')
x_train_sm, y_train_sm = sm.fit_resample(x_train, y_train)
print(len(x_train_sm), len(y_train_sm))
```

```
## Upsampling the Testing set
sm_test = SMOTE(random_state=23, sampling_strategy='minority')
x_test_sm, y_test_sm = sm_test.fit_resample(x_test, y_test)
print(len(x_test_sm), len(y_test_sm))
```

Figure 12: SMOTE on training and testing set

5 Implementation of Baseline Models

After data pre-processing and data transformation, data can be used for implementation using the baseline models such as Graident Boosting, Random Forest, Logistic Regression, HRFLR, SVM, Decision Tree, MLP.

5.1 Gradient Boosting

5.1.1 Model Building

After importing the Gradient Boosting classifier, as it helps to minimize the loss, or the difference between the actual class value of the training example and the predicted class value. The hyper parameter settings were ($n\_estimators = 20, learning\_rate = 0.5, max\_features = 2, max\_depth = 2, random\_state = 0$). The code for model development of Gradient Boosting is shown in Figure 13. Moreover, Data were prepared for start and end time by using fit() function and prediction were made on training and testing time.

Gradient Boosting Classifier ¶

```
#from sklearn.metrics import mean_squared_error,r2_score
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=20, learning_rate = 0.5, max_features=2, max_depth = 2, random_state = 0)
gb.fit(x_train, y_train)
results = {}
```

```
C:\Users\2Pac\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passe
d when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
#Training the model
start = time()
gb.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
GB_Prediction = gb.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 13: Model building of Gradient Boosting

5.1.2 Model Evaluation

The evaluation matrices were accuracy, precision, recall, f1,score, ROC curve plot and AUC score. Further, confusion matrix and classification report is generated using sklearn.metrics[6]. The code and calculation for these matrices is shown in Figure 14

```python
## Accuracy Score
GB_Accuracy = accuracy_score(y_test_sm, GB_Prediction)
print("The accuracy score for Gradient Boosting in percentage is: "+"{:.2f}".format(GB_Accuracy*100))

## Precision
GB_Precision = precision_score(y_test_sm, GB_Prediction)
print("The precision score for Gradient Boosting is: "+"{:.2f}".format(GB_Precision))

## Recall Feature
GB_Recall = recall_score(y_test_sm, GB_Prediction)
print("The recall score for Gradient Boosting is: "+"{:.2f}".format(GB_Recall))
## F1 Score
GB_F1Score = f1_score(y_test_sm, GB_Prediction)
print("The F1 Score for Gradient Boosting is: "+"{:.2f}".format(GB_F1Score))

## Confusion Matrix
GB_Confusion_Matrix=confusion_matrix(y_test_sm,GB_Prediction)
print("Confusion_Matrix: \n\n",GB_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,GB_Prediction,zero_division=1,target_names=target_names))

```

Figure 14: Model evaluation of Gradient Boosting

After implementing the above code we got the output for all the matrices as shown in Figure 15.

```
The accuracy score for Gradient Boosting in percentage is: 82.78
The precision score for Gradient Boosting is: 0.85
The recall score for Gradient Boosting is: 0.80
The F1 Score for Gradient Boosting is: 0.82
Confusion_Matrix:

 [[77 13]
 [18 72]]

              precision    recall  f1-score   support

     class 0       0.81      0.86      0.83        90
     class 1       0.85      0.80      0.82        90

    accuracy                           0.83       180
   macro avg       0.83      0.83      0.83       180
weighted avg       0.83      0.83      0.83       180
```

Figure 15: Confusion matrix and Classification report of Gradient Boosting

Now we will check how the model is expected to perform in general when used to make predictions on data not used during the training of the model by using the K- fold Cross validation accuracy selecting k=10,20,30,40.The code is shown in Figure 16.

[6] `https://scikit-learn.org/0.15/modules/model_evaluation.html`

```
## Cross Validation
#for K=10
GB_accuracies = cross_val_score(estimator = gb, X= x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(GB_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(GB_accuracies.std()*100))

#for K=20
GB_accuracies = cross_val_score(estimator = gb, X= x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(GB_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(GB_accuracies.std()*100))

#for K=30
GB_accuracies = cross_val_score(estimator = gb, X= x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(GB_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(GB_accuracies.std()*100))

#for K=40
GB_accuracies = cross_val_score(estimator = gb, X= x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(GB_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(GB_accuracies.std()*100))
```

Figure 16: Model evaluation of K fold cross validation of Gradient Boosting

After implementing the above code we got the output of K fold cross validation accuracy for Gradient Boosting as shown in Figure 17.

```
Cross Validation Accuracy: 85.06 %
Cross Validation Standard Deviation: 5.82 %

Cross Validation Accuracy: 86.73 %
Cross Validation Standard Deviation: 6.92 %

Cross Validation Accuracy: 85.96 %
Cross Validation Standard Deviation: 8.87 %

Cross Validation Accuracy: 86.06 %
Cross Validation Standard Deviation: 10.21 %
```

Figure 17: Output of K fold cross validation accuracy

5.2 Random Forest

5.2.1 Model Building

After importing the Random Forest classifier, as it builds multiple decision trees and merges them together to get a more accurate and stable prediction. The hyper parameter settings were $(n\_estimators = 10, criterion = entropy, random\_state = 0)$.The code for model development of RF Classifier is shown in Figure 18.

Random Forest Classifier

```python
RF_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy',random_state = 0)

results = {}
#Training the model
start = time()
RF_classifier.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
RF_Prediction = RF_classifier.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 18: Model building for Random Forest

5.2.2 Model Evaluation

The RF model is evaluated in a same way as explained in Gradient Boosting. The code is same for RF as well using the RF_classifier.fit() and RF_classifier.predict() function for evaluating the matrix as shown in Figure 19. Figure 20 shows the output of confusion matrix, classification report, and cross validation accuracy when k=10,20,30,40.

```python
## Accuracy Score
RF_Accuracy = accuracy_score(y_test_sm, RF_Prediction)
print("The accuracy score for Random Forest in percentage is: "+"{:.2f}".format(RF_Accuracy*100))

## Precision
RF_Precision = precision_score(y_test_sm, RF_Prediction)
print("The precision score for Random Forest is: "+"{:.2f}".format(RF_Precision))

## Recall Feature
RF_Recall = recall_score(y_test_sm, RF_Prediction)
print("The recall score for Random Forest is: "+"{:.2f}".format(RF_Recall))
## F1 Score
RF_F1Score = f1_score(y_test_sm, RF_Prediction)
print("The F1 Score for Random Forest is: "+"{:.2f}".format(RF_F1Score))

## Confusion Matrix
RF_Confusion_Matrix=confusion_matrix(y_test_sm,RF_Prediction)
print("Confusion_Matrix: \n\n",RF_Confusion_Matrix, "\n" )
## Cross Validation
#for K=10
RF_accuracies = cross_val_score(estimator = RF_classifier, X= x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(RF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(RF_accuracies.std()*100))

#for K=20
RF_accuracies = cross_val_score(estimator = RF_classifier, X= x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(RF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(RF_accuracies.std()*100))

#for K=30
RF_accuracies = cross_val_score(estimator = RF_classifier, X= x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(RF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(RF_accuracies.std()*100))

#for K=40
RF_accuracies = cross_val_score(estimator = RF_classifier, X= x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(RF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(RF_accuracies.std()*100))
```

Figure 19: Model evaluation for Random Forest

```
The accuracy score for Random Forest in percentage is: 85.00
The precision score for Random Forest is: 0.88
The recall score for Random Forest is: 0.81
The F1 Score for Random Forest is: 0.84
Confusion_Matrix:

 [[80 10]
 [17 73]]

              precision    recall  f1-score   support

     class 0       0.82      0.89      0.86        90
     class 1       0.88      0.81      0.84        90

    accuracy                           0.85       180
   macro avg       0.85      0.85      0.85       180
weighted avg       0.85      0.85      0.85       180
```

```
Cross Validation Accuracy: 86.52 %
Cross Validation Standard Deviation: 5.38 %

Cross Validation Accuracy: 88.18 %
Cross Validation Standard Deviation: 7.00 %

Cross Validation Accuracy: 87.83 %
Cross Validation Standard Deviation: 7.88 %

Cross Validation Accuracy: 88.80 %
Cross Validation Standard Deviation: 9.44 %
```

Figure 20: Model evaluation output for Random Forest

5.3 Logistic regression

5.3.1 Model Building

After importing the logistic regression classifier, as it is used to predict the categorical dependent variable using a given set of independent variables. The hyper parameter setting are (max_iter=10000,random_state = 0). The code for development of Logistic Regression is shown in Figure 21.

LOGISTIC REGRESSION

```
LR_classifier = LogisticRegression(max_iter=10000,random_state = 0)

results = {}
#Training the model
start = time()
LR_classifier.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
LR_Prediction = LR_classifier.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 21: Model Building for Logistic Regression

5.3.2 Model Evaluation

The model is evaluated by using the two variables such as LR_classifier.fit() and LR_classifier.predict(). The code for accuracy, precision, recall, f1-score, cross validation accuracy is shown in Figure 22. The output confusion matrix, classification report and cross validation accuracy is shown in Figure 23.

```python
## Accuracy Score
LR_Accuracy = accuracy_score(y_test_sm, LR_Prediction)
print("The accuracy score for Logistic Regression in percentage is: "+"{:.2f}".format(LR_Accuracy*100))

## Precision
LR_Precision = precision_score(y_test_sm, LR_Prediction)
print("The precision score for Logistic Regression is: "+"{:.2f}".format(LR_Precision))
## Recall Feature
LR_Recall = recall_score(y_test_sm, LR_Prediction)
print("The recall score for Logistic Regression is: "+"{:.2f}".format(LR_Recall))

## F1 Score
LR_F1Score = f1_score(y_test_sm, LR_Prediction)
print("The F1 Score for Logistic Regression is: "+"{:.2f}".format(LR_F1Score))

## Confusion Matrix
LR_Confusion_Matrix=confusion_matrix(y_test_sm,LR_Prediction)
print("Confusion_Matrix: \n\n",LR_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,LR_Prediction,zero_division=1,target_names=target_names))
## Cross Validation
#for K=10
LR_accuracies = cross_val_score(LR_classifier, X = x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(LR_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(LR_accuracies.std()*100))

#for K=20
LR_accuracies = cross_val_score(LR_classifier, X = x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(LR_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(LR_accuracies.std()*100))

#for K=30
LR_accuracies = cross_val_score(LR_classifier, X = x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(LR_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(LR_accuracies.std()*100))

#for K=40
LR_accuracies = cross_val_score(LR_classifier, X = x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(LR_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(LR_accuracies.std()*100))
```

Figure 22: Model evaluation for Logistic Regression

```
The accuracy score for Logistic Regression in percentage is: 87.78
The precision score for Logistic Regression is: 0.88
The recall score for Logistic Regression is: 0.88
The F1 Score for Logistic Regression is: 0.88
Confusion_Matrix:

 [[79 11]
 [11 79]]

              precision    recall  f1-score   support

     class 0       0.88      0.88      0.88        90
     class 1       0.88      0.88      0.88        90

    accuracy                           0.88       180
   macro avg       0.88      0.88      0.88       180
weighted avg       0.88      0.88      0.88       180

Cross Validation Accuracy: 85.04 %
Cross Validation Standard Deviation: 4.87 %
Cross Validation Accuracy: 84.87 %
Cross Validation Standard Deviation: 7.16 %
Cross Validation Accuracy: 85.20 %
Cross Validation Standard Deviation: 8.17 %
Cross Validation Accuracy: 84.92 %
Cross Validation Standard Deviation: 9.90 %
```

Figure 23: Model evaluation output for Logistic Regression

5.4 Hybrid Random Forest and Logistic Regression (HRFLR)

5.4.1 Model Building

First the sub model were created using the estimators = [], than logistic model were defined using hyper parameter setting as (random_state = 0,C=1, max_iter=10000). After this three Random Forest models were defined such as model121, 122 and 123. At last HRFLR model were ensemble using the voting classifier package taking voting = soft. The code is shown in Figure 24.

```python
# Create the sub-model
HRFLR_estimators = []

# Defining 1 Logistic Regression Model
model11 = LogisticRegression(random_state = 0,C=1, max_iter=10000)
HRFLR_estimators.append(('logistic1', model11))


# Defining 3 Random Forest Models
model21 = RandomForestClassifier(random_state = 0)
HRFLR_estimators.append(('RF1', model21))

model22 = RandomForestClassifier(random_state = 0)
HRFLR_estimators.append(('RF2', model22))

model23 = RandomForestClassifier(random_state = 0)
HRFLR_estimators.append(('RF3', model23))


# Defining the HRFLM ensemble model
HRFLR_ensemble = VotingClassifier(HRFLR_estimators,voting='soft')

results = {}
#Training the model
start = time()
HRFLR_ensemble.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
HRFLR_Prediction = HRFLR_ensemble.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 24: Model evaluation for HRFLR

5.4.2 Model Evaluation

The model is evaluated by using the two variables such as HRFLR_ensemble.fit() and HRFLR_ensemble.predict(). The code for accuracy, precision, recall, f1-score, cross validation accuracy is shown in Figure 25. The output confusion matrix, classification report and cross validation accuracy with cross validation time is shown in Figure 26.

```python
## Accuracy Score
HRFLR_Accuracy = accuracy_score(y_test_sm, HRFLR_Prediction)
print("The accuracy score for HRFLR in percentage is: "+"{:.2f}".format(HRFLR_Accuracy*100))

## Precision
HRFLR_Precision = precision_score(y_test_sm, HRFLR_Prediction)
print("The precision score for HRFLR is: "+"{:.2f}".format(HRFLR_Precision))

## Recall Feature
HRFLR_Recall = recall_score(y_test_sm, HRFLR_Prediction)
print("The recall score for HRFLR is as follows: "+"{:.2f}".format(HRFLR_Recall))

## F1 Score
HRFLR_F1Score = f1_score(y_test_sm, HRFLR_Prediction)
print("The F1 Score for HRFLR is: "+"{:.2f}".format(HRFLR_F1Score))

## Confusion Matrix
HRFLR_Confusion_Matrix=confusion_matrix(y_test_sm,HRFLR_Prediction)
print("Confusion_Matrix: \n\n",HRFLR_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,HRFLR_Prediction,zero_division=1,target_names=target_names))
## Cross Validation
#for K=10
start = time()
HRFLR_accuracies = cross_val_score(HRFLR_ensemble, X = x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} ".format(HRFLR_accuracies.mean()))
print("Cross Validation Standard Deviation: {:.2f} %".format(HRFLR_accuracies.std()*100))
end = time()
results['Cross_Validation time'] = end - start
print("Cross_Validation time: "+"{:.2f}".format(results['Cross_Validation time']))

#for K=20
start = time()
HRFLR_accuracies = cross_val_score(HRFLR_ensemble, X = x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} ".format(HRFLR_accuracies.mean()))
print("Cross Validation Standard Deviation: {:.2f} %".format(HRFLR_accuracies.std()*100))
end = time()
results['Cross_Validation time'] = end - start
print("Cross_Validation time: "+"{:.2f}".format(results['Cross_Validation time']))

#for K=30
start = time()
HRFLR_accuracies = cross_val_score(HRFLR_ensemble, X = x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} ".format(HRFLR_accuracies.mean()))
print("Cross Validation Standard Deviation: {:.2f} %".format(HRFLR_accuracies.std()*100))
end = time()
results['Cross_Validation time'] = end - start
print("Cross_Validation time: "+"{:.2f}".format(results['Cross_Validation time']))

#for K=40
start = time()
HRFLR_accuracies = cross_val_score(HRFLR_ensemble, X = x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} ".format(HRFLR_accuracies.mean()))
print("Cross Validation Standard Deviation: {:.2f} %".format(HRFLR_accuracies.std()*100))
end = time()
results['Cross_Validation time'] = end - start
print("Cross_Validation time: "+"{:.2f}".format(results['Cross_Validation time']))
```

Figure 25: Model evaluation for HRFLR

```
The accuracy score for HRFLR in percentage is: 87.22
The precision score for HRFLR is: 0.90
The recall score for HRFLR is as follows: 0.83
The F1 Score for HRFLR is: 0.87
Confusion_Matrix:

 [[82  8]
 [15 75]]

              precision    recall  f1-score   support

     class 0       0.85      0.91      0.88        90
     class 1       0.90      0.83      0.87        90

    accuracy                           0.87       180
   macro avg       0.87      0.87      0.87       180
weighted avg       0.87      0.87      0.87       180

Cross Validation Accuracy: 0.89
Cross Validation Standard Deviation: 5.19 %
Cross_Validation time: 9.41
Cross Validation Accuracy: 0.89
Cross Validation Standard Deviation: 5.39 %
Cross_Validation time: 17.50
Cross Validation Accuracy: 0.89
Cross Validation Standard Deviation: 6.63 %
Cross_Validation time: 23.46
Cross Validation Accuracy: 0.90
Cross Validation Standard Deviation: 8.00 %
Cross_Validation time: 33.40
```

Figure 26: Model evaluation output for HRFLR

5.5 Feature Importance

By using the permutation_importance() function with (HRFLR_ensemble, x_train_sm, y_train_sm, n_repeats=10,random_state=0) as hyper parameter which will improve the efficiency and effectiveness of a predictive model on the problem. The code with output is shown in Figure 27.

Feature importance ¶

```
result = permutation_importance(HRFLR_ensemble, x_train_sm, y_train_sm, n_repeats=10,
                                random_state=0)
## Feature names in training set
feature_names= ['Avg. F size (R) (mm)','FSH(mIU/mL)','Follicle No. (R)','Follicle No. (L)','AMH(ng/mL)','FSH/LH','Cycle(

## Printing the features based on their importance
for i in result.importances_mean.argsort()[::-1]:
    if result.importances_mean[i] - 2 * result.importances_std[i] > 0:
        print(f"{feature_names[i]:<8}"
              f"{result.importances_mean[i]:.3f}"
              f" +/- {result.importances_std[i]:.3f}")
```

```
Follicle No. (R)0.230 +/- 0.016
Follicle No. (L)0.074 +/- 0.012
Cycle length(days)0.053 +/- 0.007
Cycle(R/I)0.043 +/- 0.006
BMI     0.042 +/- 0.004
AMH(ng/mL)0.040 +/- 0.004
Avg. F size (L) (mm)0.031 +/- 0.004
Avg. F size (R) (mm)0.021 +/- 0.003
FSH/LH  0.019 +/- 0.004
FSH(mIU/mL)0.018 +/- 0.004
```

Figure 27: Feature Importance based on HRFLR

5.6 Support Vector Machine

5.6.1 Model Building

SVM is developed by using the Support Vector Classifier SVC() function having random_state = 0,probability=True as hyper parameter. SVM algorithm creates a line or a hyperplane which separates the data into classes. The code is shown in Figure 28.

Support Vector Machines

```
SVM_classifier = SVC(random_state = 0,probability=True)

results = {}
#Training the model
start = time()
SVM_classifier.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
SVM_Prediction = SVM_classifier.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 28: Model building for SVM

5.6.2 Model Evaluation

The model is evaluated by using SVM_classifier.fit() and SVM_classifier.predict(). The code for accuracy, precision, recall, f1-score, cross validation accuracy is shown in Figure 29. The output confusion matrix, classification report and cross validation accuracy with cross validation time is shown in Figure 30.

```python
## Accuracy Score
SVM_Accuracy = accuracy_score(y_test_sm, SVM_Prediction)
print("The accuracy score for SVM in percentage is as follows: "+"{:.2f}".format(SVM_Accuracy*100))

## Precision
SVM_Precision = precision_score(y_test_sm, SVM_Prediction)
print("The precision score for SVM is as follows: "+"{:.2f}".format(SVM_Precision))

## Recall Feature
SVM_Recall = recall_score(y_test_sm, SVM_Prediction)
print("The recall score for SVM is as follows: "+"{:.2f}".format(SVM_Recall))

## F1 Score
SVM_F1Score = f1_score(y_test_sm, SVM_Prediction)
print("The F1 Score for SVM is as follows: "+"{:.2f}".format(SVM_F1Score))

## Confusion Matrix
SVM_Confusion_Matrix=confusion_matrix(y_test_sm,SVM_Prediction)
print("Confusion_Matrix: \n\n",SVM_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,SVM_Prediction,zero_division=1,target_names=target_names))

## Cross Validation
#for K=10
SVM_accuracies = cross_val_score(estimator = SVM_classifier, X= x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(SVM_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(SVM_accuracies.std()*100))

#for K=20
SVM_accuracies = cross_val_score(estimator = SVM_classifier, X= x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(SVM_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(SVM_accuracies.std()*100))

#for K=30
SVM_accuracies = cross_val_score(estimator = SVM_classifier, X= x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(SVM_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(SVM_accuracies.std()*100))

#for K=40
SVM_accuracies = cross_val_score(estimator = SVM_classifier, X= x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(SVM_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(SVM_accuracies.std()*100))
```

Figure 29: Model evaluation for SVM

```
********************************************

The accuracy score for SVM in percentage is as follows: 85.00
The precision score for SVM is as follows: 0.85
The recall score for SVM is as follows: 0.84
The F1 Score for SVM is as follows: 0.85
Confusion_Matrix:

 [[77 13]
 [14 76]]

              precision    recall  f1-score   support

     class 0       0.85      0.86      0.85        90
     class 1       0.85      0.84      0.85        90

    accuracy                           0.85       180
   macro avg       0.85      0.85      0.85       180
weighted avg       0.85      0.85      0.85       180

Cross Validation Accuracy: 84.68 %
Cross Validation Standard Deviation: 4.81 %

Cross Validation Accuracy: 85.76 %
Cross Validation Standard Deviation: 8.75 %

Cross Validation Accuracy: 84.83 %
Cross Validation Standard Deviation: 9.78 %

Cross Validation Accuracy: 84.92 %
Cross Validation Standard Deviation: 11.17 %
```

Figure 30: Model evaluation output for SVM

5.7 Decision Tree

5.7.1 Model Building

The model is created by using the DecisionTreeClassifier() function having criterion=entropy and random_state = 0. It is an framework to quantify the values of outcomes and the probabilities of achieving them because DT handles non-linear data sets effectively. The code is shown in Figure 31.

Decision Tree

```python
DT_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)

results = {}
#Training the model
start = time()
DT_classifier.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
DT_Prediction = DT_classifier.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 31: Model building for Decision Tree

5.7.2 Model Evaluation

The model is evaluated by using DT_classifier.fit() and DT_classifier.predict(). The code for accuracy, precision, recall, f1-score, cross validation accuracy is shown in Figure 32. The output confusion matrix, classification report and cross validation accuracy with cross validation time is shown in Figure 33.

```python
## Accuracy Score
DT_Accuracy = accuracy_score(y_test_sm, DT_Prediction)
print("The accuracy score for Decission tree in percentage is: "+"{:.2f}".format(DT_Accuracy*100))

## Precision
DT_Precision = precision_score(y_test_sm, DT_Prediction)
print("The precision score for Decission tree is: "+"{:.2f}".format(DT_Precision))

## Recall Feature
DT_Recall = recall_score(y_test_sm, DT_Prediction)
print("The recall score for Decission tree is: "+"{:.2f}".format(DT_Recall))
## F1 Score
DT_F1Score = f1_score(y_test_sm, DT_Prediction)
print("The F1 Score for Decission tree is: "+"{:.2f}".format(DT_F1Score))

## Confusion Matrix
DT_Confusion_Matrix=confusion_matrix(y_test_sm,DT_Prediction)
print("Confusion_Matrix: \n\n",DT_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,DT_Prediction,zero_division=1,target_names=target_names))
## Cross Validation
#for K=10
DT_accuracies = cross_val_score(estimator = DT_classifier, X = x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(DT_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(DT_accuracies.std()*100))

#for K=20
DT_accuracies = cross_val_score(estimator = DT_classifier, X = x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(DT_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(DT_accuracies.std()*100))

#for K=30
DT_accuracies = cross_val_score(estimator = DT_classifier, X = x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(DT_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(DT_accuracies.std()*100))

#for K=40
DT_accuracies = cross_val_score(estimator = DT_classifier, X = x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(DT_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(DT_accuracies.std()*100))
```

Figure 32: Model evaluation for Decision Tree

```
The accuracy score for Decission tree in percentage is: 76.67
The precision score for Decission tree is: 0.79
The recall score for Decission tree is: 0.73
The F1 Score for Decission tree is: 0.76
Confusion_Matrix:

 [[72 18]
 [24 66]]

              precision    recall  f1-score   support

     class 0       0.75      0.80      0.77        90
     class 1       0.79      0.73      0.76        90

    accuracy                           0.77       180
   macro avg       0.77      0.77      0.77       180
weighted avg       0.77      0.77      0.77       180

Cross Validation Accuracy: 83.58 %
Cross Validation Standard Deviation: 6.50 %
Cross Validation Accuracy: 84.92 %
Cross Validation Standard Deviation: 8.90 %
Cross Validation Accuracy: 85.58 %
Cross Validation Standard Deviation: 9.85 %
Cross Validation Accuracy: 84.04 %
Cross Validation Standard Deviation: 10.93 %
```

Figure 33: Model evaluation output for Decision Tree

5.8 Multi layer Perceptron

5.8.1 Model Building

MLP classifier is used for building Multi layer Perceptron with three 8,8,8 hidden layer, RELU as activation, 500=iterations and random_state=0. It is suitable for classification prediction problems where inputs are assigned a class or label. Code is shown in Figure 34.

Multi layer Perceptron (MLP)

```
MLP_classifier = MLPClassifier(hidden_layer_sizes=(8,8,8), activation= 'relu',max_iter=500, random_state=0)


results = {}
#Training the model
start = time()
MLP_classifier.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
MLP_Prediction = MLP_classifier.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 34: Model building for MLP

5.8.2 Model Evaluation

The model is evaluated by using MLP_classifier.fit() and MLP_classifier.predict(). The code for accuracy, precision, recall, f1-score, cross validation accuracy is shown in Figure 35. The output confusion matrix, classification report and cross validation accuracy with cross validation time is shown in Figure 36.

```python
## Accuracy Score
MLP_Accuracy = accuracy_score(y_test_sm, MLP_Prediction)
print("The accuracy score for MLP in percentage is as follows:"+"{:.2f}".format(MLP_Accuracy*100))

## Precision
MLP_Precision = precision_score(y_test_sm, MLP_Prediction)
print("The precision score for MLP is: "+"{:.2f}".format(MLP_Precision))
## Recall Feature
MLP_Recall = recall_score(y_test_sm, MLP_Prediction)
print("The recall score for MLP is as follows:"+"{:.2f}".format(MLP_Recall))

## F1 Score
MLP_F1Score = f1_score(y_test_sm, MLP_Prediction)
print("The F1 Score for MLP is: "+"{:.2f}".format(MLP_F1Score))

## Confusion Matrix
MLP_Confusion_Matrix=confusion_matrix(y_test_sm,MLP_Prediction)
print("Confusion_Matrix: \n\n",MLP_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,MLP_Prediction,zero_division=1,target_names=target_names))
```

```python
## Cross Validation
#for K=10
MLP_accuracies = cross_val_score(estimator = MLP_classifier, X = x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(MLP_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(MLP_accuracies.std()*100))

#for K=20
MLP_accuracies = cross_val_score(estimator = MLP_classifier, X = x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(MLP_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(MLP_accuracies.std()*100))

#for K=30
MLP_accuracies = cross_val_score(estimator = MLP_classifier, X = x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(MLP_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(MLP_accuracies.std()*100))

#for K=40
MLP_accuracies = cross_val_score(estimator = MLP_classifier, X = x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(MLP_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(MLP_accuracies.std()*100))
```

Figure 35: Model evaluation for MLP

```
*******************************************

The accuracy score for MLP in percentage is as follows:83.33
The precision score for MLP is: 0.85
The recall score for MLP is as follows:0.81
The F1 Score for MLP is: 0.83
Confusion_Matrix:

 [[77 13]
 [17 73]]

              precision    recall  f1-score   support

     class 0       0.82      0.86      0.84        90
     class 1       0.85      0.81      0.83        90

    accuracy                           0.83       180
   macro avg       0.83      0.83      0.83       180
weighted avg       0.83      0.83      0.83       180

Cross Validation Accuracy: 84.31 %
Cross Validation Standard Deviation: 4.83 %

Cross Validation Accuracy: 83.94 %
Cross Validation Standard Deviation: 7.27 %

Cross Validation Accuracy: 84.07 %
Cross Validation Standard Deviation: 8.43 %

Cross Validation Accuracy: 83.80 %
Cross Validation Standard Deviation: 10.57 %
```

Figure 36: Model evaluation output for MLP

5.9 Comparison of all baseline models based on ROC curve plot

We used a majority class no skill prediction code where 0 for_1 range is used on test data. Lables were generated as true positive rate and false negative rate and ROC comparison were made of baseline approach models. This graphical way tells us the connection between sensitivity and specificity for every possible cut-off of a data test. Code is shown in Figure 37. Output is shown in Figure 38.

Comparison of all baseline models based on ROC curve plot ¶

```
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test_sm))]
ns_auc = roc_auc_score(y_test_sm, ns_probs)
fpr_NS, tpr_NS, thresholds_NS = roc_curve(y_test_sm, ns_probs)

plt.figure(figsize=(10,10))
plt.plot(fpr_SVM, tpr_SVM, marker='.', label='SVM', color='violet')
plt.plot(fpr_DT, tpr_DT, marker='.', label='Decision Tree', color='yellow')
plt.plot(fpr_MLP, tpr_MLP, marker='.', label='MLP', color='blue')
plt.plot(fpr_RF, tpr_RF, marker='.', label='Random Forest', color='pink')
plt.plot(fpr_HRFLR, tpr_HRFLR, marker='.', label='HRFLR', color='red')
plt.plot(fpr_gb, tpr_gb, marker='.', label='GB', color='green')


# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Comparision of All Models')
# show the legend
plt.legend()
# show the plot
plt.show()
```

Figure 37: ROC curve code for all baseline models

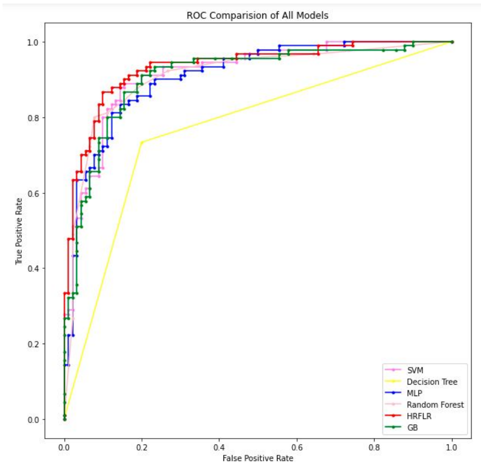


Figure 38: output of ROC comparision of all models

6 Implementation of Newly Proposed Models

Hybrid ensemble of Extreme Boosting with Random Forest(XGBRF) and CatBoost is novelty of this research project. These two models is never used for PCOS detection. Both models deals with handling the classification problem if the data is categorical in nature. The code is referred from (Bhatele and Bhadauria; 2020)(Li et al.; 2020).

6.1 Multi Hybrid ensemble of Extreme Boosting with Random Forest Perceptron

6.1.1 Model Building

First, XGBRFClassifier is imported having hyper parameters as (max_depth=3, random_state=8) then xgb_clf.fit() is used on pre-trained data. They are assigned to all independent varaibles and are fed into decision trees to predict the results. Figure 39 shows us the code for XGBRF.

Extreme Gradient Boosting and Random Forest (XGBRF)

```python
import xgboost
from xgboost import XGBRFClassifier
```

```python
# xgbrf classifier
xgb_clf = xgboost.XGBRFClassifier(max_depth=3, random_state=8)
xgb_clf.fit(x_train,y_train)
acc_xgb_clf_train = round(xgb_clf.score(x_train, y_train)*100,2)
acc_xgb_clf_test = round(xgb_clf.score(x_test,y_test)*100,2)
#cv_result.append(acc_xgb_clf_train)
print("Training Accuracy: % {}".format(acc_xgb_clf_train))
print("Testing Accuracy: % {}".format(acc_xgb_clf_test))
#Training the model
start = time()
xgb_clf.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
XGBRF_Prediction = xgb_clf.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 39: Model bulding of XGBRF

6.1.2 Model Evaluation

acc_xgb_clf_train() and acc_xgb_clf_test() is used to get the accuracy of the model. xgb_clf.fit() and xgb_clf.predict() is used for precision, recall, f1-score, cross validation accuracy is shown in Figure 40. The output confusion matrix, classification report and cross validation accuracy with cross validation time is shown in Figure 41.

```python
## Precision
XGBRF_Precision = precision_score(y_test_sm, XGBRF_Prediction)
print("The precision score for XGBRF Classifier is: "+"{:.2f}".format(XGBRF_Precision))

## Recall Feature
XGBRF_Recall = recall_score(y_test_sm, XGBRF_Prediction)
print("The recall score for XGBRF Classifier is: "+"{:.2f}".format(XGBRF_Recall))

## F1 Score
XGBRF_F1Score = f1_score(y_test_sm, XGBRF_Prediction)
print("The F1 Score for XGBRF Classifier is: "+"{:.2f}".format(XGBRF_F1Score))

## Confusion Matrix
XGBRF_Confusion_Matrix=confusion_matrix(y_test_sm,XGBRF_Prediction)
print("Confusion_Matrix: \n\n",XGBRF_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,XGBRF_Prediction,zero_division=1,target_names=target_names))
## Cross Validation
#for K=10
XGBRF_accuracies = cross_val_score(estimator = xgb_clf, X= x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(XGBRF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(XGBRF_accuracies.std()*100))

#for K=20
XGBRF_accuracies = cross_val_score(estimator = xgb_clf, X= x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(XGBRF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(XGBRF_accuracies.std()*100))

#for K=30
XGBRF_accuracies = cross_val_score(estimator = xgb_clf, X= x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(XGBRF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(XGBRF_accuracies.std()*100))

#for K=40
XGBRF_accuracies = cross_val_score(estimator = xgb_clf, X= x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(XGBRF_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(XGBRF_accuracies.std()*100))
```

Figure 40: Model Evaluation of XGBRF

```
Training Accuracy: % 89.63
Testing Accuracy: % 87.5

The precision score for XGBRF Classifier is: 0.89
The recall score for XGBRF Classifier is: 0.82
The F1 Score for XGBRF Classifier is: 0.86
Confusion_Matrix:

 [[81  9]
 [16 74]]

              precision    recall  f1-score   support

     class 0       0.84      0.90      0.87        90
     class 1       0.89      0.82      0.86        90

    accuracy                           0.86       180
   macro avg       0.86      0.86      0.86       180
weighted avg       0.86      0.86      0.86       180

Cross Validation Accuracy: 83.95 %
Cross Validation Standard Deviation: 4.80 %

Cross Validation Accuracy: 83.95 %
Cross Validation Standard Deviation: 7.26 %

Cross Validation Accuracy: 84.12 %
Cross Validation Standard Deviation: 8.07 %

Cross Validation Accuracy: 84.20 %
Cross Validation Standard Deviation: 10.03 %
```

Figure 41: Model Evaluation output of XGBRF

6.2 CatBoost

6.2.1 Model Building

After importing CatBoostClassifier with np.set_printoptions(precision=4) having hyper parameters as (iterations=199, learning_rate=0.1). cat_clf.fit() is used on pre-trained data. CatBoost is very effective algorithm of handling categorical features. It is fast and easy to use. Code is shown in Figure 42.

CatBoost

```python
import os
import pandas as pd
import numpy as np
np.set_printoptions(precision=4)

import catboost
print(catboost.__version__)
from catboost import CatBoostClassifier
```

```
0.26
```

```python
#CatBoost Classifier
cat_clf = CatBoostClassifier(iterations=199,
    learning_rate=0.1,)
cat_clf.fit(x_train,y_train)
acc_cat_clf_train = round(cat_clf.score(x_train, y_train)*100,2)
acc_cat_clf_test = round(cat_clf.score(x_test,y_test)*100,2)
#cv_result.append(acc_cat_clf_train)
print("Training Accuracy: % {}".format(acc_cat_clf_train))
print("Testing Accuracy: % {}".format(acc_cat_clf_test))
#Training the model
start = time()
cat_clf.fit(x_train_sm, y_train_sm)
end = time()
results['training_time'] = end - start

#Testing the model
start = time()
cat_Prediction = cat_clf.predict(x_test_sm)
end = time()
results['testing_time'] = end - start
```

Figure 42: Model Building of CatBoost

6.2.2 Model Evaluation

acc_cat_clf_train() and acc_cat_clf_test() is used to get the accuracy of the model. cat_clf.fit() and cat_clf.predict() is used for precision, recall, f1-score, cross validation accuracy is shown in Figure 43. The output confusion matrix, classification report and cross validation accuracy with cross validation time is shown in Figure 44.

```python
## Precision
cat_Precision = precision_score(y_test_sm, cat_Prediction)
print("The precision score for CatBoostClassifier is: "+"{:.2f}".format(cat_Precision))

## Recall Feature
cat_Recall = recall_score(y_test_sm, cat_Prediction)
print("The recall score for CatBoostClassifier is: "+"{:.2f}".format(cat_Recall))

## F1 Score
cat_F1Score = f1_score(y_test_sm, cat_Prediction)
print("The F1 Score for CatBoostClassifier is: "+"{:.2f}".format(cat_F1Score))

## Confusion Matrix
cat_Confusion_Matrix=confusion_matrix(y_test_sm,cat_Prediction)
print("Confusion_Matrix: \n\n",cat_Confusion_Matrix, "\n" )

## Classification Report
target_names =['class 0', 'class 1']
print(classification_report(y_test_sm,cat_Prediction,zero_division=1,target_names=target_names))

## Cross Validation
#for K=10
cat_accuracies = cross_val_score(estimator = cat_clf, X= x_train_sm, y = y_train_sm, cv = 10)
print("Cross Validation Accuracy: {:.2f} %".format(cat_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(cat_accuracies.std()*100))

#for K=20
cat_accuracies = cross_val_score(estimator = cat_clf, X= x_train_sm, y = y_train_sm, cv = 20)
print("Cross Validation Accuracy: {:.2f} %".format(cat_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(cat_accuracies.std()*100))

#for K=30
cat_accuracies = cross_val_score(estimator = cat_clf, X= x_train_sm, y = y_train_sm, cv = 30)
print("Cross Validation Accuracy: {:.2f} %".format(cat_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(cat_accuracies.std()*100))

#for K=40
cat_accuracies = cross_val_score(estimator = cat_clf, X= x_train_sm, y = y_train_sm, cv = 40)
print("Cross Validation Accuracy: {:.2f} %".format(cat_accuracies.mean()*100))
print("Cross Validation Standard Deviation: {:.2f} %".format(cat_accuracies.std()*100))
```

Figure 43: Model Evaluation of CatBoost

```
Training Accuracy: % 95.31
Testing Accuracy: % 86.03

The precision score for CatBoostClassifier is: 0.89
The recall score for CatBoostClassifier is: 0.81
The F1 Score for CatBoostClassifier is: 0.85
Confusion_Matrix:

 [[81  9]
 [17 73]]

              precision    recall  f1-score   support

     class 0       0.83      0.90      0.86        90
     class 1       0.89      0.81      0.85        90

    accuracy                           0.86       180
   macro avg       0.86      0.86      0.86       180
weighted avg       0.86      0.86      0.86       180

Cross Validation Accuracy: 89.06 %
Cross Validation Standard Deviation: 4.66 %
Cross Validation Accuracy: 88.55 %
Cross Validation Standard Deviation: 6.97 %
Cross Validation Accuracy: 89.43 %
Cross Validation Standard Deviation: 7.18 %
Cross Validation Accuracy: 89.15 %
Cross Validation Standard Deviation: 8.00 %
```

Figure 44: Model Evaluation output of CatBoost

The scripts and functions mentioned above are all provided in the ICT solution along with this project.

References

Bhatele, K. R. and Bhadauria, S. S. (2020). Glioma segmentation and classification system based on proposed texture features extraction method and hybrid ensemble learning, *2017 2nd International Conference for Convergence in Technology (I2CT)*, Vol. 37, pp. 989–1001.

Li, Y., Mai, Y., Lin, Z. and Liang, S. (2020). Online transaction detection method using catboost model, *2020 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pp. 236–240.