Configuration Manual

MSc Research Project
Data Analytics

Jayanta Behera
Student ID: x18188834

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes, Dr. Pramod Pathak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jayanta Behera
Student ID:	x18188834
Programme:	Data Analytics **Year:** 2020
Module:	MSc Research Project
Lecturer:	Dr. Paul Stynes, Dr. Pramod Pathak
Submission Due Date:	17th August 2020
Project Title:	Configuration Manual
Word Count:	1612
Page Count	11

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:	Jayanta Behera
Date:	17th August 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

Jayanta Behera
x18188834
MSc Research Project in Data Analytics
17th August 2020

1 Introduction

This configuration manual specifies the details of computer hardware as well as software that are required along with the programming phases to implement of the below research project in detail:
"Flood Severity Classification using Machine Learning"

2 System Configuration

2.1 Hardware

- Processor: Intel Core i5-8th Gen
- RAM: 16 GB
- System Type: Windows OS, 64-bit
- Storage: 512 GB SSD

2.2 Software

Jupyter Notebook (Anaconda) : Anaconda is an open source software available in official website of anaconda platform where machine learning models are run. Data merging, pre-processing, visualisation etc. are done in Python language (version 3.7.3) using Jupyter notebook.

RStudio : RStudio is a programming platform used to run machine learning models. In the current project, data imputation and feature selection are performed using R programming in the RStudio Desktop version.

Microsoft Excel 2016 : MS Excel is a spreadsheet used to store the dataset downloaded from various websites in form of comma separated files. This is also used to plot a few visualisations to show model accuracy.

Tableau : Tableau is a visualisation software used to create histograms and bar plots. In this project, this is done to show a few evaluation plots.

UiPath : It is a robotic automation platform used to extract the data from various websites automatically. In this project, UiPath is used to extract the weather data via web scrapping.

3 Project Development

The project was implemented using python and R programming. Initial stage of the project deals with extracting climatic and topographic data and merging with the historical flood data. This was followed by data clean-up, feature engineering, feature selection and implementing various machine learning techniques using python programming available under keras and sklearn library.

3.1 Data Preparation

Initially flood archive data was downloaded as csv file from official website of Colorado[1] and imported via python programming. With the geographical coordinates and the flood date, URLs were created in python Beautifulsoup library to extract the weather data via web scrapping as shown in Figure 1 and saved in the same excel dataset as new column-

```
def getdetails(URL,began_date):
    dt_object = began_date.date()
    dt_object1 = dt_object-timedelta(days=2) # this is for day-2's values
    dt_object = dt_object1


    req = Request(URL, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(req).read()
    soup = BeautifulSoup(webpage, 'html.parser')
    value1 = str(soup.find_all("div", {"class":"station-name ng-star-inserted"}))
    result_list = re.sub(r"^.+?(?=history)", "", value1)

    split_string = result_list.split("date", 1)
    substring = split_string[0]

    hist_url = 'https://www.wunderground.com/'+substring+'date/'+str(dt_object)+'/'
    return hist_url
```

```
url= "https://www.wunderground.com/weather/"
for i in range(len(df)):
    # adding this line as internet got interrupted inbetween
    if i > 4163:
        new_url = url + str(df.loc[i, "lat"]) + ',' + str(df.loc[i, "long"]) +'/'
        df.loc[i,"coordinate url"]=new_url
        try :
            val= getdetails(new_url,df.loc[i, "Began"])
            df.loc[i,"began date url"]=val
```

Figure 1: Weather URL Extraction - Web Scrapping

3.1.1. Weather Data via Web Scraping

A sequence was created in UiPath to extract the weather data from the official website of weatherunderground[2] via web scrapping as shown in Figure 2.

[1] http://floodobservatory.colorado.edu/Archives/
[2] https://www.wunderground.com/

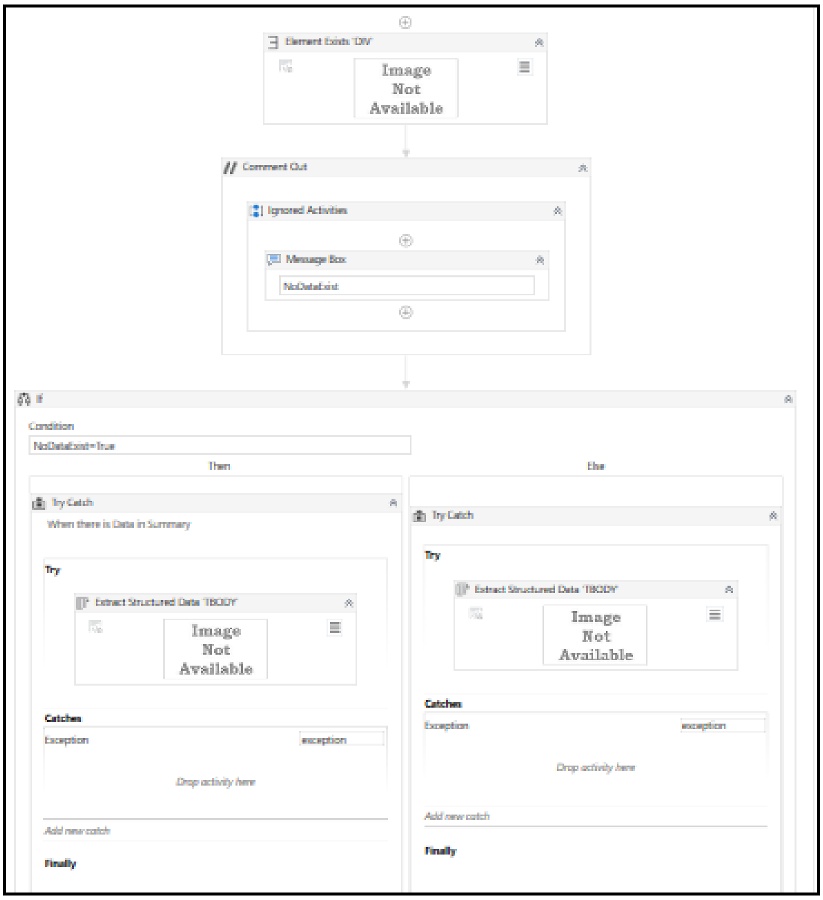

Figure 2: UiPath Sequence Architecture

3.1.2. Weather Data via Application Program Interface

As the web scrapping took prolonged hours to run, API for the same website was used to extract the climatic details using python programming shown in Figure 3-

```
URL = df1.loc[i,"began date url"]

start_date = df1.loc[i,"Began"]
start_date = ((start_date).date())
start_date=start_date-timedelta(days=1)# 1 days before
start_date=str(start_date)
start_date = start_date.replace('-', '')
print("URL :",URL)
try:
    print("i :",i)
    req = Request(URL, headers={'User-Agent': 'Mozilla/5.0'})
    webpage = urlopen(req).read()
    soup = BeautifulSoup(webpage, 'html.parser')
    value1 = soup.find('div', {'class':'station-name ng-star-inserted'})#this is to get the city code like br/cerro-azul

    if value1 is None:
        j=i
    else:
        value2=value1.find('span',{'class':"station-id"})
        value3 = (value2.contents[0][1:-1])
        result_list = re.sub(r"^.+?(?=daily)", "", URL) #get the new values to add to the new_url
        string_in_slashes = result_list.split('/')[1].strip() # to get the country code from URL
        string_in_slashes.upper()
        new_url = 'https://api.weather.com/v1/location/' +value3+':9:'+string_in_slashes.upper()+'/observations/historic
        try:
            |
            res=requests.get(new_url)
            print("opened new url res:",res)
            w_u = res.json()
            df1.loc[i,"Time"] = w_u['observations'][0]['expire_time_gmt']

            df1.loc[i,"Temperature"] = w_u['observations'][0]['temp']
            df1.loc[i,"Dew Point"] = w_u['observations'][0]['dewPt']
```

Figure 3: API code– weatherunderground

However, data was not available for each of the dates. Hence 2 other weather APIs were used[3][4] and the data was extracted using python as showing in Figure 4 and 5-

```
def retrieve_hist_data(api_key, location_list, start_date):

    for location in location_list:

        start_d= str(start_date)
        end_d = str(start_date)
        frequency=1
        url_page = 'http://api.worldweatheronline.com/premium/v1/past-weather.ashx?key=' + api_key + '&q=' + location + '&fo
                frequency)
        print("url_page :",url_page)
        json_page = urllib.request.urlopen(url_page, timeout=10)

        json_data = json.loads(json_page.read().decode())


        print("data collection for row :",i)
        data = json_data['data']['weather'][0]['date']
        data1 = json_data['data']['weather'][0]['avgtempF']
        data2 = json_data['data']['weather'][0]['hourly'][0]["DewPointF"]
        data3 = json_data['data']['weather'][0]['hourly'][0]["humidity"]
        data4 = json_data['data']['weather'][0]['hourly'][0]["winddir16Point"]
        data5 = json_data['data']['weather'][0]['hourly'][0]["windspeedMiles"]
        data6 = json_data['data']['weather'][0]['hourly'][0]["WindGustMiles"]
        data7 = json_data['data']['weather'][0]['hourly'][0]["pressureInches"]
        data8 = json_data['data']['weather'][0]['hourly'][0]["precipInches"]
```

```
for i, row in df2.iterrows():
    #internet went off, so retrying afte 4546th row
    if i >= 4546:
        start_date = df2.loc[i,"Began"]
        start_date = ((start_date).date())
        start_date=start_date-timedelta(days=1)#1 days before
        #api_key = '365f0b99b7f2410fbb8111748201107'
        #api_key = 'dfd495982ade45cfb30183830201507'
        api_key = '064d3cdae3804e168e501356201107'
        location_list = [str(df2.loc[i,"lat"])+','+str(df2.loc[i,"long"])]
        print("row number :",i)
        try :
            data, data1, data2, data3, data4, data5, data6, data7, data8, data9 = retrieve_hist_data(api_key,location_lis
            df2.loc[i,"Time"]=data
            df2.loc[i,"Temperature"]=data1
            df2.loc[i,"Dew Point"]=data2
            df2.loc[i,"Humidity"]=data3
            df2.loc[i,"Wind"]=data4
            df2.loc[i,"Wind Speed"]=data5
            df2.loc[i,"Wind Gust"]=data6
            df2.loc[i,"Pressure"]=data7
            df2.loc[i,"Precipitation"]=data8
            df2.loc[i,"Condition"]=data9


        except KeyError:
            continue
```

Figure 4: API code– worldweather

```
def weather_data(query):
    res=requests.get('http://api.openweathermap.org/data/2.5/weather?'+query+
                     '&APPID=b35975e18dc93725acb092f7272cc6b8&units=metric');
    return res.json();

########change the date before running
import time
import datetime
import re

for i, row in all_null_df.iterrows():
    api_key = '365f0b99b7f2410fbb8111748201107'
    lat = str(all_null_df.loc[i,"lat"])
    long = str(all_null_df.loc[i,"long"])
    start = (all_null_df.loc[i,"Began"]).date()
    start=start-timedelta(days=1)### 1 days before
    newformat = start.strftime('%d-%m-%Y')
    dt = str(int(time.mktime(datetime.datetime.strptime(newformat , "%d-%m-%Y").timetuple())))
    try:
        query='&lat='+lat+'&lon='+long+'&type=hour&start='+dt
        w_data=weather_data(query)
        data = w_data['dt']
        all_null_df.loc[i,"Time"]=data
        data1 = float(w_data['main']['temp'])
        data1=round((data1 * 9/5) + 32,2)
        all_null_df.loc[i,"Temperature"]=data1
        all_null_df.loc[i,"Humidity"] = w_data['main']['humidity']
        all_null_df.loc[i,"Wind Speed"] = w_data['wind']['speed']
        data7 = float(w_data['main']['pressure'])
        data7 = round(data7 * (0.02953/1.000000573),2) # conversion of milibars into inHg
        all_null_df.loc[i,"Pressure"] = data7
        all_null_df.loc[i,"Condition"] = w_data['weather'][0]['main']
```

3 https://www.worldweatheronline.com/

4 https://openweathermap.org/history

Figure 5: API code– openweatherdata

3.1.3. Topographic Data via Application Program Interface

The topographic data was extracted from official maps website[5] using python as shown in Figure 6.

```
longitude = df.loc[i,"long"]
longitude = str(longitude)
lattitude = df.loc[i,"lat"]
lattitude = str(lattitude)

api_key = 'LbcFldJNXW3OYpDkdeJMAc5xOrnnIJZDbfyabPghgHmFJkzKiTLVYIYuug91Ctw8'

location_list = lattitude+','+longitude
try :
    url_page = 'https://api.jawg.io/elevations?locations=' + location_list + '&access-token=' + api_key
    print("url :", url_page)
    json_page = urllib.request.urlopen(url_page, timeout=10)
    json_data = json.loads(json_page.read().decode())
    df.loc[i,"Elevation"] = round(json_data[0]['elevation'],2)
```

Figure 6: API code– Topographic Details

3.1.4. Data Merging

All the API data were merged in python using merge function in pandas dataframes as shown Figure 7-

```
#merging datasets
df_inner1 = pd.merge(df1, df2, on='ID', how='inner')
```

```
#merging datasets
df_inner2 = pd.merge(df3, df4, on='ID', how='inner')
```

```
#merging datasets
df_inner = pd.merge(df_inner1, df_inner2, on='ID', how='inner')
```

```
df = pd.merge(df0, df_inner, on='ID', how='outer')
```

Figure 7: Data Merge

3.2 Missing Value Imputation

The missing values in the final pandas dataframe was checked using isnull function of pandas library in python as shown in Figure 8

```
df.apply(lambda x: sum(x.isnull()))
```

```
Day.3_Dew_Point          0
Day.4_Dew_Point          0
Day0_Wind             2349
Day.1_Wind            2456
Day.2_Wind            2407
Day.3_Wind            2423
Day.4_Wind            2404
Day0_Humidity            0
```

Figure 8: Missing Value Detection

5 https://www.maps.ie/coordinates.html

These missing values were imputed in MICE package available in RStudio as shown in Figure 9.

```r
library(mice)
md.pattern(data)
md.pairs(data)

##install.packages("VIM")
##library(VIM)
##mice_plot <- aggr(data, col=c('navyblue','yellow'),
##                    numbers=TRUE, sortVars=TRUE,
##                    labels=names(data), cex.axis=.7,
##                    gap=3, ylab=c("Missing data","Pattern"))

library(ggplot2)
library(ggpubr)
##marginplot(data[,c('Area_Affected_per_day','Day.1_Precipitation')])
####impute with 3 iterations with random forest
imputed_Data <- mice(data, m=3, maxit = 3, method = 'rf', seed = 123)
summary(imputed_Data)
imputed_Data$imp$Day0_Temperature
typeof(imputed_Data$imp$Day.1_Temperature)
```

Figure 9: Missing Value Imputation

3.3 Feature Engineering

Before implementing machine learning algorithms, feature engineering was done to improve the model performance. Various steps were performed such as feature selection, one hot encoding, standardization, dimensionality reduction, class imbalance.

3.3.1. One-Hot Encoding

One hot encoding was done to convert the categorical variables to binary values in python using get_dummies function in pandas library as shown in Figure 10-

```python
pd.get_dummies(df_pca, columns=['MainCause','Day0_Condition','Day.1_Condition','Day.2_Condition','Day.3_Condition'
```

Figure 10: One-Hot Encoding Code

3.3.2. Standardisation

Standardisation was done to get all the columns under same scale to avoid the impact of higher valued columns. This was done using StandardScaler function in preprocessing library of sk learn using python as shown in Figure 11

```python
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

Figure 11: Data Standardization Code

3.3.3. Dimensionality Reduction

As more input dimension requires more processing time and storage space, dimensionality reduction techniques were applied to reduce the input dimensions to 2 or 3 components explaining most of the variances in the dataset. This was done using PCA function available under sklear.decomposition library in python shown in Figure 12-

```python
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, df_dm[['Flood_Risk']]], axis = 1)
```

Figure 12: Dimensionality Reduction using PCA

As the PCA components didn't explain the variance of dataset, TSNE function available under sklearn.manifold library was used in python as shown in Figure 13-

```
from sklearn.manifold import TSNE
X_embedded = TSNE(n_components=2, perplexity=50.0, n_iter=1000).fit_transform(x)
tSNEDf = pd.DataFrame(data = X_embedded
                , columns = ['tSNE component 1', 'tSNE component 2'])
tDf = pd.concat([tSNEDf, df_dm[['Flood_Risk']]], axis = 1)
tDf.head(5)
```

Figure 13: Dimensionality Reduction using TSNE

SVD, ICA and Isomap techniques were also used as shown in the Figures 14-165-

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, random_state=42).fit_transform(x)
plt.figure(figsize=(12,8))
plt.title('SVD Components')
plt.scatter(svd[:,0], svd[:,1])
plt.scatter(svd[:,1], svd[:,0])
```

Figure 14: Dimensionality reduction using SVD

```
from sklearn.decomposition import FastICA
ICA = FastICA(n_components=2, random_state=12)
a=ICA.fit_transform(x)
plt.figure(figsize=(12,8))
plt.title('ICA Components')
plt.scatter(a[:,0], a[:,1])
plt.scatter(a[:,1], a[:,0])
```

Figure 15: Dimensionality Reduction using ICA

```
from sklearn import manifold
trans_data = manifold.Isomap(n_neighbors=5, n_components=2, n_jobs=-1).fit_transform(x)
plt.figure(figsize=(12,8))
plt.title('Decomposition using ISOMAP')
plt.scatter(trans_data[:,0], trans_data[:,1])
plt.scatter(trans_data[:,1], trans_data[:,0])
```

Figure 16: Dimensionality Reduction using Isomap

3.3.4. Feature Elimination

Various feature elimination techniques were used as shown in Figure 17-20. Recursive feature elimination technique was run with all the columns of the dataset in the Figure 17-

```
rfe_selector = RFE(estimator=LogisticRegression(), n_features_to_select=len(df.columns), step=10, verbose=5)
rfe_selector.fit(x, y)
rfe_support = rfe_selector.get_support()
rfe_feature = df.loc[:,rfe_support].columns.tolist()
print(str(len(rfe_feature)), 'selected features')
```

Figure 17: Recursive Feature Elimination Technique

Random forest classifier was run for maximum of 121 features as its upper limit for 100 estimators as shown in Figure 18-

```
embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=100), max_features=121)
embeded_rf_selector.fit(x, y)

embeded_rf_support = embeded_rf_selector.get_support()
embeded_rf_feature = df.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features')
```

Figure 18: RandomForestClassifier Technique

LGBM classifier was run for maximum of 121 features as its upper limit for 500 estimators as shown in Figure 19-

```
from lightgbm import LGBMClassifier

lgbc=LGBMClassifier(n_estimators=500, learning_rate=0.05, num_leaves=32, colsample_bytree=0.2,
            reg_alpha=3, reg_lambda=1, min_split_gain=0.01, min_child_weight=40)

embeded_lgb_selector = SelectFromModel(lgbc, max_features=121)
embeded_lgb_selector.fit(x, y)

embeded_lgb_support = embeded_lgb_selector.get_support()
embeded_lgb_feature = df.loc[:,embeded_lgb_support].columns.tolist()
print(str(len(embeded_lgb_feature)), 'selected features')
```

Figure 19: LGBM Classifier Technique

Boruta function under mtools library in RStudio was used as shown in Figure 20-

```
library(mltools)
traindata <- one_hot(as.data.table(traindata~.-Severity))
summary(traindata)

#####implement and check the performance of boruta package
set.seed(123)
#boruta.train <- Boruta(Area_Affected_per_day~.-ID, data = traindata, doTrace = 2)
boruta.train <- Boruta(Severity~.-ID, data = traindata, doTrace = 2)
print(boruta.train)


plot(boruta.train, xlab = "", xaxt = "n")
lz<-lapply(1:ncol(boruta.train$ImpHistory),function(i)
  boruta.train$ImpHistory[is.finite(boruta.train$ImpHistory[,i]),i])
names(lz) <- colnames(boruta.train$ImpHistory)
Labels <- sort(sapply(lz,median))
axis(side = 1,las=2,labels = names(Labels),
       at = 1:ncol(boruta.train$ImpHistory), cex.axis = 0.7)
```

Figure 20: Boruta feature elimination Technique

3.3.5. Class Imbalance

As the output variable was categorical with imbalanced class distribution, SMOTE analysis was done to balance the output class as shown in Figure 21-

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=2)
X_T,y_T = sm.fit_sample(X, y)
```

Figure 21: Code for SMOTE

3.4 Modelling

All the machine learning models were run in python using sklearn library as shown in Figure 22-31.

3.3.1. Data Split

Before applying the models, data was split using train_test_split function in the ratio 80:20 as shown in Figure 22-

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Figure 22: Data Split Code

3.3.2. kNeighborsClassifier

k Nearest Neighbour Classifier was the first model applied with k value as 6. This is obtained from the results of gridsearch function to get the best parameter. The model is shown in Figure 23-

```
#Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=6)
#Fit the model
knn.fit(X_train,y_train)
#Get accuracy. Note: In case of classification algorithms score method represents accuracy
knn.score(X_test,y_test)
```

Figure 23: Code for kNN

3.3.3. SVC

Support Vector Machine was used with parameter value of rbf for kernel, 0.1 for gamma and 10 for C as shown in Figure 24 with rest other parameters as default value-

```
from sklearn.svm import SVC

#svc = SVC(kernel='rbf', C = 1, gamma='scale')
svc = SVC(kernel='rbf', C = 10, gamma=0.1)
svc.fit(X_train,y_train)

pred_y_svc = svc.predict(X_test)
```

Figure 24: Code for SVC

3.3.4. Decision Tree

Decision Tree Classifier was applied with criterion value as entropy and max_depth as 29 shown in Figure 25 and rest other parameters with default values-

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=29)
clf = clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Figure 25: Code for DecisionTreeClassifier

3.3.5. Random Forest

Random Forest Classifier was applied with criterion value 1000 for n_estimators and 100 as random_state as shown in Figure 26 and rest other parameters with default values-

```
rf = RandomForestClassifier(n_estimators = 1000, random_state = 100)
rf=rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Figure 26: Code for RandomForestClassifier

3.3.6. Bagging

Bagging Classifier was applied as shown in Figure 27 with all parameters as default values except random_state as 1-

```
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
model = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1))
a=model.fit(X_train,y_train)
model.score(X_test,y_test)
```

Figure 27: Code for BaggingClassifier

3.3.7. AdaBoost

AdaBoost Classifier was applied as shown in figure 27 with all parameters as default values except random_state as 1-

```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=1)
a=model.fit(X_train,y_train)
model.score(X_test,y_test)
```

Figure 27. Code for AdaBoostClassifier

3.3.8. Gradient Boost

Gradient Boosting Classifier was applied as shown in Figure 28 with all parameters as default values except learning rate as 0.01 and random_state as 1-

```
from sklearn.ensemble import GradientBoostingClassifier
model= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
a= model.fit(X_train,y_train)
model.score(X_test,y_test)
```

Figure 28: Code for GradientBoostingClassifier

3.3.9. XGBoost

VGBoost Classifier was applied as shown in Figure 29 with all parameters as default values except random_state as 1 and learning_rate as 0.01-

```
import xgboost as xgb
model=xgb.XGBClassifier(random_state=1,learning_rate=0.01)
a= model.fit(X_train,y_train)
model.score(X_test,y_test)
```

Figure 29: Code for XGBoostClassifier

3.3.10. Neural Network

Figure 30 shows the architecture of the artificial neural networks with 29 input parameters selected after different combinations to increase the model efficiency-

```
model = Sequential()
#adding layer
model.add(Dense(units =15,input_dim =29,activation='relu')) # hidden layer
model.add(Dense(units=7,activation='relu'))
#model.add(Dense(units=10,activation='relu'))
model.add(Dense(units=3,activation='sigmoid')) #output layer

#compiling model

model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
history = model.fit(x_train,y_train,batch_size=64,epochs=500, validation_data = (x_test, y_test), verbose = 1)
```

Figure 30: Code for Artificial Neural Network

3.3.11. Cross Validation for Neural Network

In order to avoid overfitting, cross validation technique was applied to neural network with the codes shown in Figure 31-

```python
def my_model():
    model = Sequential()
    #adding layer
    model.add(Dense(units =15,input_dim =29,activation='relu')) # hidden layer
    model.add(Dense(units=7,activation='relu'))
    #model.add(Dense(units=10,activation='relu'))
    model.add(Dense(units=3,activation='sigmoid')) #output layer
    #compiling model

    model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
    return model
```

```python
from keras.wrappers.scikit_learn import KerasClassifier

classifier = KerasClassifier(build_fn=my_model,epochs=200,batch_size=10,verbose=1)
```

```python
from sklearn.model_selection import cross_val_score
cv_results = cross_val_score(estimator=classifier,X=x_train,y=y_train, n_jobs=-1,verbose=1,cv=10)
```

Figure 31: Cross validation code for Artificial Neural Network