Jason Chow

a01069757

COMP 4254 – Assignment 2

Introduction

The goal of this report is to describe logistic regression models that can predict if surveyors do not have diabetes or have either prediabetes or diabetes. The models are based on a clean dataset of 70,692 survey responses to the Centre of Disease and Control from 2015. It has an equal 50-50 split of respondents with no diabetes and with either prediabetes or diabetes. The target variable Diabetes\_binary has 2 classes. 0 is for no diabetes, and 1 is for prediabetes or diabetes (prediabetes or diabetes will be referred to as diabetes in the rest of the report).

Data Exploration

Tableau is used to visualize the dataset. The following figure shows the feature variables (listed in absolute numbers and in percentages) that falls under class 0 or 1.

Table

Description automatically generated A screenshot of a text message

Description automatically generated with low confidence

This figure highlights which feature is typically highly associated with having diabetes and which feature variables are least typically associated with having diabetes. Features that have a high percentage that fall under class 1 is highly associated with having diabetes. Features that have a high percentage that fall under class 0 is least associated with having diabetes.

The following figure shows which features are typically highly associated with having diabetes.

Chart, pie chart

Description automatically generated

This figure shows visually which features are found in the highest percentage in diabetics compared to non-diabetics. The lay person would commonly associate these features to diabetes.

The following figure shows which features are typically least associated with having diabetes.

Chart, pie chart

Description automatically generated

This figure shows visually which features are found in the least percentages in diabetics compared to non-diabetics. The lay person would not commonly associate these features to diabetes. The only surprising result is for heavy alcohol use. Most people would not associate heavy alcohol use to lower chances of having diabetes.

Identifying Significant Features

The significant features are shown in the pie charts previously shown. These features are described in detail below:

HeartDiseaseorAttack - coronary heart disease (CHD) or myocardial infarction (MI). 0 = no. 1 = yes.

Stroke - (Ever told) you had a stroke. 0 = no. 1 = yes.

DiffWalk - Do you have serious difficulty walking or climbing stairs? 0 = no. 1 = yes.

PhysHlth – How many physical illness or injury days have you experienced in the past 30 days? The results are 1-30 days.

HighBP – Do you have high blood pressure? 0 = no. 1 = yes.

HighChol – Do you have high cholesterol? 0 = no. 1 = yes.

HvyAlcoholConsump – Do you consume alcohol greater or equal to 14 drinks per week (for adult men) or greater or equal to 7 drinks per week (for adult women). 0 = no. 1 = yes.

PhysActivity – Have you done physical activity in past 30 days, not including job. 0 = no. 1 = yes.

Income – Personal income from a scale of 1 to 8. 1 = less than $10,000. 5 = less than $35,000. 8 = $75,000 or more.

Fruits – Do you consume fruit more than 1 or more times per day. 0 = no. 1 = yes.

Veggies - Do you consume veggies more than 1 or more times per day. 0 = no. 1 = yes.

Education – Education level from a scale of 1 to 6. 1 = never attended school or only kindergarten. 6 = college 4 years or more (College graduate).

Imputing and Variable Creation

The dataset is very clean. The only features that were created is related to BMI. The BMI results are originally listed in absolute numbers. These results are binned according to the categorizes that the World Health Organization provides. These categorizes are Underweight < 18.5, Normal Range = 18 .5 - 24.9, Overweight = 25.0 - 29.9, Obese Class 1 >= 30.0 – 34.9, Obese Class 2 >= 35.0 - 39.9, Obese Class >= 40.

Data Modelling

Four models are created. The first, second, and third model are a logistic regression model that predicts using cross fold validation. Cross fold validation is splitting data into random test and training set over several runs to produce more accurate results. Each model is created identically except that the first model includes all features that are significant, according to the chi test, the second model list features that are typically highly associated with having diabetes, and the third model list features that are typically least associated with having diabetes. The fourth model is a logistic regression model that predicts using stacking. Stacking is the procedure of building a model with the output from multiple models. It includes all features that are significant, according to the chi test.

Model Evaluation

The accuracy, precision, recall, f1 scores, and ROC plot for all models are listed below.

Text

Description automatically generatedText

Description automatically generated

Model 1 Results Model 2 Results

Text

Description automatically generated Text

Description automatically generated

Model 3 Results Model 4 Results

Chart

Description automatically generated

Model 1 ROC

Chart

Description automatically generated

Model 2 ROC

Chart, line chart

Description automatically generated

Model 3 ROC

Chart

Description automatically generated

Model 4 ROC

The accuracy percentage shows how accurately the model predicts. The precision percentage shows how relevant are the model’s results. The recall percentage show how many of the correct prediction are found. The F-1 score reflects the model’s overall accuracy. The results from Models 1 and 4 are nearly identical and outperforms the others. This result is not surprising as the more significant factors, there are in a model, the more accurate it is. Model 2 outperforms Model 3. This result is due to Model 2’s features are more statistically significant to the dependant variable that Model 3’s features.

The scores overall for each model is above average. The model performs well as it uses very clean data and the data is balanced. Furthermore, the results for Model 1, 2, 3 performs cross fold validation with averages 8 outputs and Model 4 performs stacking, which builds a model with the output from multiple models. The more outputs that are averaged, the more accurate the results.

Similarly, the ROC curves from Model 1 and 4 are nearly identical and outperforms the others. Model 2’s ROC curve outperforms Model 3’s. ROC curves are frequently used to show in a graphical way the connection/trade-off between the true positive and false positive rates of a model using different probability thresholds. The further left the logistic regression and KNN lines are from the middle line, the more likely the model predicts correctly. The AUC score shows the model’s ability to distinguish between classes. It is a summary of the ROC curve. Each ROC curve and AUC score shows that its respective model predicts accurately except for Model 3’s. Model’s 3 logistic regression and KNN are fairly close to the middle line which shows the model only slightly predicts better than a 50/50 guess.

The advantages of using Model 1 and 4 are they are the most accurate models. The advantage of using model 2 is that only 6 variables are needed to produce a relatively reliable model. There is no advantage of using model 3.

Conclusion

The results from Model 1 and 4 are nearly identical. Just slightly, the AUC scores from the ROC curves are higher in Model 4. Furthermore, the coding necessary to create Model 4 is slightly quicker to produce for most. Therefore, Model 4 is the top model. This model can be improved if more models are averaged to produce the output.

**Appendix**

Logistic Regression Model Python Script for Model 1

import pandas as pd  
from sklearn import preprocessing  
from sklearn.linear\_model import LogisticRegression  
import numpy as np  
from sklearn import metrics  
  
  
PATH = "C:\\datasets\\" # Windows  
CSV\_DATA = "diabetes.csv"  
df = pd.read\_csv(PATH + CSV\_DATA, sep=',')  
  
# Show all columns.  
pd.set\_option('display.max\_columns', None)  
pd.set\_option('display.width', 1000)  
  
print(df.describe())  
  
print("Value counts: " + str(df['Diabetes\_binary'].value\_counts()))  
le = preprocessing.LabelEncoder()  
  
df['y'] = le.fit\_transform(df['Diabetes\_binary'])  
print(df.tail())  
y = df[['y']]  
  
  
predictorVariables = list(df.keys())  
predictorVariables.remove('Diabetes\_binary')  
predictorVariables.remove('y')  
  
X = df[predictorVariables]  
X = X.copy()  
  
y = df['y']  
  
  
X['bmiBin'] = pd.cut(x=X['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9, 99])  
  
tempDf = X[['bmiBin']] # Isolate columns  
dummyDf = pd.get\_dummies(tempDf, columns=['bmiBin'])  
X = pd.concat(([X, dummyDf]), axis=1) # Join dummy df with original  
del X['bmiBin']  
  
X.rename(columns={'bmiBin\_(0.0, 18.49]':'BMI - Underweight','bmiBin\_(18.49, 24.9]':'BMI - Normal', 'bmiBin\_(24.9, 29.9]':'BMI - Overweight', 'bmiBin\_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin\_(34.9, 39.9]':'BMI - Obese Class 2', 'bmiBin\_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)  
  
print(X.head())  
  
  
from sklearn.preprocessing import MinMaxScaler  
sc\_x = MinMaxScaler()  
X\_Scale = sc\_x.fit\_transform(X)  
  
  
from sklearn.feature\_selection import SelectKBest  
from sklearn.feature\_selection import chi2  
  
test = SelectKBest(score\_func=chi2, k=26)  
chiScores = test.fit(X, y) # Summarize scores  
np.set\_printoptions(precision=3)  
  
print("\nPredictor variables: " + str(list(X.keys())))  
print("Predictor Chi-Square Scores: " + str(chiScores.scores\_))  
  
  
cols = chiScores.get\_support(indices=True)  
print("\nSignificant columns after chi-square test")  
print(cols)  
features = X.columns[cols]  
print("Significant column names after chi-square test")  
print(np.array(features))  
  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
  
X = X[['HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'Stroke',  
 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',  
 'HvyAlcoholConsump', 'NoDocbcCost', 'GenHlth', 'MentHlth',  
 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income',  
 'BMI - Underweight', 'BMI - Normal', 'BMI - Overweight',  
 'BMI - Obese Class 1', 'BMI - Obese Class 2', 'BMI - Obese Class 3']]  
  
X\_Scale = sc\_x.fit\_transform(X)  
  
# Split data.  
X\_train,X\_test,y\_train,y\_test = train\_test\_split(X\_Scale, y, test\_size=0.25,  
 random\_state=0)  
  
# Build logistic regression model and make predictions.  
logisticModel = LogisticRegression(fit\_intercept=True, solver='liblinear',  
 random\_state=0)  
logisticModel.fit(X\_train,y\_train)  
y\_pred=logisticModel.predict(X\_test)  
print("\nPredictions from logistic model")  
print(y\_pred)  
  
  
# enumerate splits - returns train and test arrays of indexes.  
# scikit-learn k-fold cross-validation  
from sklearn.model\_selection import KFold  
#kfold = KFold(3, True)  
kfold = KFold(n\_splits=8, shuffle=True)  
count = 0  
  
accuracyList = []  
precisionList = []  
recallList = []  
f1List = []  
  
for train\_index, test\_index in kfold.split(X\_Scale):  
  
 X\_train, X\_test = X\_Scale[train\_index], X\_Scale[test\_index]  
 y\_train, y\_test = y[train\_index], y[test\_index]  
  
 # Perform logistic regression.  
 logisticModel = LogisticRegression(fit\_intercept=True,  
 solver='liblinear')  
 # Fit the model.  
 logisticModel.fit(X\_train, np.ravel(y\_train))  
  
 y\_pred = logisticModel.predict(X\_test)  
 y\_prob = logisticModel.predict\_proba(X\_test)  
  
 # Show confusion matrix and accuracy scores.  
 cm = pd.crosstab(y\_test, y\_pred,  
 rownames=['Actual'],  
 colnames=['Predicted'])  
 count += 1  
 print("\n\*\*\*K-fold: " + str(count))  
  
 # Calculate accuracy and precision scores and add to the list.  
 accuracy = metrics.accuracy\_score(y\_test, y\_pred)  
 precision = metrics.precision\_score(y\_test, y\_pred)  
 recall = metrics.recall\_score(y\_test, y\_pred)  
 f1 = metrics.f1\_score(y\_test, y\_pred)  
  
 print('\nAccuracy: ', accuracy)  
 accuracyList.append(accuracy)  
 print("\nPrecision: ", precision)  
 precisionList.append(precision)  
 print("\nRecall: ", recall)  
 recallList.append(recall)  
 print("\nF1: ", f1)  
 f1List.append(f1)  
 print("\nConfusion Matrix")  
 print(cm)  
  
print("\nAccuracy and Standard Deviation For All Folds:")  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
print("Average Accuracy: ")  
print(np.mean(accuracyList))  
print("Accuracy SD: ")  
print(np.std(accuracyList))  
print("Average Precision: ")  
print(np.mean(precisionList))  
print("Precision SD: ")  
print(np.std(precisionList))  
print("Average Recall: ")  
print(np.mean(recallList))  
print("Recall SD: ")  
print(np.std(recallList))  
print("Average F1: ")  
print(np.mean(f1List))  
print("F1 SD: ")  
print(np.std(f1List))  
  
  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
  
  
# split into train-test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=27)  
  
# train models  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
  
# logistic regression  
model1 = LogisticRegression()  
# knn  
model2 = KNeighborsClassifier(n\_neighbors=4)  
  
# fit model  
model1.fit(X\_train, y\_train)  
model2.fit(X\_train, y\_train)  
  
# predict probabilities  
pred\_prob1 = model1.predict\_proba(X\_test)  
pred\_prob2 = model2.predict\_proba(X\_test)  
  
from sklearn.metrics import roc\_curve  
  
# roc curve for models  
fpr1, tpr1, thresh1 = roc\_curve(y\_test, pred\_prob1[:,1], pos\_label=1)  
fpr2, tpr2, thresh2 = roc\_curve(y\_test, pred\_prob2[:,1], pos\_label=1)  
  
# roc curve for tpr = fpr  
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr, p\_tpr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
# Compute AUC score.  
from sklearn.metrics import roc\_auc\_score  
  
# auc scores  
auc\_score1 = roc\_auc\_score(y\_test, pred\_prob1[:,1])  
auc\_score2 = roc\_auc\_score(y\_test, pred\_prob2[:,1])  
  
print("AUC scores")  
print(auc\_score1, auc\_score2)  
  
# matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')  
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')  
plt.plot(p\_fpr, p\_tpr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
  
plt.legend(loc='best')  
plt.savefig('ROC',dpi=300)  
plt.show();

Logistic Regression Model Python Script for Model 2

import pandas as pd  
from sklearn import preprocessing  
from sklearn.linear\_model import LogisticRegression  
import numpy as np  
from sklearn import metrics  
  
  
PATH = "C:\\datasets\\" # Windows  
CSV\_DATA = "diabetes.csv"  
df = pd.read\_csv(PATH + CSV\_DATA, sep=',')  
  
# Show all columns.  
pd.set\_option('display.max\_columns', None)  
pd.set\_option('display.width', 1000)  
  
print(df.describe())  
  
print("Value counts: " + str(df['Diabetes\_binary'].value\_counts()))  
le = preprocessing.LabelEncoder()  
  
df['y'] = le.fit\_transform(df['Diabetes\_binary'])  
print(df.tail())  
y = df[['y']]  
  
  
predictorVariables = list(df.keys())  
predictorVariables.remove('Diabetes\_binary')  
predictorVariables.remove('y')  
  
X = df[predictorVariables]  
X = X.copy()  
  
y = df['y']  
  
  
X['bmiBin'] = pd.cut(x=X['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9, 99])  
  
tempDf = X[['bmiBin']] # Isolate columns  
dummyDf = pd.get\_dummies(tempDf, columns=['bmiBin'])  
X = pd.concat(([X, dummyDf]), axis=1) # Join dummy df with original  
del X['bmiBin']  
  
X.rename(columns={'bmiBin\_(0.0, 18.49]':'BMI - Underweight','bmiBin\_(18.49, 24.9]':'BMI - Normal', 'bmiBin\_(24.9, 29.9]':'BMI - Overweight', 'bmiBin\_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin\_(34.9, 39.9]':'BMI - Obese Class 2', 'bmiBin\_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)  
  
print(X.head())  
  
  
from sklearn.preprocessing import MinMaxScaler  
sc\_x = MinMaxScaler()  
X\_Scale = sc\_x.fit\_transform(X)  
  
  
from sklearn.feature\_selection import SelectKBest  
from sklearn.feature\_selection import chi2  
  
test = SelectKBest(score\_func=chi2, k=26)  
chiScores = test.fit(X, y) # Summarize scores  
np.set\_printoptions(precision=3)  
  
print("\nPredictor variables: " + str(list(X.keys())))  
print("Predictor Chi-Square Scores: " + str(chiScores.scores\_))  
  
  
cols = chiScores.get\_support(indices=True)  
print("\nSignificant columns after chi-square test")  
print(cols)  
features = X.columns[cols]  
print("Significant column names after chi-square test")  
print(np.array(features))  
  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
  
X = X[['HighBP', 'HighChol', 'Stroke',  
 'HeartDiseaseorAttack', 'PhysHlth', 'DiffWalk']]  
  
X\_Scale = sc\_x.fit\_transform(X)  
  
# Split data.  
X\_train,X\_test,y\_train,y\_test = train\_test\_split(X\_Scale, y, test\_size=0.25,  
 random\_state=0)  
  
# Build logistic regression model and make predictions.  
logisticModel = LogisticRegression(fit\_intercept=True, solver='liblinear',  
 random\_state=0)  
logisticModel.fit(X\_train,y\_train)  
y\_pred=logisticModel.predict(X\_test)  
print("\nPredictions from logistic model")  
print(y\_pred)  
  
  
# enumerate splits - returns train and test arrays of indexes.  
# scikit-learn k-fold cross-validation  
from sklearn.model\_selection import KFold  
#kfold = KFold(3, True)  
kfold = KFold(n\_splits=8, shuffle=True)  
count = 0  
  
accuracyList = []  
precisionList = []  
recallList = []  
f1List = []  
  
for train\_index, test\_index in kfold.split(X\_Scale):  
  
 X\_train, X\_test = X\_Scale[train\_index], X\_Scale[test\_index]  
 y\_train, y\_test = y[train\_index], y[test\_index]  
  
 # Perform logistic regression.  
 logisticModel = LogisticRegression(fit\_intercept=True,  
 solver='liblinear')  
 # Fit the model.  
 logisticModel.fit(X\_train, np.ravel(y\_train))  
  
 y\_pred = logisticModel.predict(X\_test)  
 y\_prob = logisticModel.predict\_proba(X\_test)  
  
 # Show confusion matrix and accuracy scores.  
 cm = pd.crosstab(y\_test, y\_pred,  
 rownames=['Actual'],  
 colnames=['Predicted'])  
 count += 1  
 print("\n\*\*\*K-fold: " + str(count))  
  
 # Calculate accuracy and precision scores and add to the list.  
 accuracy = metrics.accuracy\_score(y\_test, y\_pred)  
 precision = metrics.precision\_score(y\_test, y\_pred)  
 recall = metrics.recall\_score(y\_test, y\_pred)  
 f1 = metrics.f1\_score(y\_test, y\_pred)  
  
 print('\nAccuracy: ', accuracy)  
 accuracyList.append(accuracy)  
 print("\nPrecision: ", precision)  
 precisionList.append(precision)  
 print("\nRecall: ", recall)  
 recallList.append(recall)  
 print("\nF1: ", f1)  
 f1List.append(f1)  
 print("\nConfusion Matrix")  
 print(cm)  
  
print("\nAccuracy and Standard Deviation For All Folds:")  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
print("Average Accuracy: ")  
print(np.mean(accuracyList))  
print("Accuracy SD: ")  
print(np.std(accuracyList))  
print("Average Precision: ")  
print(np.mean(precisionList))  
print("Precision SD: ")  
print(np.std(precisionList))  
print("Average Recall: ")  
print(np.mean(recallList))  
print("Recall SD: ")  
print(np.std(recallList))  
print("Average F1: ")  
print(np.mean(f1List))  
print("F1 SD: ")  
print(np.std(f1List))  
  
  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
  
  
# split into train-test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=27)  
  
# train models  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
  
# logistic regression  
model1 = LogisticRegression()  
# knn  
model2 = KNeighborsClassifier(n\_neighbors=4)  
  
# fit model  
model1.fit(X\_train, y\_train)  
model2.fit(X\_train, y\_train)  
  
# predict probabilities  
pred\_prob1 = model1.predict\_proba(X\_test)  
pred\_prob2 = model2.predict\_proba(X\_test)  
  
from sklearn.metrics import roc\_curve  
  
# roc curve for models  
fpr1, tpr1, thresh1 = roc\_curve(y\_test, pred\_prob1[:,1], pos\_label=1)  
fpr2, tpr2, thresh2 = roc\_curve(y\_test, pred\_prob2[:,1], pos\_label=1)  
  
# roc curve for tpr = fpr  
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr, p\_tpr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
# Compute AUC score.  
from sklearn.metrics import roc\_auc\_score  
  
# auc scores  
auc\_score1 = roc\_auc\_score(y\_test, pred\_prob1[:,1])  
auc\_score2 = roc\_auc\_score(y\_test, pred\_prob2[:,1])  
  
print("AUC scores")  
print(auc\_score1, auc\_score2)  
  
# matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')  
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')  
plt.plot(p\_fpr, p\_tpr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
  
plt.legend(loc='best')  
plt.savefig('ROC',dpi=300)  
plt.show();

Logistic Regression Model Python Script for Model 3

import pandas as pd  
from sklearn import preprocessing  
from sklearn.linear\_model import LogisticRegression  
import numpy as np  
from sklearn import metrics  
  
  
PATH = "C:\\datasets\\" # Windows  
CSV\_DATA = "diabetes.csv"  
df = pd.read\_csv(PATH + CSV\_DATA, sep=',')  
  
# Show all columns.  
pd.set\_option('display.max\_columns', None)  
pd.set\_option('display.width', 1000)  
  
print(df.describe())  
  
print("Value counts: " + str(df['Diabetes\_binary'].value\_counts()))  
le = preprocessing.LabelEncoder()  
  
df['y'] = le.fit\_transform(df['Diabetes\_binary'])  
print(df.tail())  
y = df[['y']]  
  
  
predictorVariables = list(df.keys())  
predictorVariables.remove('Diabetes\_binary')  
predictorVariables.remove('y')  
  
X = df[predictorVariables]  
X = X.copy()  
  
y = df['y']  
  
  
X['bmiBin'] = pd.cut(x=X['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9, 99])  
  
tempDf = X[['bmiBin']] # Isolate columns  
dummyDf = pd.get\_dummies(tempDf, columns=['bmiBin'])  
X = pd.concat(([X, dummyDf]), axis=1) # Join dummy df with original  
del X['bmiBin']  
  
X.rename(columns={'bmiBin\_(0.0, 18.49]':'BMI - Underweight','bmiBin\_(18.49, 24.9]':'BMI - Normal', 'bmiBin\_(24.9, 29.9]':'BMI - Overweight', 'bmiBin\_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin\_(34.9, 39.9]':'BMI - Obese Class 2', 'bmiBin\_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)  
  
print(X.head())  
  
  
from sklearn.preprocessing import MinMaxScaler  
sc\_x = MinMaxScaler()  
X\_Scale = sc\_x.fit\_transform(X)  
  
  
from sklearn.feature\_selection import SelectKBest  
from sklearn.feature\_selection import chi2  
  
test = SelectKBest(score\_func=chi2, k=26)  
chiScores = test.fit(X, y) # Summarize scores  
np.set\_printoptions(precision=3)  
  
print("\nPredictor variables: " + str(list(X.keys())))  
print("Predictor Chi-Square Scores: " + str(chiScores.scores\_))  
  
  
cols = chiScores.get\_support(indices=True)  
print("\nSignificant columns after chi-square test")  
print(cols)  
features = X.columns[cols]  
print("Significant column names after chi-square test")  
print(np.array(features))  
  
  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
  
X = X[['PhysActivity', 'Fruits', 'Veggies',  
 'HvyAlcoholConsump', 'Education', 'Income']]  
  
X\_Scale = sc\_x.fit\_transform(X)  
  
# Split data.  
X\_train,X\_test,y\_train,y\_test = train\_test\_split(X\_Scale, y, test\_size=0.25,  
 random\_state=0)  
  
# Build logistic regression model and make predictions.  
logisticModel = LogisticRegression(fit\_intercept=True, solver='liblinear',  
 random\_state=0)  
logisticModel.fit(X\_train,y\_train)  
y\_pred=logisticModel.predict(X\_test)  
print("\nPredictions from logistic model")  
print(y\_pred)  
  
  
# enumerate splits - returns train and test arrays of indexes.  
# scikit-learn k-fold cross-validation  
from sklearn.model\_selection import KFold  
#kfold = KFold(3, True)  
kfold = KFold(n\_splits=8, shuffle=True)  
count = 0  
  
accuracyList = []  
precisionList = []  
recallList = []  
f1List = []  
  
for train\_index, test\_index in kfold.split(X\_Scale):  
  
 X\_train, X\_test = X\_Scale[train\_index], X\_Scale[test\_index]  
 y\_train, y\_test = y[train\_index], y[test\_index]  
  
 # Perform logistic regression.  
 logisticModel = LogisticRegression(fit\_intercept=True,  
 solver='liblinear')  
 # Fit the model.  
 logisticModel.fit(X\_train, np.ravel(y\_train))  
  
 y\_pred = logisticModel.predict(X\_test)  
 y\_prob = logisticModel.predict\_proba(X\_test)  
  
 # Show confusion matrix and accuracy scores.  
 cm = pd.crosstab(y\_test, y\_pred,  
 rownames=['Actual'],  
 colnames=['Predicted'])  
 count += 1  
 print("\n\*\*\*K-fold: " + str(count))  
  
 # Calculate accuracy and precision scores and add to the list.  
 accuracy = metrics.accuracy\_score(y\_test, y\_pred)  
 precision = metrics.precision\_score(y\_test, y\_pred)  
 recall = metrics.recall\_score(y\_test, y\_pred)  
 f1 = metrics.f1\_score(y\_test, y\_pred)  
  
 print('\nAccuracy: ', accuracy)  
 accuracyList.append(accuracy)  
 print("\nPrecision: ", precision)  
 precisionList.append(precision)  
 print("\nRecall: ", recall)  
 recallList.append(recall)  
 print("\nF1: ", f1)  
 f1List.append(f1)  
 print("\nConfusion Matrix")  
 print(cm)  
  
print("\nAccuracy and Standard Deviation For All Folds:")  
print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
print("Average Accuracy: ")  
print(np.mean(accuracyList))  
print("Accuracy SD: ")  
print(np.std(accuracyList))  
print("Average Precision: ")  
print(np.mean(precisionList))  
print("Precision SD: ")  
print(np.std(precisionList))  
print("Average Recall: ")  
print(np.mean(recallList))  
print("Recall SD: ")  
print(np.std(recallList))  
print("Average F1: ")  
print(np.mean(f1List))  
print("F1 SD: ")  
print(np.std(f1List))  
  
  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
  
  
# split into train-test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=27)  
  
# train models  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
  
# logistic regression  
model1 = LogisticRegression()  
# knn  
model2 = KNeighborsClassifier(n\_neighbors=4)  
  
# fit model  
model1.fit(X\_train, y\_train)  
model2.fit(X\_train, y\_train)  
  
# predict probabilities  
pred\_prob1 = model1.predict\_proba(X\_test)  
pred\_prob2 = model2.predict\_proba(X\_test)  
  
from sklearn.metrics import roc\_curve  
  
# roc curve for models  
fpr1, tpr1, thresh1 = roc\_curve(y\_test, pred\_prob1[:,1], pos\_label=1)  
fpr2, tpr2, thresh2 = roc\_curve(y\_test, pred\_prob2[:,1], pos\_label=1)  
  
# roc curve for tpr = fpr  
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr, p\_tpr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
# Compute AUC score.  
from sklearn.metrics import roc\_auc\_score  
  
# auc scores  
auc\_score1 = roc\_auc\_score(y\_test, pred\_prob1[:,1])  
auc\_score2 = roc\_auc\_score(y\_test, pred\_prob2[:,1])  
  
print("AUC scores")  
print(auc\_score1, auc\_score2)  
  
# matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')  
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')  
plt.plot(p\_fpr, p\_tpr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
  
plt.legend(loc='best')  
plt.savefig('ROC',dpi=300)  
plt.show();

Logistic Regression Model Python Script for Model 4

from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score  
from sklearn.model\_selection import train\_test\_split  
import numpy as np  
import pandas as pd  
  
# Show all columns.  
pd.set\_option('display.max\_columns', None)  
pd.set\_option('display.width', 1000)  
  
# Prepare the data.  
PATH = "C:\\datasets\\"  
CSV\_DATA = "diabetes.csv"  
  
df = pd.read\_csv(PATH + CSV\_DATA)  
df['bmiBin'] = pd.cut(x=df['BMI'], bins=[0, 18.49, 24.9, 29.9, 34.9, 39.9, 99])  
  
tempDf = df[['bmiBin']] # Isolate columns  
dummyDf = pd.get\_dummies(tempDf, columns=['bmiBin'])  
X = pd.concat(([df, dummyDf]), axis=1) # Join dummy df with original  
del X['bmiBin']  
  
X.rename(columns={'bmiBin\_(0.0, 18.49]':'BMI - Underweight','bmiBin\_(18.49, 24.9]':'BMI - Normal', 'bmiBin\_(24.9, 29.9]':'BMI - Overweight', 'bmiBin\_(29.9, 34.9]':'BMI - Obese Class 1', 'bmiBin\_(34.9, 39.9]':'BMI - Obese Class 2', 'bmiBin\_(39.9, 99.0]':'BMI - Obese Class 3' }, inplace=True)  
  
X = X.copy()  
del X['Diabetes\_binary']  
y = df['Diabetes\_binary']  
  
print(X.head())  
  
def getUnfitModels():  
 models = list()  
 models.append(LogisticRegression())  
 models.append(DecisionTreeClassifier())  
 models.append(AdaBoostClassifier())  
 models.append(RandomForestClassifier(n\_estimators=10))  
 return models  
  
def evaluateModel(y\_test, predictions, model):  
 precision = round(precision\_score(y\_test, predictions),2)  
 recall = round(recall\_score(y\_test, predictions), 2)  
 f1 = round(f1\_score(y\_test, predictions), 2)  
 accuracy = round(accuracy\_score(y\_test, predictions), 2)  
  
 print("Precision:" + str(precision) + " Recall:" + str(recall) +\  
 " F1:" + str(f1) + " Accuracy:" + str(accuracy) +\  
 " " + model.\_\_class\_\_.\_\_name\_\_)  
  
def fitBaseModels(X\_train, y\_train, X\_test, models):  
 dfPredictions = pd.DataFrame()  
  
 # Fit base model and store its predictions in dataframe.  
 for i in range(0, len(models)):  
 models[i].fit(X\_train, y\_train)  
 predictions = models[i].predict(X\_test)  
 colName = str(i)  
 dfPredictions[colName] = predictions  
 return dfPredictions, models  
  
def fitStackedModel(X, y):  
 model = LogisticRegression()  
 model.fit(X, y)  
 return model  
  
# Split data into train, test and validation sets.  
X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.70)  
X\_test, X\_val, y\_test, y\_val = train\_test\_split(X\_temp, y\_temp, test\_size=0.50)  
  
# Get base models.  
unfitModels = getUnfitModels()  
  
# Fit base and stacked models.  
dfPredictions, models = fitBaseModels(X\_train, y\_train, X\_test, unfitModels)  
stackedModel = fitStackedModel(dfPredictions, y\_test)  
  
# Evaluate base models with validation data.  
print("\n\*\* Evaluate Base Models \*\*")  
dfValidationPredictions = pd.DataFrame()  
for i in range(0, len(models)):  
 predictions = models[i].predict(X\_val)  
 colName = str(i)  
 dfValidationPredictions[colName] = predictions  
 evaluateModel(y\_val, predictions, models[i])  
  
# Evaluate stacked model with validation data.  
stackedPredictions = stackedModel.predict(dfValidationPredictions)  
print("\n\*\* Evaluate Stacked Model \*\*")  
evaluateModel(y\_val, stackedPredictions, stackedModel)  
  
  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
  
  
# split into train-test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=27)  
  
# train models  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
  
# logistic regression  
model1 = LogisticRegression()  
# knn  
model2 = KNeighborsClassifier(n\_neighbors=4)  
  
# fit model  
model1.fit(X\_train, y\_train)  
model2.fit(X\_train, y\_train)  
  
# predict probabilities  
pred\_prob1 = model1.predict\_proba(X\_test)  
pred\_prob2 = model2.predict\_proba(X\_test)  
  
from sklearn.metrics import roc\_curve  
  
# roc curve for models  
fpr1, tpr1, thresh1 = roc\_curve(y\_test, pred\_prob1[:,1], pos\_label=1)  
fpr2, tpr2, thresh2 = roc\_curve(y\_test, pred\_prob2[:,1], pos\_label=1)  
  
# roc curve for tpr = fpr  
random\_probs = [0 for i in range(len(y\_test))]  
p\_fpr, p\_tpr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)  
  
# Compute AUC score.  
from sklearn.metrics import roc\_auc\_score  
  
# auc scores  
auc\_score1 = roc\_auc\_score(y\_test, pred\_prob1[:,1])  
auc\_score2 = roc\_auc\_score(y\_test, pred\_prob2[:,1])  
print("AUC scores")  
print(auc\_score1, auc\_score2)  
  
# matplotlib  
import matplotlib.pyplot as plt  
plt.style.use('seaborn')  
  
# plot roc curves  
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')  
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')  
plt.plot(p\_fpr, p\_tpr, linestyle='--', color='blue')  
# title  
plt.title('ROC curve')  
# x label  
plt.xlabel('False Positive Rate')  
# y label  
plt.ylabel('True Positive rate')  
  
plt.legend(loc='best')  
plt.savefig('ROC',dpi=300)  
plt.show()