#!/usr/bin/env python

# coding: utf-8

# In[1]:

# for numerical computing

import numpy as np

# for dataframes

import pandas as pd

# for easier visualization

import seaborn as sns

# for visualization and to display plots

from matplotlib import pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# import color maps

from matplotlib.colors import ListedColormap

# Ignore Warnings

import warnings

warnings.filterwarnings("ignore")

from math import sqrt

# to split train and test set

from sklearn.model\_selection import train\_test\_split

# to perform hyperparameter tuning

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.model\_selection import cross\_val\_score

# Machine Learning Models

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.metrics import roc\_curve, auc, roc\_auc\_score, confusion\_matrix

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.neighbors import KNeighborsClassifier

from matplotlib.colors import ListedColormap

from sklearn.metrics import accuracy\_score

#import xgboost

import os

mingw\_path = 'C:\\Program Files\\mingw-w64\\x86\_64-7.2.0-posix-seh-rt\_v5-rev0\\mingw64\\bin'

os.environ['PATH'] = mingw\_path + ';' + os.environ['PATH']

from xgboost import XGBClassifier

from xgboost import plot\_importance

# to plot feature importance

# In[2]:

df=pd.read\_csv('indian\_liver\_patient.csv')

# In[3]:

df.shape

# In[4]:

df.columns

# In[5]:

df.head()

# In[6]:

df.dtypes[df.dtypes=='object']

# In[7]:

# Plot histogram grid

df.hist(figsize=(15,15), xrot=-45, bins=10) ## Display the labels rotated by 45 degress

# Clear the text "residue"

plt.show()

# In[8]:

df.describe()

# In[9]:

## if score==negative, mark 0 ;else 1

def partition(x):

if x == 2:

return 0

return 1

df['Dataset'] = df['Dataset'].map(partition)

# In[10]:

df.describe(include=['object'])

# In[11]:

plt.figure(figsize=(5,5))

sns.countplot(y='Gender', data=df)

# In[12]:

df[df['Gender'] == 'Male'][['Dataset', 'Gender']].head()

# In[13]:

sns.factorplot (x="Age", y="Gender", hue="Dataset", data=df);

# In[14]:

sns.countplot(data=df, x = 'Gender', label='Count')

M, F = df['Gender'].value\_counts()

print('Number of patients that are male: ',M)

print('Number of patients that are female: ',F)

# In[15]:

## if score==negative, mark 0 ;else 1

def partition(x):

if x =='Male':

return 0

return 1

df['Gender'] = df['Gender'].map(partition)

# In[16]:

sns.set\_style('whitegrid') ## Background Grid

sns.FacetGrid(df, hue = 'Dataset', size = 5).map(plt.scatter, 'Total\_Bilirubin', 'Direct\_Bilirubin').add\_legend()

# In[17]:

sns.set\_style('whitegrid') ## Background Grid

sns.FacetGrid(df, hue = 'Dataset', size = 5).map(plt.scatter, 'Total\_Bilirubin', 'Albumin').add\_legend()

# In[18]:

sns.set\_style('whitegrid') ## Background Grid

sns.FacetGrid(df, hue = 'Dataset', size = 5).map(plt.scatter, 'Total\_Protiens', 'Albumin\_and\_Globulin\_Ratio').add\_legend()

# In[19]:

df.corr()

# In[20]:

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

sns.heatmap(df.corr())

# In[21]:

mask=np.zeros\_like(df.corr())

mask[np.triu\_indices\_from(mask)] = True

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

with sns.axes\_style("white"):

ax = sns.heatmap(df.corr()\*100, mask=mask, fmt='.0f', annot=True, lw=1, cmap=ListedColormap(['green', 'yellow', 'red','blue']))

# In[22]:

df = df.drop\_duplicates()

print( df.shape )

# In[23]:

sns.boxplot(df.Aspartate\_Aminotransferase)

# In[24]:

df.Aspartate\_Aminotransferase.sort\_values(ascending=False).head()

# In[25]:

df = df[df.Aspartate\_Aminotransferase <=3000 ]

df.shape

# In[26]:

sns.boxplot(df.Aspartate\_Aminotransferase)

# In[27]:

df.Aspartate\_Aminotransferase.sort\_values(ascending=False).head()

# In[28]:

df = df[df.Aspartate\_Aminotransferase <=2500 ]

df.shape

# In[29]:

df.isnull().values.any()

# In[30]:

df=df.dropna(how='any')

# In[31]:

df.shape

# In[32]:

df.head()

# In[33]:

# Create separate object for target variable

y = df.Dataset

# Create separate object for input features

X = df.drop('Dataset', axis=1)

# In[34]:

# Split X and y into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.2,

random\_state=1234,

stratify=df.Dataset)

# In[35]:

# Print number of observations in X\_train, X\_test, y\_train, and y\_test

print(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

# In[36]:

train\_mean = X\_train.mean()

train\_std = X\_train.std()

# In[37]:

## Standardize the train data set

X\_train = (X\_train - train\_mean) / train\_std

# In[38]:

## Check for mean and std dev.

X\_train.describe()

# In[39]:

## Note: We use train\_mean and train\_std\_dev to standardize test data set

X\_test = (X\_test - train\_mean) / train\_std

# In[40]:

## Check for mean and std dev. - not exactly 0 and 1

X\_test.describe()

# In[41]:

tuned\_params = {'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1', 'l2']}

model = GridSearchCV(LogisticRegression(), tuned\_params, scoring = 'roc\_auc', n\_jobs=-1)

model.fit(X\_train, y\_train)

# In[42]:

model.best\_estimator\_

# In[43]:

## Predict Train set results

y\_train\_pred = model.predict(X\_train)

# In[44]:

## Predict Test set results

y\_pred = model.predict(X\_test)

# In[45]:

# Get just the prediction for the positive class (1)

y\_pred\_proba = model.predict\_proba(X\_test)[:,1]

# In[46]:

# Display first 10 predictions

y\_pred\_proba[:10]

# In[47]:

i=28 ## Change the value of i to get the details of any point (56, 213, etc.)

print('For test point {}, actual class = {}, precited class = {}, predicted probability = {}'.

format(i, y\_test.iloc[i], y\_pred[i], y\_pred\_proba[i]))

# In[48]:

confusion\_matrix(y\_test, y\_pred).T

# In[49]:

# Calculate ROC curve from y\_test and pred

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

# In[50]:

# Plot the ROC curve

fig = plt.figure(figsize=(8,8))

plt.title('Receiver Operating Characteristic')

# Plot ROC curve

plt.plot(fpr, tpr, label='l1')

plt.legend(loc='lower right')

# Diagonal 45 degree line

plt.plot([0,1],[0,1],'k--')

# Axes limits and labels

plt.xlim([-0.1,1.1])

plt.ylim([-0.1,1.1])

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.show()

# In[51]:

# Calculate AUC for Train set

print(roc\_auc\_score(y\_train, y\_train\_pred))

# In[52]:

# Calculate AUC for Test set

print(auc(fpr, tpr))

# In[53]:

## Building the model again with the best hyperparameters

model = LogisticRegression(C=1, penalty = 'l2')

model.fit(X\_train, y\_train)

# In[54]:

indices = np.argsort(-abs(model.coef\_[0,:]))

print("The features in order of importance are:")

print(50\*'-')

for feature in X.columns[indices]:

print(feature)

# In[82]:

# creating odd list of K for KNN

neighbors = list(range(1,20,2))

# empty list that will hold cv scores

cv\_scores = []

# 10-fold cross validation , 9 datapoints will be considered for training and 1 for cross validation (turn by turn) to determine value of k

for k in neighbors:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X\_train, y\_train, cv=5, scoring='accuracy')

cv\_scores.append(scores.mean())

# changing to misclassification error

MSE = [1 - x for x in cv\_scores]

# determining best k

optimal\_k = neighbors[MSE.index(min(MSE))]

print('\nThe optimal number of neighbors is %d.' % optimal\_k)

# In[83]:

MSE.index(min(MSE))

# In[84]:

# plot misclassification error vs k

plt.plot(neighbors, MSE)

plt.xlabel('Number of Neighbors K')

plt.ylabel('Misclassification Error')

plt.show()

# In[85]:

classifier = KNeighborsClassifier(n\_neighbors = optimal\_k)

classifier.fit(X\_train, y\_train)

# In[86]:

y\_pred = classifier.predict(X\_test)

# In[87]:

y\_train\_pred = classifier.predict(X\_train)

# In[88]:

acc = accuracy\_score(y\_test, y\_pred, normalize=True) \* float(100) ## get the accuracy on testing data

acc

# In[89]:

cnf=confusion\_matrix(y\_test,y\_pred).T

cnf

# In[90]:

# Get just the prediction for the positive class (1)

y\_pred\_proba = classifier.predict\_proba(X\_test)[:,1]

# In[91]:

# Display first 10 predictions

y\_pred\_proba[:10]

# In[92]:

# Calculate ROC curve from y\_test and pred

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

# In[93]:

# Plot the ROC curve

fig = plt.figure(figsize=(8,8))

plt.title('Receiver Operating Characteristic')

# Plot ROC curve

plt.plot(fpr, tpr, label='l1')

plt.legend(loc='lower right')

# Diagonal 45 degree line

plt.plot([0,1],[0,1],'k--')

# Axes limits and labels

plt.xlim([-0.1,1.1])

plt.ylim([-0.1,1.1])

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.show()

# In[94]:

# Calculate AUC for Train

roc\_auc\_score(y\_train, y\_train\_pred)

# In[95]:

# Calculate AUC for Test

print(auc(fpr, tpr))

# In[109]:

from sklearn import svm

def svc\_param\_selection(X, y, nfolds):

Cs = [0.001, 0.01, 0.1, 1, 10]

gammas = [0.001, 0.01, 0.1, 1]

param\_grid = {'C': Cs, 'gamma' : gammas}

grid\_search = GridSearchCV(svm.SVC(kernel='rbf'), param\_grid, cv=nfolds)

grid\_search.fit(X\_train, y\_train)

grid\_search.best\_params\_

return grid\_search.best\_params\_

# In[110]:

svClassifier=SVC(kernel='rbf',probability=True)

svClassifier.fit(X\_train,y\_train)

# In[111]:

svc\_param\_selection(X\_train,y\_train,5)

# In[112]:

###### Building the model again with the best hyperparameters

model = SVC(C=1, gamma=1)

model.fit(X\_train, y\_train)

# In[113]:

## Predict Train results

y\_train\_pred = model.predict(X\_train)

# In[114]:

## Predict Test results

y\_pred = model.predict(X\_test)

# In[115]:

confusion\_matrix(y\_test, y\_pred).T

# In[117]:

# Calculate ROC curve from y\_test and pred

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

# In[118]:

# Plot the ROC curve

fig = plt.figure(figsize=(8,8))

plt.title('Receiver Operating Characteristic')

# Plot ROC curve

plt.plot(fpr, tpr, label='l1')

plt.legend(loc='lower right')

# Diagonal 45 degree line

plt.plot([0,1],[0,1],'k--')

# Axes limits and labels

plt.xlim([-0.1,1.1])

plt.ylim([-0.1,1.1])

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.show()

# In[119]:

# Calculate AUC for Train

roc\_auc\_score(y\_train, y\_train\_pred)

# In[120]:

print(auc(fpr, tpr))

# In[130]:

# Neural Networks# Neural

neural = MLPClassifier(hidden\_layer\_sizes=40,

activation='relu',

solver='adam',

alpha=0.001,

batch\_size='auto',

max\_iter=200,

random\_state=137,

tol=0.0001,

early\_stopping=False,

validation\_fraction=0.1,

beta\_1=0.9,

beta\_2=0.999,

epsilon=1e-08,

learning\_rate='constant',

power\_t=0.5,

momentum=0.8,

nesterovs\_momentum=True,

shuffle=True,

learning\_rate\_init=0.001)

neural.fit(X\_train, y\_train)

#Predict Output

predicted = neural.predict(X\_test)

neural\_score = round(neural.score(X\_train, y\_train) \* 100, 2)

neural\_score\_test = round(neural.score(X\_test, y\_test) \* 100, 2)

print('Neural Score: \n', neural\_score)

print('Neural Test Score: \n', neural\_score\_test)

print('Accuracy: \n', accuracy\_score(y\_test, predicted))

print(confusion\_matrix(predicted,y\_test))

print(classification\_report(y\_test,predicted))

# In[131]:

## Predict Train results

y\_train\_pred = neural.predict(X\_train)

# In[132]:

## Predict Test results

y\_pred = neural.predict(X\_test)

# In[133]:

# Calculate ROC curve from y\_test and pred

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

# In[134]:

# Plot the ROC curve

fig = plt.figure(figsize=(8,8))

plt.title('Receiver Operating Characteristic')

# Plot ROC curve

plt.plot(fpr, tpr, label='l1')

plt.legend(loc='lower right')

# Diagonal 45 degree line

plt.plot([0,1],[0,1],'k--')

# Axes limits and labels

plt.xlim([-0.1,1.1])

plt.ylim([-0.1,1.1])

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.show()

# In[135]:

roc\_auc\_score(y\_train,y\_train\_pred )

# In[136]:

# Calculate AUC for Test

print(auc(fpr, tpr))