**Import Libraries**

%matplotlib inline

import numpy as np

import pandas as pd

import seaborn as s

from sklearn import model\_selection

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

from sklearn.model\_selection import KFold

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

from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score

from sklearn import svm

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, LabelEncoder

#optimum parameter choosing

from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

from xgboost import XGBClassifier

import pickle

import os

import warnings

warnings.filterwarnings('ignore')

**Load Dataset**

# Data file is in the user’s folder

os.chdir ('C:\\Noble\\Training\\Top Mentor\\Training\\Presentation\\Project\\Project to Explain Classifciation\\')

data = pd.read\_csv('data.csv')

display (data)

**Display Shape**

display (data.shape)

**Create Variable df from data**

df = data

display (df)

**print distinct value count**

display (df['diagnosis'].value\_counts())

**Print Data Types**

print (df.dtypes)

**Convert Diagnosis column type to category**

df['diagnosis']= df['diagnosis'].astype('category')

display (df.dtypes)

**Display the data set**

df.head()

**Label Encoding – Diagnosis**

le = LabelEncoder()

df['diagnosis']= le.fit\_transform (df['diagnosis'])

display (df)

**Create X variable with out diagnosis column**

x= df.drop ('diagnosis',axis =1).drop('id',axis =1)

display (x)

**Create Y variable (diagnosis)**

y = df['diagnosis']

display (y)

**Extract all column names**

col = x. columns

display (col)

**Null Check**

x.isnull().sum()

**Normalization**

x\_norm = (x- x.mean()) / (x.max()- x.min())

x\_norm= pd.concat ([x\_norm,y], axis =1 )

display (x\_norm)

**Print Co relation**

co\_rel= x\_norm.corr()

display (co\_rel)

**Heat map with co relation**

plt.rcParams['figure.figsize']=(20,12)

s.set(font\_scale=1.4)

# In co relation 1 is the highest and -1 is lowest

s.heatmap (co\_rel,cmap = 'coolwarm',annot = True)

plt.show()

**Heat map with out corelation value – Annot = None**

plt.rcParams['figure.figsize']=(20,12)

s.set(font\_scale=1.4)

# In co relation 1 is the highest and -1 is lowest

s.heatmap (co\_rel,cmap = 'coolwarm',annot = None)

plt.show()

**Create Box Plot**

#box plot to check the outliers. Not going to remove outliers since this data is important.

# Observation, when 'diagnosis' is "B", the values are lower

plt.rcParams['figure.figsize']=(20,8)

f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)

s.boxplot ('diagnosis', y = 'radius\_mean',data = df , ax = ax1)

s.boxplot ('diagnosis', y = 'texture\_mean',data = df , ax = ax2)

s.boxplot ('diagnosis', y = 'perimeter\_mean',data = df , ax = ax3)

s.boxplot ('diagnosis', y = 'area\_mean',data = df , ax = ax4)

s.boxplot ('diagnosis', y = 'smoothness\_mean',data = df , ax = ax5)

f .tight\_layout()

f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)

s.boxplot ('diagnosis', y = 'compactness\_mean',data = df , ax = ax1)

s.boxplot ('diagnosis', y = 'concavity\_mean',data = df , ax = ax2)

s.boxplot ('diagnosis', y = 'concave points\_mean',data = df , ax = ax3)

s.boxplot ('diagnosis', y = 'symmetry\_mean',data = df , ax = ax4)

s.boxplot ('diagnosis', y = 'fractal\_dimension\_mean',data = df , ax = ax5)

f .tight\_layout()

**Distribution plot**

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "radius\_mean", hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, 'texture\_mean', hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, 'perimeter\_mean', hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "area\_mean", hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "smoothness\_mean", hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "compactness\_mean", hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "concavity\_mean", hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "concave points\_mean", hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "symmetry\_mean", hist = False, rug = True)

g = s.FacetGrid (df,col = 'diagnosis', hue = 'diagnosis')

g.map (s.distplot, "fractal\_dimension\_mean", hist = False, rug = True)

**Box Plot**

plt.rcParams['figure.figsize']=(20,8)

f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)

s.boxplot ('diagnosis', y = 'radius\_se',data = df , ax = ax1,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'texture\_se',data = df , ax = ax2,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'perimeter\_se',data = df , ax = ax3,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'area\_se',data = df , ax = ax4,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'smoothness\_se',data = df , ax = ax5,palette = 'cubehelix')

f .tight\_layout()

f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)

s.boxplot ('diagnosis', y = 'compactness\_se',data = df , ax = ax1,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'concavity\_se',data = df , ax = ax2,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'concave points\_se',data = df , ax = ax3,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'symmetry\_se',data = df , ax = ax4,palette = 'cubehelix')

s.boxplot ('diagnosis', y = 'fractal\_dimension\_se',data = df , ax = ax5,palette = 'cubehelix')

f .tight\_layout()

**Box Plot**

plt.rcParams['figure.figsize']=(20,8)

f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)

s.boxplot ('diagnosis', y = 'radius\_worst',data = df , ax = ax1,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'texture\_worst',data = df , ax = ax2,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'perimeter\_worst',data = df , ax = ax3,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'area\_worst',data = df , ax = ax4,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'smoothness\_worst',data = df , ax = ax5,palette = 'coolwarm')

f .tight\_layout()

f, (ax1,ax2,ax3,ax4,ax5) = plt.subplots (1,5)

s.boxplot ('diagnosis', y = 'compactness\_worst',data = df , ax = ax1,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'concavity\_worst',data = df , ax = ax2,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'concave points\_worst',data = df , ax = ax3,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'symmetry\_worst',data = df , ax = ax4,palette = 'coolwarm')

s.boxplot ('diagnosis', y = 'fractal\_dimension\_worst',data = df , ax = ax5,palette = 'coolwarm')

f .tight\_layout()

**Function for Model fitting and best parameter values**

# Functionalize Model Fitting

# User Randon seed to get the same value everywhere

n\_jobs : int, default=None

Number of jobs to run in parallel.

``None`` means 1 unless in a :obj:`joblib.parallel\_backend` context.

``-1`` means using all processors. See :term:`Glossary <n\_jobs>`

for more details.

def FitModel (X,Y, algo\_name , algorithm, gridSearchParams, cv):

np.random.seed(10)

x\_train, x\_test, y\_train, y\_test = train\_test\_split (X,Y,test\_size = 0.2)

# Find the Parameters , then choose best parameters

grid = GridSearchCV(estimator = algorithm, param\_grid = gridSearchParams,

cv = cv, scoring = 'accuracy', verbose = 1 , n\_jobs = -1 )

grid\_result = grid.fit(x\_train, y\_train)

best\_params = grid\_result.best\_params\_

pred = grid\_result.predict (x\_test)

cm = confusion\_matrix (y\_test,pred)

print (pred)

pickle.dump(grid\_result,open(algo\_name,'wb'))

print ('Best Params :', best\_params)

print ('Classification Report:',classification\_report(y\_test,pred))

print ('Accuracy Score', (accuracy\_score(y\_test,pred)))

print ('Confusion Matrix :\n',cm)

**Create SVM Model**

#cv = cross validation

param = {

'C': [0.1,1,100,1000],

'gamma':[0.0001,0.001, 0.005, 0.1,1, 3,5,10, 100]

}

FitModel (x\_norm,y,'SVC',SVC(), param, cv =10)

**Create Random Forest**

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (x\_norm,y,'Random Forest',RandomForestClassifier(), param, cv =10)

**Create Random Forest Normal Way**

np.random.seed(10)

x\_train,x\_test, y\_train,y\_test = train\_test\_split (x\_norm,y,test\_size = 0.2)

forest = RandomForestClassifier (n\_estimators = 500)

fit = forest.fit (x\_train, y\_train)

accuracy = fit.score(x\_test,y\_test)

predict = fit.predict(x\_test)

cmatrix = confusion\_matrix (y\_test, predict)

print ('Classification Report:',classification\_report(y\_test,predict))

print ('Accuracy Score', (accuracy\_score(y\_test,predict)))

print ('Accuracy of Random Forest ', (accuracy))

print ('Confusion Matrix :\n',cmatrix)

**XG Boost**

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (x\_norm,y,'XGBoost', XGBClassifier(),param, cv = 10)

**Balancing the Data**

**Install the Following packages in Anaconda**

# pip install imblearn

**Over Sampling Algorithm**

from imblearn.over\_sampling import SMOTE

**Data Split**

df['diagnosis'].value\_counts()

**Over Sampling**

sm = SMOTE(random\_state =42)

X\_res, Y\_res = sm.fit\_resample (x\_norm, y)

**Print Count**

display (Y\_res.value\_counts())

**Create Random Forest Model**

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (X\_res, Y\_res ,'Random Forest',RandomForestClassifier(), param, cv =10)

**Create SVC Model**

param = {

'C': [0.1,1,100,1000],

'gamma':[0.0001,0.001, 0.005, 0.1,1, 3,5,10, 100]

}

FitModel (X\_res, Y\_res,'SVC',SVC(), param, cv =10)

**XG Boost Model**

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (X\_res, Y\_res,'XGBoost', XGBClassifier(),param, cv = 10)

**Feature Selection**

**Feature Importance /Feature Selection**

importances = forest.feature\_importances\_

indices = np.argsort(importances)[::-1]

print ("Feature Ranking:")

for f in range (x\_norm.shape[1]):

print ("Feature %s (%f)" %(list (x\_norm)[f],importances[indices[f]]))

**Feature Selection Chart**

feat\_imp = pd.DataFrame({'Feature': list(x\_norm), 'Gini importance': importances[indices]})

plt.rcParams['figure.figsize']= (12,12)

s.set\_style ('whitegrid')

ax= s.barplot(x ='Gini importance', y = 'Feature', data = feat\_imp )

ax.set (xlabel = 'Gini Importances')

plt.show()

feat\_imp.index = feat\_imp.Feature

**Get first 15 Columns**

feat\_to\_keep = feat\_imp.iloc[:15].index

type(feat\_to\_keep),feat\_to\_keep

**Create Random Forest Model**

X\_res = pd.DataFrame(X\_res)

Y\_res = pd.DataFrame(Y\_res)

X\_res.columns = x\_norm.columns

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (X\_res [feat\_to\_keep], Y\_res ,'Random Forest',RandomForestClassifier(), param, cv =10)

**Create SVM Forest Model**

param = {

'C': [0.1,1,100,1000],

'gamma':[0.0001,0.001, 0.005, 0.1,1, 3,5,10, 100]

}

FitModel (X\_res [feat\_to\_keep], Y\_res,'SVC',SVC(), param, cv =5)

**Create XGBOOST Model**

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (X\_res [feat\_to\_keep], Y\_res,'XGBoost', XGBClassifier(),param, cv = 5)

**Create the model with resample data**

**Random Forest Classifier**

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (X\_res, Y\_res ,'Random Forest',RandomForestClassifier(), param, cv =10)

**Support Vector Machine**

param = {

'C': [0.1,1,100,1000],

'gamma':[0.0001,0.001, 0.005, 0.1,1, 3,5,10, 100]

}

FitModel (X\_res, Y\_res,'SVC',SVC(), param, cv =10)

**XG Boost Classifier**

param = { 'n\_estimators': [100,500,1000,2000] }

FitModel (X\_res, Y\_res,'XGBoost', XGBClassifier(),param, cv = 10)

**Load Pickle file XG Boost**

load\_model =pickle.load(open("XGBoost","rb"))

**Prediction**

pred1 = load\_model.predict (x\_test)

pred1

**Best Parameter**

load\_model.best\_params\_

**Accuracy Score**

print (accuracy\_score (pred1,y\_test))

**Load Pickle file Support Vector Machine**

load\_model =pickle.load(open("SVC","rb"))

pred1 = load\_model.predict (x\_test)

print (load\_model.best\_params\_)

print (accuracy\_score (pred1,y\_test))

display (pred1)

**Load Pickle file Random Forest**

load\_model =pickle.load(open("Random Forest","rb"))

pred1 = load\_model.predict (x\_test)

print (load\_model.best\_params\_)

print (accuracy\_score (pred1,y\_test))

display (pred1)



