Experiment - 1

Aim-To predict Bicarbonate(ppm) present in the well water of Northwest Texas data via Linear Regression Machine learning moodel.

```
cd /content/drive/MyDrive/Machine Learning/Colab Notebooks/ML
Practicals/1_Practical/Linear regression P1
```

```
/content/drive/MyDrive/Machine Learning/Colab Notebooks/ML
Practicals/1_Practical/Linear regression P1
```

```
ls
```

```
 edcCO2.csv     'Ground Water Survey.csv'
 fruitohms.csv  'Linear regression_1.ipynb'
```

Importing Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
#import data set
dataset = pd.read_csv('Ground Water Survey.csv')
X= dataset.iloc[:,:-1].values
Y= dataset.iloc[:,1].values
dataset.head()
```

```
     X    Y
0  7.6  157
1  7.1  174
2  8.2  175
3  7.5  188
4  7.4  171
```

In the following data

X = pH of well water

Y = Bicarbonate (parts per million) of well water

The data is by water well from a random sample of wells in Northwest Texas. Reference: Union Carbide Technical Report K/UR-1

```
dataset.tail()
```

```
      X    Y
29  8.5   48
30  7.8  147
31  6.7  117
```

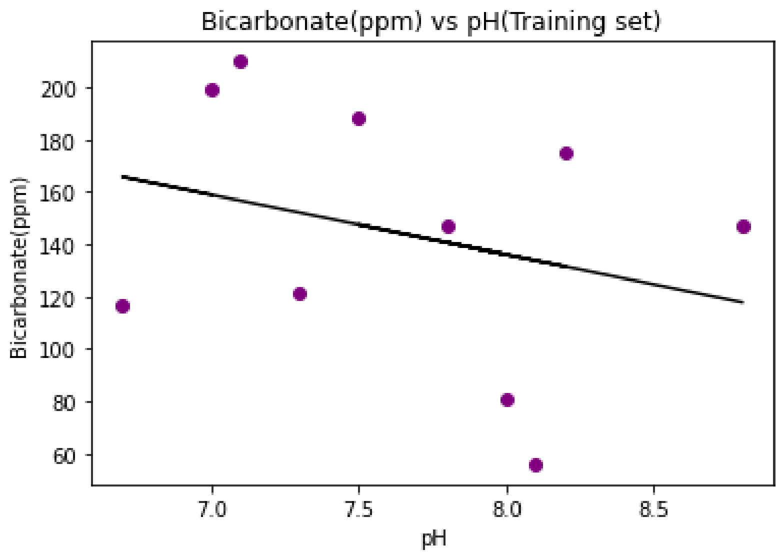
```
32  7.1  182
33  7.3   87
```

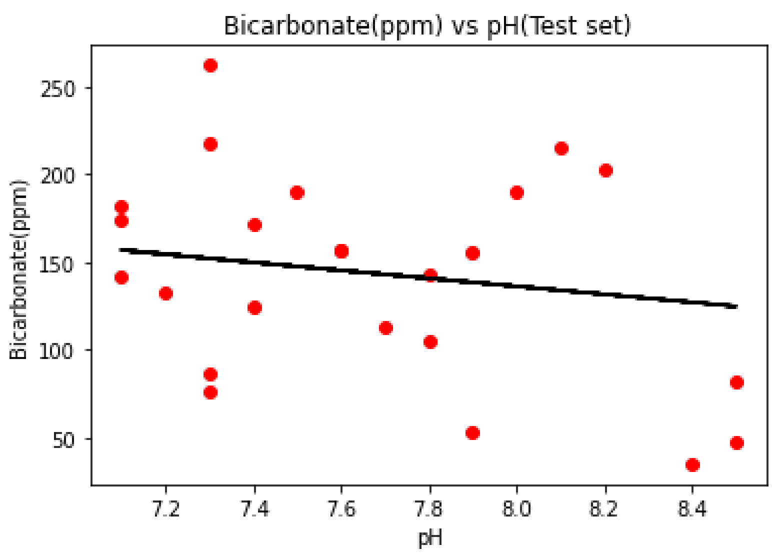
Bicarbonate can be found in water with a pH between 4.3 and 12.3. Above a pH of 8.3, carbonate is also present.

```
#Splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X,Y,test_size= 0.7)

#Fitting Simple Linear Regression ipynb
#This is called Model
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(X_train,Y_train)
```

```
LinearRegression()
```

```
##Predicting the test results
Y_pred= regressor.predict(X_test)

#Visualising the training set Results

plt.scatter(X_train, Y_train, color='Purple')
plt.plot(X_train, regressor.predict(X_train), color='black')
plt.title('Bicarbonate(ppm) vs pH(Training set)')
plt.xlabel('pH')
plt.ylabel('Bicarbonate(ppm)')
plt.show()

plt.scatter(X_test, Y_test, color='red')
plt.plot(X_test, regressor.predict(X_test), color='black')
plt.title('Bicarbonate(ppm) vs pH(Test set)')
plt.xlabel('pH')
plt.ylabel('Bicarbonate(ppm)')
plt.show()
```






```
print(regressor.predict([[7.6]]))
```

```
[145.2435247]
```

Now we will perform the prediction of Bicarbonate(ppm) Present in the well water.

```
a=float(input("What is the pH of your well water? "))
print('The Bicarbonate (parts per million) in your well water',
regressor.predict([[a]]))
```

```
What is the pH of your well water? 7.1
The Bicarbonate (parts per million) in your well water [156.6787717]
```

Conclusion- Hence we are able to predict the Bicarbonate(ppm) present in the well water of Northeast Texas by training the Linear Regression model with the Water Survey Dataset.

Experiment – 2

Aim- To predict Coronary Heart Disease using Logistic Regression Classifier

Logistic Regression: The target variable has three or more nominal categories such as predicting the type of Wine. Ordinal Logistic Regression: the target variable has three or more ordinal categories such as restaurant or product rating from 1 to 5. Model building in Scikit-learn Let's build the diabetes prediction model.

Here, We are going to predict Coronary Heart Disease using Logistic Regression Classifier.

Let's first load the required Coronary Heart Disease dataset using the pandas' read CSV function.

We will download data from the following link:
https://www.kaggle.com/datasets/billbasener/coronary-heart-disease?resource=download

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
cd /content/gdrive/MyDrive/Machine Learning/Colab Notebooks/ML_Practicals/1_Practical/Logistic regression P2
```

```
/content/gdrive/MyDrive/Machine Learning/Colab Notebooks/ML_Practicals/1_Practical/Logistic regression P2
```

```
ls
```

```
CHDdata.csv  CHD_Data.csv  CHDdata.gsheet  Logistic_Regression.ipynb
```

```
import pandas as pd
col_names = ['Systolic BP', 'Tobacco', 'low-density lipoprotein', 'Adiposity', 'Famhist', 'typea', 'Obesity', 'Alcohol', 'Age', 'Chd']
# load dataset
CHD = pd.read_csv("CHD_Data.csv", header=None, names=col_names)
```

Context

The data set CHDdata.csv contains cases of coronary heart disease (CHD) and variables associated with the patient's condition: systolic blood pressure, yearly tobacco use (in kg), low density lipoprotein (Idl), adiposity, family history (0 or 1), type A personality score (typea), obesity (body mass index), alcohol use, age, and the diagnosis of CHD (0 or 1).

```
CHD.head()

   Systolic BP  Tobacco  low-density lipoprotein  Adiposity  Famhist  typea
\
1          160    12.00                     5.73      23.11  Present     49
2          144     0.01                     4.41      28.61   Absent     55
3          118     0.08                     3.48      32.28  Present     52
4          170     7.50                     6.41      38.03  Present     51
5          134    13.60                     3.50      27.78  Present     60

   Obesity  Alcohol  Age  Chd
1    25.30    97.20   52    1
2    28.87     2.06   63    1
3    29.14     3.81   46    0
4    31.99    24.26   58    1
5    25.99    57.34   49    1

CHD.tail()

     Systolic BP  Tobacco  low-density lipoprotein  Adiposity  Famhist  typea
\
458          214      0.4                     5.98      31.72   Absent     64
459          182      4.2                     4.41      32.10   Absent     52
460          108      3.0                     1.59      15.23   Absent     40
461          118      5.4                    11.61      30.79   Absent     64
462          132      0.0                     4.82      33.41  Present     62

     Obesity  Alcohol  Age  Chd
458    28.45     0.00   58    0
459    28.61    18.72   52    1
460    20.09    26.64   55    0
461    27.35    23.97   40    0
462    14.70     0.00   46    1

CHD.drop('Famhist', inplace=True, axis=1)
```

Selecting Feature Here, we need to divide the given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

```
#split dataset in features and target variable
feature_cols = ['Systolic BP', 'Tobacco', 'low-density lipoprotein',
'Adiposity', 'typea', 'Obesity', 'Alcohol', 'Age', ]
X = CHD[feature_cols] # Features
y = CHD.Chd # Target variable
```

Splitting Data To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split dataset by using function train_test_split(). We need to pass 3 parameters features, target, and test_set size. Additionally, We can use random_state to select records randomly.

```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.65,random_stat
e=0)
```

Here, the Dataset is broken into two parts in a ratio of 75:25. It means 75% data will be used for model training and 25% for model testing.

Model Development and Prediction First, import the Logistic Regression module and create a Logistic Regression classifier object using LogisticRegression() function.

Then, fit our model on the train set using fit() and perform prediction on the test set using predict().

```
#import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Model Evaluation using Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. We can also visualize the performance of an algorithm. The fundamental of a confusion matrix is the number of correct and incorrect predictions are summed up class-wise.

```
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[174,  28],
       [ 56,  43]])
```

Here, we can see the confusion matrix in the form of the array object. The dimension of this matrix is 2*2 because this model is binary classification. We have two classes 0 and 1. Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions. In the output, 174 and 28 are actual predictions, and 56 and 43 are incorrect predictions.

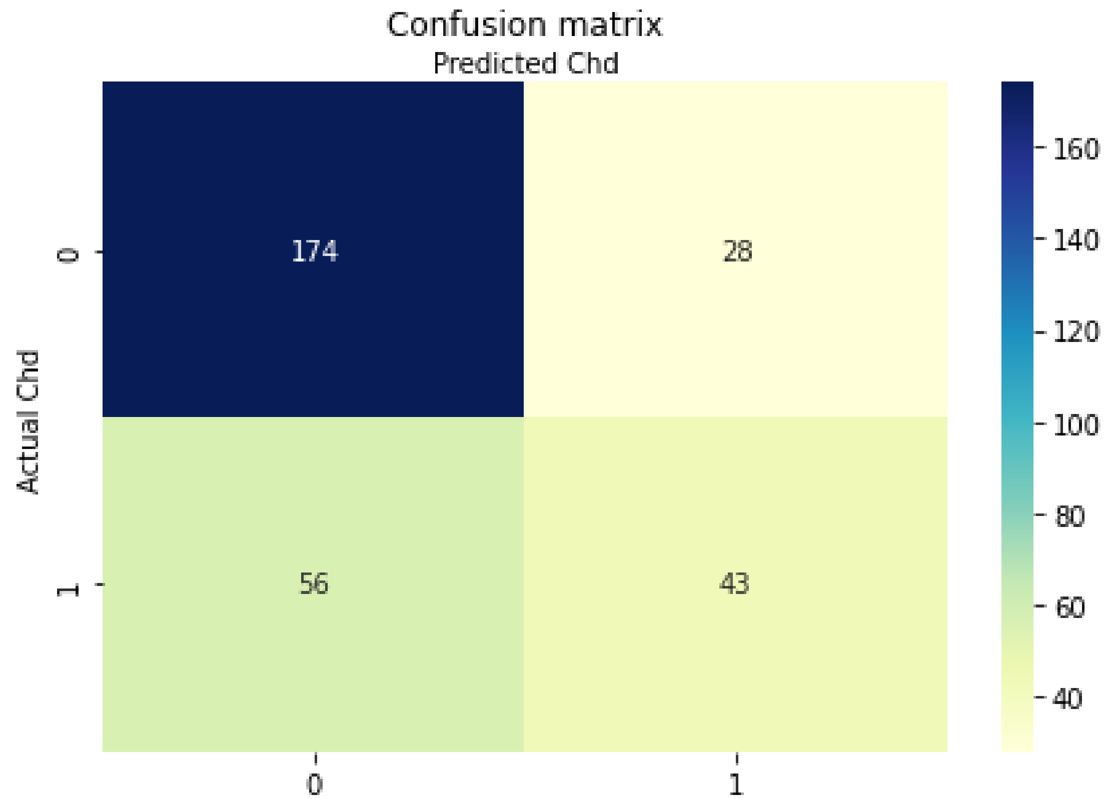
Visualizing Confusion Matrix using Heatmap Let's visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

Here, we will visualize the confusion matrix using Heatmap.

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual Chd')
plt.xlabel('Predicted Chd')
```

```
Text(0.5, 257.44, 'Predicted Chd')
```



Confusion Matrix Evaluation Metrics Let's evaluate the model using model evaluation metrics such as accuracy, precision, and recall.

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.7209302325581395
Precision: 0.6056338028169014
Recall: 0.43434343434343436
```

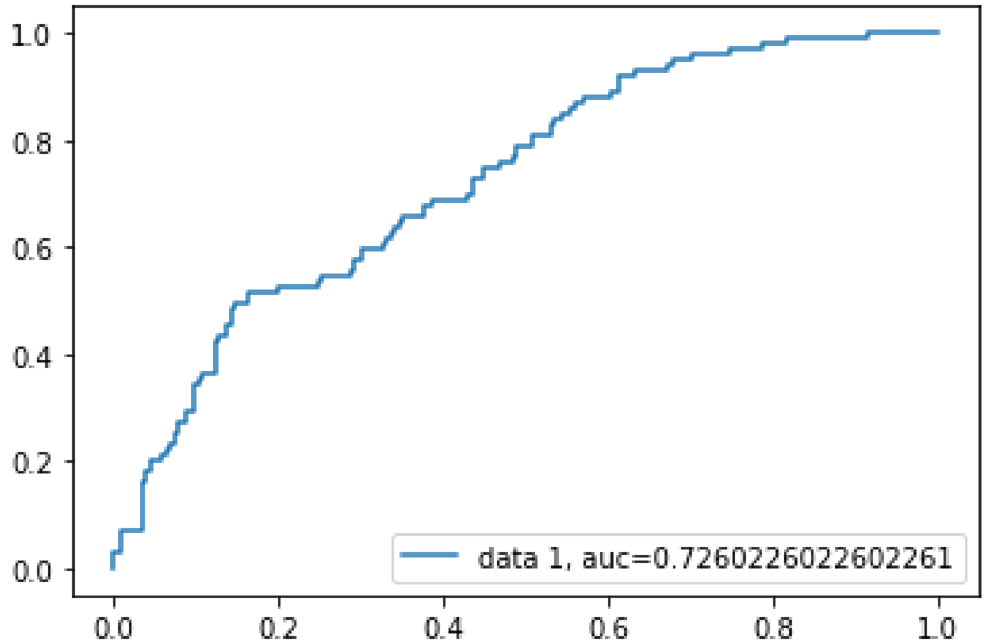
Well, we got a classification rate of 72%, considered as good accuracy.

Precision: Precision is about being precise, i.e., how accurate our model is. In other words, we can say, when a model makes a prediction, how often it is correct. In our prediction case, when our Logistic Regression model predicted patients are going to suffer from diabetes, that patients have 60% of the time.

Recall: If there are patients who have diabetes in the test set and our Logistic Regression model can identify it 43% of the time.

ROC Curve Receiver Operating Characteristic(ROC) curve is a plot of the true positive rate against the false positive rate. It shows the tradeoff between sensitivity and specificity.

```
y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



AUC score for the case is 0.73. AUC score 1 represents perfect classifier, and 0.5 represents a worthless classifier.

Conclusion In this Notebook we were able to measure and evaluate the Accuracy, Recall, Precision of the data thoroughly.

Experiment – 2

Aim - To observe the performance of dataset using Decision Tree Algorithm. Attribute Selection Measures Information Gain, Gain Ratio Gini index Optimizing Decision Tree Performance Classifier Building in Scikit-learn Pros and Cons Conclusion.

Here, we are going to predict coronary heart disease using Decision Tree Classifier.

Let's first load the required coronary heart disease dataset using the pandas' read CSV function.

We will download data from the following link: https://www.kaggle.com/datasets/billbasener/coronary-heart-disease?resource=download

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree
Classifier
from sklearn.model_selection import train_test_split # Import
train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy
calculation

from google.colab import drive

drive.mount('/content/gdrive', force_remount=True)

Mounted at /content/gdrive

cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P3_Decision Tree

/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P3_Decision Tree

ls

CHDdata.csv  CHD_Data.csv  CHDdata.gsheet  Decision_Tree.ipynb  diabetes.png

import pandas as pd
col_names = ['Systolic BP', 'Tobacco', 'low-density lipoprotein',
'Adiposity', 'Famhist', 'typea', 'Obesity', 'Alcohol', 'Age', 'Chd']
# load dataset
CHD = pd.read_csv("CHD_Data.csv", header=None, names=col_names)

CHD.head()

   Systolic BP  Tobacco  low-density lipoprotein  Adiposity  Famhist  typea
\
1          160    12.00                     5.73      23.11  Present     49
```

```
2          144     0.01                          4.41      28.61   Absent     55
3          118     0.08                          3.48      32.28  Present     52
4          170     7.50                          6.41      38.03  Present     51
5          134    13.60                          3.50      27.78  Present     60

   Obesity  Alcohol  Age  Chd
1    25.30    97.20   52    1
2    28.87     2.06   63    1
3    29.14     3.81   46    0
4    31.99    24.26   58    1
5    25.99    57.34   49    1
```

Feature Selection Here, we need to divide given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

```
CHD.drop('Famhist', inplace=True, axis=1)
```

```
CHD.head()
```

```
   Systolic BP  Tobacco  low-density lipoprotein  Adiposity  typea  Obesity  \
1          160    12.00                     5.73      23.11     49    25.30
2          144     0.01                     4.41      28.61     55    28.87
3          118     0.08                     3.48      32.28     52    29.14
4          170     7.50                     6.41      38.03     51    31.99
5          134    13.60                     3.50      27.78     60    25.99

   Alcohol  Age  Chd
1    97.20   52    1
2     2.06   63    1
3     3.81   46    0
4    24.26   58    1
5    57.34   49    1
```

```
#split dataset in features and target variable
feature_cols = ['Systolic BP', 'Tobacco', 'low-density lipoprotein',
'Adiposity', 'typea', 'Obesity', 'Alcohol', 'Age', ]
X = CHD[feature_cols] # Features
y = CHD.Chd # Target variable
```

Splitting Data To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split the dataset by using function train_test_split(). We need to pass 3 parameters features, target, and test_set size.

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=1) # 70% training and 30% test
```

Building Decision Tree Model Let's create a Decision Tree Model using Scikit-learn.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Evaluating Model Let's estimate, how accurately the classifier or model can predict the type of cultivars.

Accuracy can be computed by comparing actual test set values and predicted values.

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

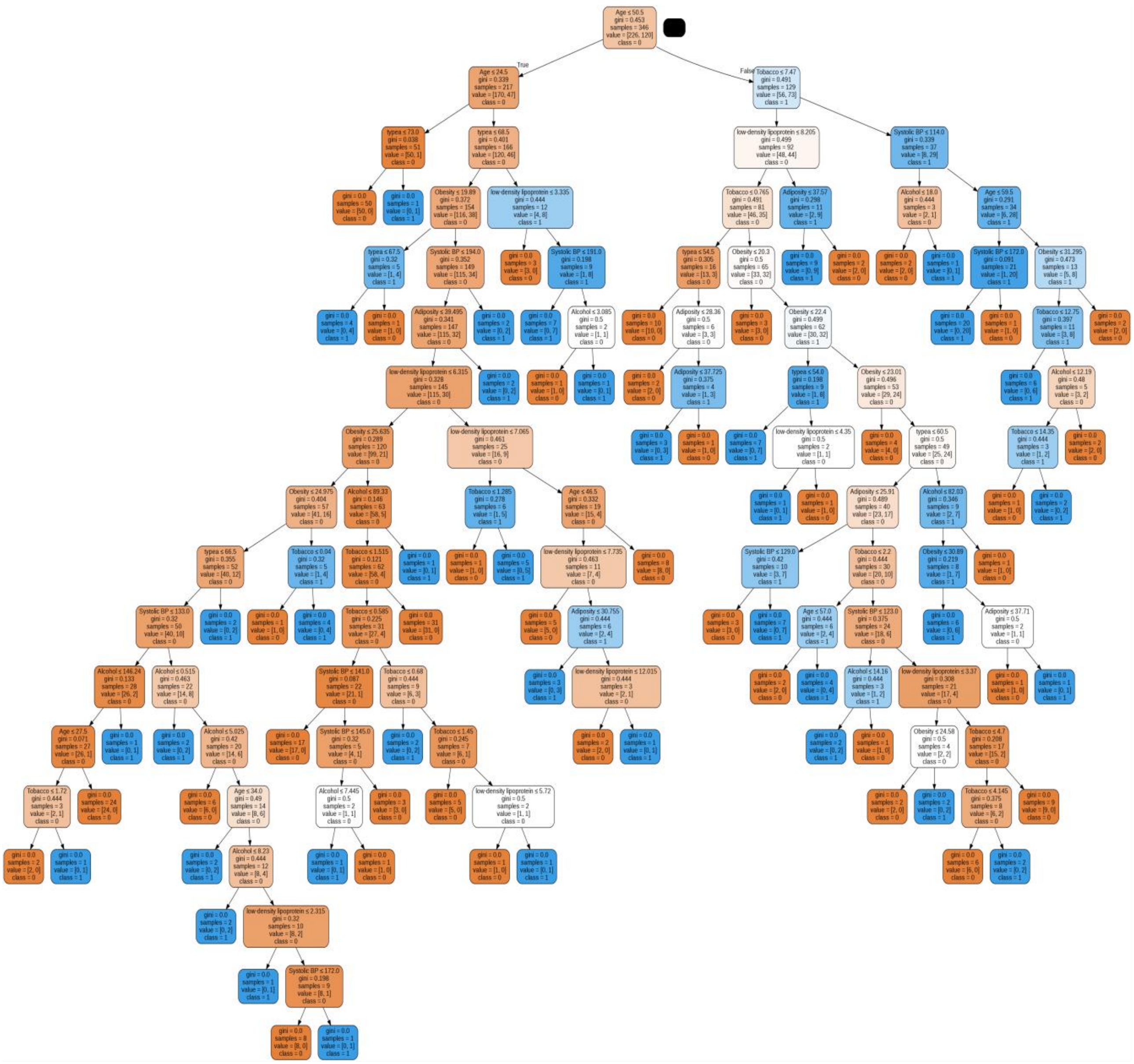
```
Accuracy: 0.6896551724137931
```

Well, We got a classification rate of 68%, considered as good accuracy. We can improve this accuracy by tuning the parameters in the Decision Tree Algorithm.

Indented block

Visualizing Decision Trees You can use Scikit-learn's export_graphviz function for display the tree within a Jupyter notebook. For plotting tree, We also need to install graphviz and pydotplus.

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names =
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



In the decision tree chart, each internal node has a decision rule that splits the data. Gini referred as Gini ratio, which measures the impurity of the node. You can say a node is pure when all of its records belong to the same class, such nodes known as the leaf node.

Here, the resultant tree is unpruned. This unpruned tree is unexplainable and not easy to understand. In the next section, let's optimize it by pruning.

Optimizing Decision Tree Performance criterion: optional (default="gini") or Choose attribute selection measure: This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.

splitter :string, optional (default=”best”) or Split Strategy: This parameter allows us to choose the split strategy. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

max_depth : int or None, optional (default=None) or Maximum Depth of a Tree: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

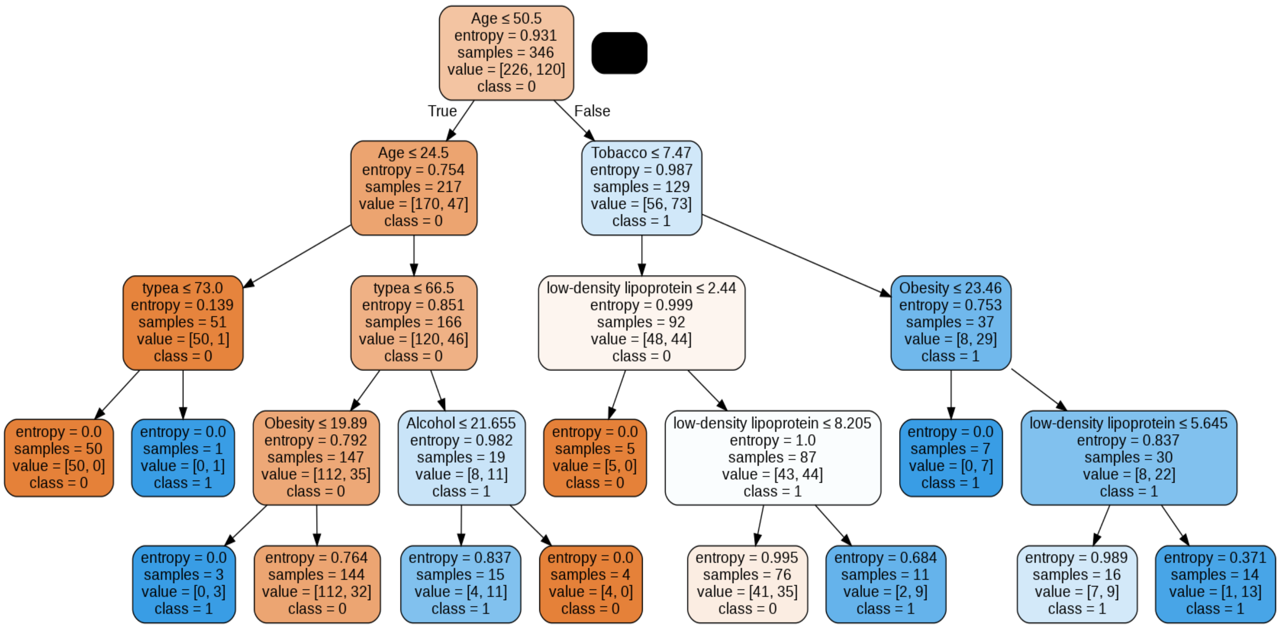
In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning. In the following the example, you can plot a decision tree on the same data with max_depth=3. Other than pre-pruning parameters, You can also try other attribute selection measure such as entropy.

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=4)


# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)


#Predict the response for test dataset
y_pred = clf.predict(X_test)


# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7068965517241379
```

```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names =
```

```
feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



Conclusion Finally we were able to observe the performance of dataset using Decision Tree Algorithm. Attribute Selection Measures Information Gain Ratio Gini index Optimizing Decision Tree Performance Classifier Building in Scikit-learn Pros and Cons Conclusion.

Experiment – 4

Aim – To Perform Object Classification using SVM and KNN.

Platform – Google colab

SVM – Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.

How it Works?

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane.

KNN – KNN is a non-parametric method used for classification. It is also one of the best-known classification algorithms. The principle is that known data are arranged in a space defined by the selected features.

How it Works?

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

About the Dataset – The dataset contains concrete images having cracks. The data is collected from various METU Campus Buildings.

The dataset is divided into two as negative and positive crack images for image classification.

Each class has 20000images with a total of 40000 images with 227 x 227 pixels with RGB channels.

The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by Zhang et al (2016).

High-resolution images have variance in terms of surface finish and illumination conditions.

No data augmentation in terms of random rotation or flipping is applied.

Dataset Link - https://data.mendeley.com/datasets/5y9wdsg2zt/2

Generating Data –

```
import pandas as pd
import os
import glob
import numpy
import cv2

from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P4_SVM/New Data

/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P4_SVM/New Data

imagePaths = []

# input images
for img in glob.glob("Data/*.jpg"):  # folder train1 contains multiple dog
and cat images in .jpg
    imagePaths = list(glob.glob("Data/*.jpg"))

# Extract the image into vector
def image_vector(image, size=(128, 128)):
    return cv2.resize(image, size).flatten()

# initialize the pixel intensities matrix, labels list
imagematrix = []
imagelabels = []
pixels = None

# Build image vector matrix
for (i, path) in enumerate(imagePaths):
    # load the image and extract the class label, image intensities
    image = cv2.imread(path)
    label = path.split(os.path.sep)[-1].split(".")[0]
    pixels = image_vector(image)

    # update the images and labels matricies respectively
    imagematrix.append(pixels)
    imagelabels.append(label)

imagematrix = numpy.array(imagematrix)
imagelabels = numpy.array(imagelabels)

# save numpy arrays for future use
numpy.save("matrix.npy", imagematrix)
numpy.save("labels.npy", imagelabels)
```

Testing of Trained Classifier Model –

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import numpy
import cv2
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount,
call drive.mount("/content/gdrive", force_remount=True).
```

```
cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P4_SVM/New Data
```

```
/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P4_SVM/New Data
```

```
ls
```

```
case1.jpg  case2.jpg  case3.jpg  case4.jpg  Data/  labels.npy  matrix.npy
```

```
# Extract the image into vector
def image_vector(image, size=(128, 128)):
    return cv2.resize(image, size).flatten()

imagematrix = numpy.load("matrix.npy")
imagelabels = numpy.load("labels.npy")

# Prepare data for training and testing
(train_img, test_img, train_label, test_label) =
train_test_split(imagematrix, imagelabels, test_size=0.1, random_state=50)

'''SVM MODEL IN SKLEARN'''
model1 = SVC(max_iter=-1, kernel='linear',
class_weight='balanced',gamma='scale')  # kernel linear is better Gausian
kernel here
model1.fit(train_img, train_label)
acc1 = model1.score(test_img, test_label)
print("SVM model accuracy: {:.2f}%".format(acc1 * 100))
```

```
SVM model accuracy: 52.50%
```

```
'''KNN MODEL IN SKLEARN'''
model2 = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
model2.fit(train_img, train_label)
```

```
acc2 = model2.score(test_img, test_label)
print("KNN model accuracy: {:.2f}%".format(acc2 * 100))
```

```
KNN model accuracy: 49.64%
```

```
'''PREDICATION SAMPLE'''
for t in range(1,5):
  pixel = image_vector(cv2.imread("case{0}.jpg".format(t)))
  rawImage = numpy.array([pixel])
  prediction1 = model1.predict(rawImage)
  prediction2 = model2.predict(rawImage)
  print("Test Case {0}".format(t))
  print("Prediction by SVM - {0}".format(prediction1[0]))
  print("Prediction by KNN - {0}".format(prediction1[0]))
```

```
Test Case 1
Prediction by SVM - Crack (307)
Prediction by KNN - Crack (307)
Test Case 2
Prediction by SVM - Not_Crack (724)
Prediction by KNN - Not_Crack (724)
Test Case 3
Prediction by SVM - Crack (312)
Prediction by KNN - Crack (312)
Test Case 4
Prediction by SVM - Not_Crack (725)
Prediction by KNN - Not_Crack (725)
```

Prediction of Test Case files using Trained Algorithm –

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from google.colab import files
from google.colab.patches import cv2_imshow


import pandas as pd
import numpy
import cv2
import os
import glob
```

```
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving case1.jpg to case1.jpg
Saving case2.jpg to case2.jpg
```

```
Saving case3.jpg to case3.jpg
Saving case4.jpg to case4.jpg
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount,
call drive.mount("/content/gdrive", force_remount=True).
```

```
cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P4_SVM/New Data
```

```
/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P4_SVM/New Data
```

```
ls
```

```
case1.jpg  case2.jpg  case3.jpg  case4.jpg  Data/  labels.npy  matrix.npy
```

```
imagematrix = numpy.load("matrix.npy")
imagelabels = numpy.load("labels.npy")
(train_img, test_img, train_label, test_label) =
train_test_split(imagematrix, imagelabels, test_size=0.2, random_state=50)
```

```
model1 = SVC(max_iter=-1, kernel='linear',
class_weight='balanced',gamma='scale')  # kernel linear is better Gausian
kernel here
model1.fit(train_img, train_label)
acc1 = model1.score(test_img, test_label)
print("SVM model accuracy: {:.2f}%".format(acc1 * 100))
```

```
SVM model accuracy: 50.50%
```

```
model2 = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
model2.fit(train_img, train_label)
acc2 = model2.score(test_img, test_label)
print("KNN model accuracy: {:.2f}%".format(acc2 * 100))
```

```
KNN model accuracy: 49.64.64%
```

```
# Extract the image into vector
def image_vector(image, size=(128, 128)):
    return cv2.resize(image, size).flatten()


for t in range(1,5):
  img = cv2.imread("case{0}.jpg".format(t))
  pixel = image_vector(img)
  rawImage = numpy.array([pixel])
  prediction1 = model1.predict(rawImage)
  prediction2 = model2.predict(rawImage)
```

```
print("Test Case {0}".format(t))
print("Prediction by SVM - {0}".format(prediction1[0]))
print("Prediction by KNN - {0}".format(prediction1[0]))
w, h = len(img[0]), len(img)
if w>1000:
  w, h = w//4, h//4
else:
  w, h = w//2, h//2
cv2_imshow(cv2.resize(img,(w,h)))
```

```
Test Case 1
Prediction by SVM - Crack (307)
Prediction by KNN - Crack (307)
```

```
Test Case 2
Prediction by SVM - Not_Crack (724)
Prediction by KNN - Not_Crack (724)
```

```
Test Case 3
Prediction by SVM - Crack (312)
Prediction by KNN - Crack (312)
```

```
Test Case 4
Prediction by SVM - Not_Crack (725)
Prediction by KNN - Not_Crack (725)
```

Result –

Hence, We were able to Predict the class of the Cases given as the input to the algorithm.

Experiment – 5

Aim – To perform K-means Algorithm on Customer Data.

Platform – Google Colab

Clustering – Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

KMeans Clustering – K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. You'll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.

About the dataset – This input file contains the basic information (ID, age, gender, income, spending score) about the customers of a mall. Spending Score is something you assign to the customer based on your defined parameters like customer behavior and purchasing data.

K Means Clustering for Customer Data –

```
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P5_Kmeans
```

```
/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P5_Kmeans
```

```
ls
```

```
segmented_customers_new.csv  Untitled0.ipynb
```

```
data = pd.read_csv("segmented_customers_new.csv", index_col="id")
data.head()
```

```
    Annual Income (k$)  Spending Score (1-100)
id
1                   15                      39
2                   15                      81
3                   16                       6
```

```
4                    16                        77
5                    17                        40
```

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4)

kmeans.fit(data)
```

```
KMeans(n_clusters=4)
```

```
kmeans.cluster_centers_
```

```
array([[26.30434783, 20.91304348],
       [86.53846154, 82.12820513],
       [48.26      , 56.48      ],
       [87.        , 18.63157895]])
```

```
kmeans.labels_
```

```
array([0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1], dtype=int32)
```

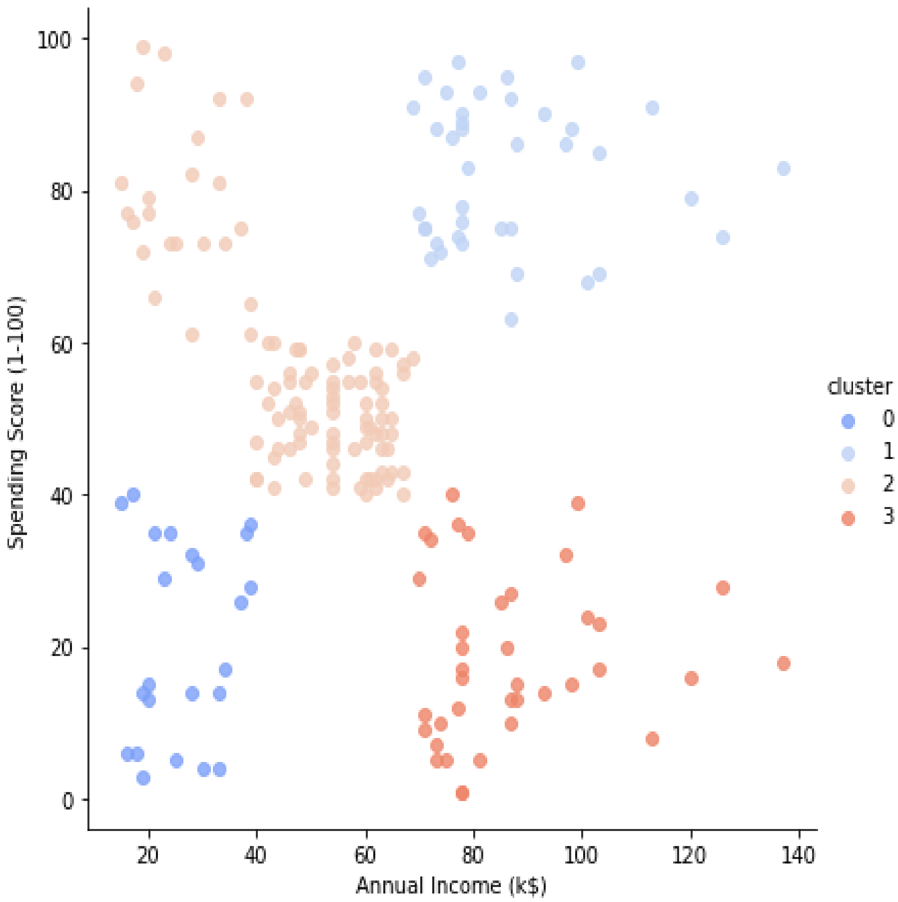
```
import numpy as np

unique, counts = np.unique(kmeans.labels_, return_counts=True)

dict_data = dict(zip(unique, counts))
dict_data
```

```
{0: 23, 1: 39, 2: 100, 3: 38}
```

```
import seaborn as sns
data["cluster"] = kmeans.labels_
sns.lmplot('Annual Income (k$)', 'Spending Score (1-100)', data=data,
hue='cluster', palette='coolwarm', height=6, aspect=1, fit_reg=False)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variables as keyword args: x, y. From
version 0.12, the only valid positional argument will be `data`, and passing
other arguments without an explicit keyword will result in an error or
misinterpretation.
  FutureWarning
```

```
<seaborn.axisgrid.FacetGrid at 0x7fecb3e2a310>
```




```
# Inertia is the sum of squared error for each cluster.

# Therefore, the smaller the inertia the denser the cluster(closer together all the points are)


kmeans.inertia_

73679.78903948836

kmeans.score

<bound method KMeans.score of KMeans(n_clusters=4)>
```

```
data

     Annual Income (k$)  Spending Score (1-100)  cluster
id
1                    15                      39        0
2                    15                      81        2
3                    16                       6        0
4                    16                      77        2
5                    17                      40        0
..                  ...                     ...      ...
196                 120                      79        1
197                 126                      28        3
198                 126                      74        1
199                 137                      18        3
200                 137                      83        1

[200 rows x 3 columns]
```

Result – Thus, we have analyzed Customer data and performed 2D clustering using K Means Algorithm. This kind of cluster analysis helps design better customer acquisition strategies and helps in business growth.

Experiment – 6

Aim – To Perform KNN (K-Nearest Neighbors) Algorithm on the Haberman's Survival Dataset.

Platform – Google Colab

What is KNN ?

K-Nearest Neighbors (KNN)

KNN is a non-parametric method used for classification. It is also one of the best-known classification algorithms. The principle is that known data are arranged in a space defined by the selected features.

How does it work?

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression)

Dataset Link - https://archive.ics.uci.edu/ml/datasets/haberman's+survival

Haberman's Survival Data Set

Data Set Information:

The dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer.

Attribute Information:

1 - Age of patient at time of operation (numerical)

2 - Number of positive axillary nodes detected (numerical)

3 - Survival status (class attribute)

0= the patient died within 5 years

1= the patient survived 5 years or longer

Importing the Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
from google.colab import drive

drive.mount('/content/gdrive')

Mounted at /content/gdrive

cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P6_KNN

/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P6_KNN

ls

haberman.csv  KNN.ipynb  Surgical_deepnet.csv
```

Importing the Dataset

```
dataset = pd.read_csv('haberman.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

dataset

     Age  Axillry Nodes(+ve)  Survival
0     30                  1         1
1     30                  3         1
2     30                  0         1
3     31                  2         1
4     31                  4         1
..   ...                ...       ...
301   75                  1         1
302   76                  0         1
303   77                  3         1
304   78                  1         0
305   83                  2         0

[306 rows x 3 columns]
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
random_state = 0)
```

```
print(X_train)
```

```
[[41  0]
 [70  4]
 [62  0]
 [55  3]
 [38  0]
 [52  0]
  ... ...

 [56  0]
 [63  9]
 [56  0]
 [49  0]
 [41  8]
 [54  3]]
```

```
print(y_train)
```

```
[1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 0 1 0 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1 0 0 0 1 1 0 1 1 1 1 1 0 1 1 1 1 0
 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1 1
 1 1 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1]
```

```
print(X_test)
```

```
[[67  1]
 [43 14]
 [65  0]
 [58  1]
 [53  9]
 [37 15]
 [60  0]

  ... ...

 [74  3]
 [70  8]
 [47  3]
 [62 19]
 [38  1]
 [48 11]
 [37  0]]
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test) #avoid data leakage
```

```
print(X_train)
```

```
[[-1.09229448 -0.53827882]
 [ 1.61641636 -0.00810312]
 [ 0.86918578 -0.53827882]
 [ 0.21535903 -0.14064705]
 [-1.37250594 -0.53827882]
 [-0.06485244 -0.53827882]
 [ 1.14939725 -0.53827882]
 [-1.27910212 -0.53827882]
 [-0.25166008 -0.40573489]
 [ 0.12195521  0.38952865]
 [ 0.77578196  0.52207257]
 [-0.06485244  0.1244408 ]
  .............  ............

 [ 1.70982018 -0.27319097]
 [ 0.86918578 -0.53827882]
 [ 1.33620489 -0.53827882]
 [ 0.4955705  -0.53827882]
 [-1.74612123  3.43803889]
 [ 0.30876285 -0.53827882]
 [ 0.96258961  0.6546165 ]
 [ 0.30876285 -0.53827882]
 [-0.3450639  -0.53827882]
 [-1.09229448  0.52207257]
 [ 0.12195521 -0.14064705]]
```

```
print(X_test.dtype)
```

```
float64
```

Training the K-NN model on the Training set

```
from math import sqrt
class KNN():
  def __init__(self,k):
    self.k=k
    print(self.k)
  def fit(self,X_train,y_train):
    self.x_train=X_train
    self.y_train=y_train
  def calculate_euclidean(self,sample1,sample2):
    distance=0.0
```

```
    for i in range(len(sample1)):
      distance+=(sample1[i]-sample2[i])**2 #Euclidean Distance = sqrt(sum i
to N (x1_i – x2_i)^2)
    return sqrt(distance)
  def nearest_neighbors(self,test_sample):
    distances=[]#calculate distances from a test sample to every sample in a
training set
    for i in range(len(self.x_train)):

distances.append((self.y_train[i],self.calculate_euclidean(self.x_train[i],te
st_sample)))
    distances.sort(key=lambda x:x[1])#sort in ascending order, based on a
distance value
    neighbors=[]
    for i in range(self.k): #get first k samples
      neighbors.append(distances[i][0])
    return neighbors
  def predict(self,test_set):
    predictions=[]
    for test_sample in test_set:
      neighbors=self.nearest_neighbors(test_sample)
      labels=[sample for sample in neighbors]
      prediction=max(labels,key=labels.count)
      predictions.append(prediction)
    return predictions

model=KNN(5) #our model
model.fit(X_train,y_train)

5

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
2)#The default metric is minkowski, and with p=2 is equivalent to the
standard Euclidean metric.
classifier.fit(X_train, y_train)

KNeighborsClassifier()
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
predictions=model.predict(X_test)#our model's predictions
```

Making the Confusion Matrix to compare both models

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

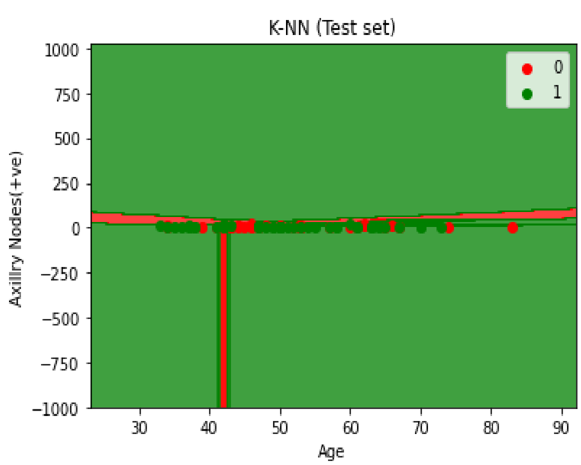
```
[[ 8 23]
 [ 5 41]]

0.6363636363636364
```

```
cm = confusion_matrix(y_test, predictions) #our model
print(cm)
accuracy_score(y_test, predictions)
```

```
[[ 8 23]
 [ 3 43]]

0.6623376623376623
```

Visualising the Test set results

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop =
X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop =
X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Axillry Nodes(+ve)')
plt.legend()
plt.show()
```



Conclusion - Patient's age and operation year alone are not deciding factors survival. People less than 35 years have more chance of survival.

Result – Hence, I have successfully performed KNN Algorithm over Haberman's Survival Dataset.

Experiment – 7

Aim – To Perform Naïve Bayes Algorithm on Weather Dataset.

Platform – Google Colab

What is Naïve Byes Algorithm?

A naive Bayes classifier is an algorithm that uses Bayes' theorem to classify objects. Naive Bayes classifiers assume strong, or naive, independence between attributes of data points. Popular uses of naive Bayes classifiers include spam filters, text analysis and medical diagnosis.

About Dataset – This Dataset is about weather condition for play time.

Attributes in dataset –

1 – Outlook - {Sunny, Rainy, Outcast}

2 – Temp – {Hot, Cold, Mild}

3 – Humidity – {High, Normal, High}

4 – Windy – {f}

5 – Play – {Yes, No}

Code –

Importing Libraries

```
from functools import reduce
import pandas as pd
import pprint
```

Importing Dataset

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P7_Naive Bayes
```

```
/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P7_Naive Bayes
```

```
ls
```

```
bayes.py  LICENSE  new_dataset.csv  README.md  Untitled0.ipynb
```

```
#Reading CSV files
df=pd.read_csv('new_dataset.csv')
df
```

	Outlook	Temp	Humidity	Windy	Play
0	Rainy	Hot	High	f	no
1	Rainy	Hot	High	t	no
2	Overcast	Hot	High	f	yes
3	Sunny	Mild	High	f	yes
4	Sunny	Cool	Normal	f	yes
5	Sunny	Cool	Normal	t	no
6	Overcast	Cool	Normal	t	yes
7	Rainy	Mild	High	f	no
8	Rainy	Cool	Normal	f	yes
9	Sunny	Mild	Normal	f	yes
10	Rainy	Mild	Normal	t	yes
11	Overcast	Mild	High	t	yes
12	Overcast	Hot	Normal	f	yes
13	Sunny	Mild	High	t	no

Implementation of Naive Bayes Algorithm

```
class Classifier():
    data = None
    class_attr = None
    priori = {}
    cp = {}
    hypothesis = None


    def __init__(self,filename=None, class_attr=None ):
        self.data = pd.read_csv(filename, sep=',', header =(0))
        self.class_attr = class_attr

    '''
        probability(class) =    How many  times it appears in cloumn
                             __________________________________________
                                  count of all class attribute
    '''
    def calculate_priori(self):
        class_values = list(set(self.data[self.class_attr]))
        class_data =  list(self.data[self.class_attr])
        for i in class_values:
            self.priori[i]  = class_data.count(i)/float(len(class_data))
        print ("Priori Values: ", self.priori)

    '''
        Here we calculate the individual probabilites
        P(outcome|evidence) =   P(Likelihood of Evidence) x Prior prob of
```

```
outcome
                               ____________________________________________
                                                    P(Evidence)
    '''
    def get_cp(self, attr, attr_type, class_value):
        data_attr = list(self.data[attr])
        class_data = list(self.data[self.class_attr])
        total =1
        for i in range(0, len(data_attr)):
            if class_data[i] == class_value and data_attr[i] == attr_type:
                total+=1
        return total/float(class_data.count(class_value))

    '''
        Here we calculate Likelihood of Evidence and multiple all individual
probabilities with priori
        (Outcome|Multiple Evidence) = P(Evidence1|Outcome) x
P(Evidence2|outcome) x ... x P(EvidenceN|outcome) x P(Outcome)
        scaled by P(Multiple Evidence)
    '''
    def calculate_conditional_probabilities(self, hypothesis):
        for i in self.priori:
            self.cp[i] = {}
            for j in hypothesis:
                self.cp[i].update({ hypothesis[j]: self.get_cp(j,
hypothesis[j], i)})
        print ("\nCalculated Conditional Probabilities: \n")
        pprint.pprint(self.cp)

    def classify(self):
        print ("Result: ")
        for i in self.cp:
            print (i, " ==> ", reduce(lambda x, y: x*y,
self.cp[i].values())*self.priori[i])


if __name__ == "__main__":
    c = Classifier(filename="new_dataset.csv", class_attr="Play" )
    c.calculate_priori()
    c.hypothesis = {"Outlook":'Rainy', "Temp":"Mild", "Humidity":'Normal' ,
"Windy":'t'}

    c.calculate_conditional_probabilities(c.hypothesis)
    c.classify()
```

```
Priori Values:  {'yes': 0.6428571428571429, 'no': 0.35714285714285715}

Calculated Conditional Probabilities:

{'no': {'Mild': 0.6, 'Normal': 0.4, 'Rainy': 0.8, 't': 0.8},
 'yes': {'Mild': 0.5555555555555556,
         'Normal': 0.7777777777777778,
         'Rainy': 0.3333333333333333,
         't': 0.4444444444444444}}
Result:
yes  ==>  0.04115226337448559
no  ==>  0.05485714285714286
```

Result -

Hence, according to the output Calculated Probability of Yes is Greater than No.

Experiment – 8

Aim – To perform Random Forest Algorithm on the image Dataset of two Class.

Platform – Google Colab

Random Forest?

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

About Dataset –

The dataset contains concrete images having cracks. The data is collected from various METU Campus Buildings. The dataset is divided into two as negative and positive crack images for image classification. Each class has 20000images with a total of 40000 images with 227 x 227 pixels with RGB channels. The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by Zhang et al (2016). High-resolution images have variance in terms of surface finish and illumination conditions. No data augmentation in terms of random rotation or flipping is applied.

Dataset Link – https://data.mendeley.com/datasets/5y9wdsg2zt/2

```
pip install mahotas
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting mahotas
  Downloading mahotas-1.4.13-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (5.7 MB)
ent already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from mahotas) (1.21.6)
Installing collected packages: mahotas
Successfully installed mahotas-1.4.13
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import mahotas
import cv2
import os
import h5py
import glob
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier


# make a fix file size
fixed_size  = tuple((500,500))

#train path
train_path = "dataset/train"

# no of trees for Random Forests
num_tree = 100

# bins for histograms
bins = 8

# train_test_split size
test_size = 0.10

# seed for reproducing same result
seed = 9

# features description -1:  Hu Moments

def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor -2 Haralick Texture

def fd_haralick(image):
    # conver the image to grayscale
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    # Ccompute the haralick texture fetature ve tor
    haralic = mahotas.features.haralick(gray).mean(axis=0)
    return haralic

# feature-description -3 Color Histogram

def fd_histogram(image, mask=None):
    # conver the image to HSV colors-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    #COPUTE THE COLOR HISTPGRAM
    hist  = cv2.calcHist([image],[0,1,2],None,[bins,bins,bins], [0, 256, 0,
256, 0, 256])
    # normalize the histogram
```

```
    cv2.normalize(hist,hist)
    # return the histog....
    return hist.flatten()

from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

cd /content/gdrive/MyDrive/Colab Notebooks/Random Forest

/content/gdrive/MyDrive/Colab Notebooks/Random Forest

# get the training data labels
train_labels = os.listdir(train_path)

# sort the training labesl
train_labels.sort()
print(train_labels)

# empty list to hold feature vectors and labels
global_features = []
labels = []

i, j = 0, 0
k = 0

# num of images per class
images_per_class = 80

['Crack', 'Not Crack']

# ittirate the folder to get the image label name

%time
# lop over the training data sub folder

for training_name in train_labels:
    # join the training data path and each species training folder
    dir = os.path.join(train_path, training_name)

    # get the current training label
    current_label = training_name

    k = 1
    # loop over the images in each sub-folder

    for file in os.listdir(dir):
```

```
        file = dir + "/" + os.fsdecode(file)

        # read the image and resize it to a fixed-size
        image = cv2.imread(file)

        if image is not None:
            image = cv2.resize(image,fixed_size)
            fv_hu_moments = fd_hu_moments(image)
            fv_haralick   = fd_haralick(image)
            fv_histogram  = fd_histogram(image)
        #else:
            #print("image not loaded")

        #image = cv2.imread(file)
        #image = cv2.resize(image,fixed_size)

        # Concatenate global features
        global_feature = np.hstack([fv_histogram, fv_haralick,
fv_hu_moments])

        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)

        i += 1
        k += 1
    print("[STATUS] processed folder: {}".format(current_label))
    j += 1

print("[STATUS] completed Global Feature Extraction...")
```

```
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 8.58 µs
[STATUS] processed folder: Crack
[STATUS] processed folder: Not Crack
[STATUS] completed Global Feature Extraction...
```

```
%time
import h5py
# get the overall feature vector size
print("[STATUS] feature vector size
{}".format(np.array(global_features).shape))

# get the overall training label size
print("[STATUS] training Labels {}".format(np.array(labels).shape))

# encode the target labels
targetNames = np.unique(labels)
```

```
le = LabelEncoder()
target = le.fit_transform(labels)
print("[STATUS] training labels encoded...{}")
# normalize the feature vector in the range (0-1)
scaler = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)
print("[STATUS] feature vector normalized...")

print("[STATUS] target labels: {}".format(target))
print("[STATUS] target labels shape: {}".format(target.shape))

# save the feature vector using HDF5
h5f_data = h5py.File('output/data.h5', 'w')
h5f_data.create_dataset('dataset_1', data=np.array(rescaled_features))

h5f_label = h5py.File('output/labels.h5', 'w')
h5f_label.create_dataset('dataset_1', data=np.array(target))

h5f_data.close()
h5f_label.close()

print("[STATUS] end of training..")
```

```
CPU times: user 2 µs, sys: 2 µs, total: 4 µs
Wall time: 6.2 µs
[STATUS] feature vector size (200, 532)
[STATUS] training Labels (200,)
[STATUS] training labels encoded...{}
[STATUS] feature vector normalized...
[STATUS] target labels: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[STATUS] target labels shape: (200,)
[STATUS] end of training..
```

```
# import the feature vector and trained labels

h5f_data = h5py.File('output/data.h5', 'r')
h5f_label = h5py.File('output/labels.h5', 'r')

global_features_string = h5f_data['dataset_1']
global_labels_string = h5f_label['dataset_1']
```

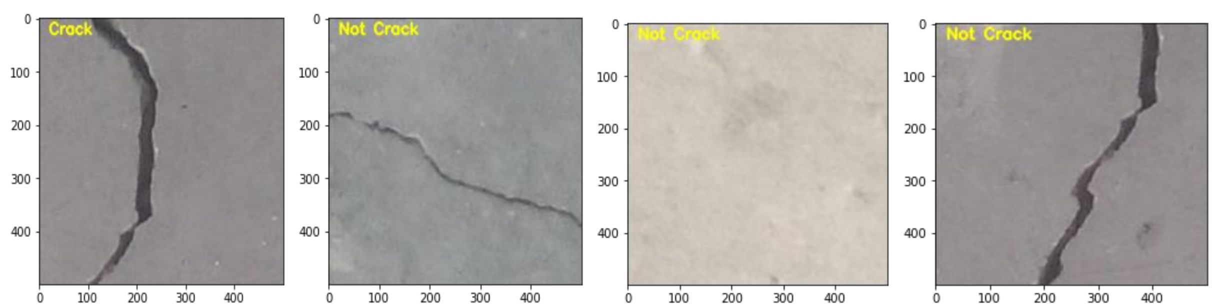
```
global_features = np.array(global_features_string)
global_labels = np.array(global_labels_string)

# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) =
train_test_split(np.array(global_features),

np.array(global_labels),

test_size=test_size,

random_state=seed)

from sklearn.metrics import classification_report
# create the model - Random Forests
clf  = RandomForestClassifier(n_estimators=100)
from sklearn.ensemble import AdaBoostClassifier


# fit the training data to the model
clf.fit(trainDataGlobal, trainLabelsGlobal)

#print(clf.fit(trainDataGlobal, trainLabelsGlobal))

clf_pred = clf.predict(trainDataGlobal)
#clf_pred = clf.predict(global_feature.reshape(1,-1))[0]
print(classification_report(trainLabelsGlobal,clf_pred))
#print(confusion_matrix(trainLabelsGlobal,clf_pred))

#print(clf.predict(trainDataGlobal))

#print(clf.predict(global_feature.reshape(1,-1))[0])
```

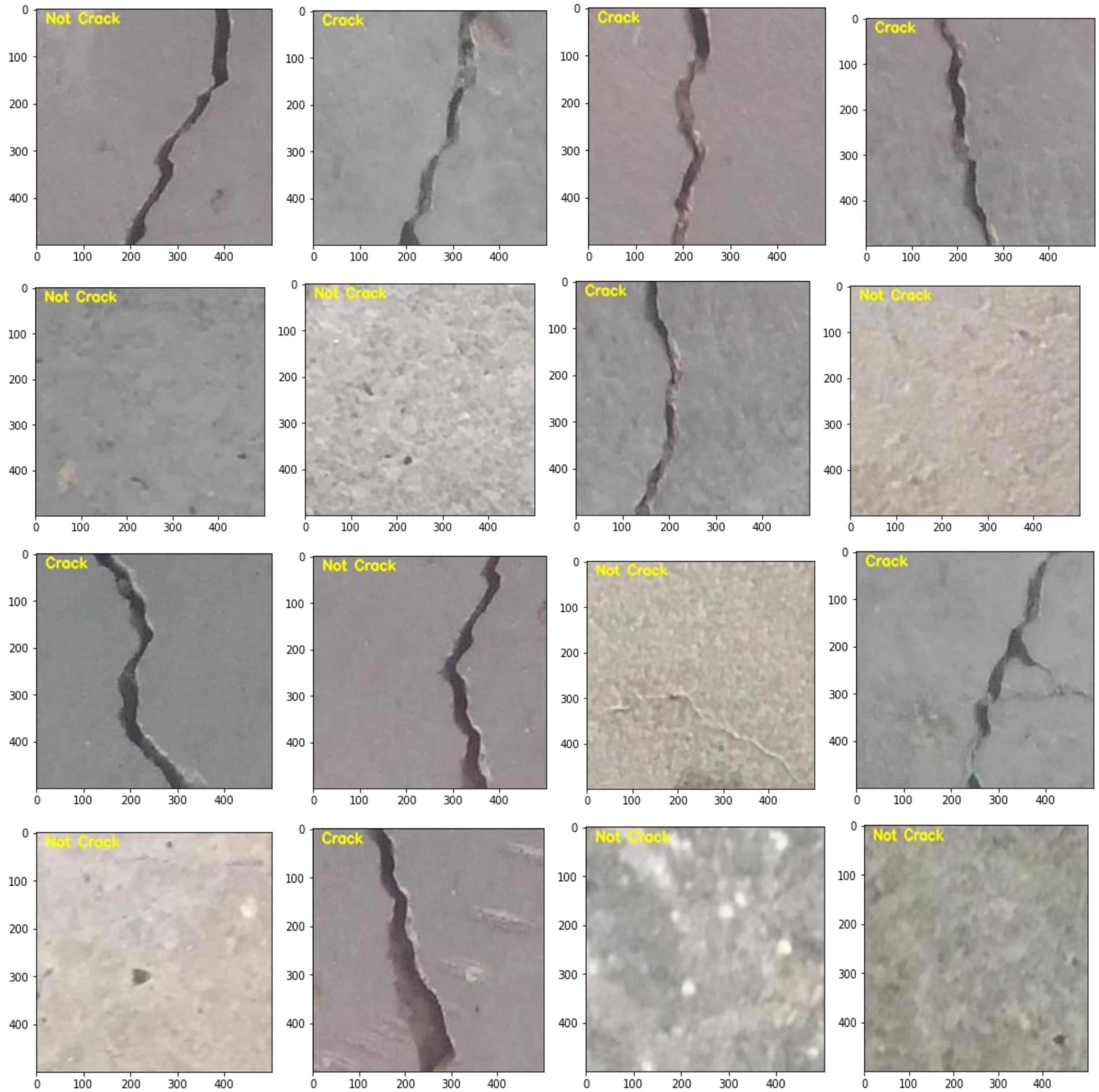
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        91
           1       1.00      1.00      1.00        89

    accuracy                           1.00       180
   macro avg       1.00      1.00      1.00       180
weighted avg       1.00      1.00      1.00       180
```

```
# path to test data
test_path = "dataset/test"

# loop through the test images
#for file in glob.glob(test_path + "/*.jpg"):
```

```
for file in os.listdir(test_path):

    file = test_path + "/" + file
    #print(file)

    # read the image
    image = cv2.imread(file)

    # resize the image
    image = cv2.resize(image, fixed_size)

    # Global Feature extraction
    fv_hu_moments = fd_hu_moments(image)
    fv_haralick   = fd_haralick(image)
    fv_histogram  = fd_histogram(image)

    # Concatenate global features

    global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])

    # predict label of test image
    prediction = clf.predict(global_feature.reshape(1,-1))[0]

    # show predicted label on image
    cv2.putText(image, train_labels[prediction], (20,30),
cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,255), 3)

    # display the output image
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.show()
```

Output hidden; open in https://colab. research.google.com to view.






```
from mlxtend.evaluate import confusion_matrix

y_target =    [1, 2, 3,4, 5, 6, 7, 8,9,10]
y_predicted = [0, 2, 3, 4, 5,6, 7, 8,9,0]

cm = confusion_matrix(y_target=y_target,
                      y_predicted=y_predicted,
                      binary=False)
cm
```

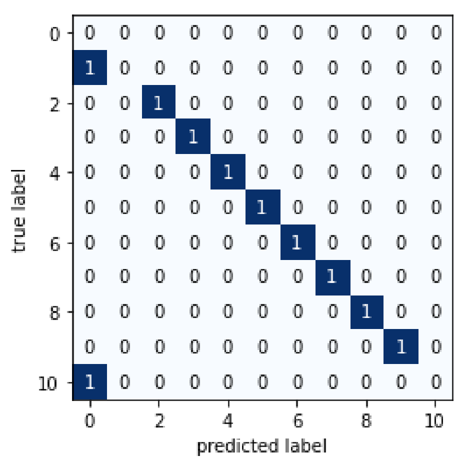
```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
pip install mlxtend
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.18->mlxtend)
(3.1.0)
```

```
from mlxtend.plotting import plot_confusion_matrix
fig, ax = plot_confusion_matrix(conf_mat=cm)
plt.show()
```

```
/usr/local/lib/python3.7/dist-
packages/mlxtend/plotting/plot_confusion_matrix.py:59: RuntimeWarning:
invalid value encountered in true_divide
  normed_conf_mat = conf_mat.astype('float') / total_samples
```

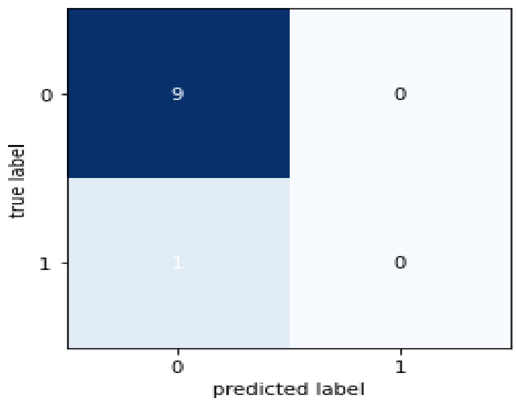



```
from mlxtend.evaluate import confusion_matrix


y_target =    [1, 2, 3,4, 5, 6, 7, 8,9,10]
y_predicted = [0, 2, 3, 4, 5,6, 7, 8,9,0]
```

```
cm = confusion_matrix(y_target=y_target,
                      y_predicted=y_predicted,
                      binary=True,
                      positive_label=1)
cm
```

```
array([[9, 0],
       [1, 0]])
```

```
from mlxtend.plotting import plot_confusion_matrix

fig, ax = plot_confusion_matrix(conf_mat=cm)
plt.show()
```




```
def precision(label, confusion_matrix):
    col = confusion_matrix[:, label]
    return confusion_matrix[label, label] / col.sum()

def recall(label, confusion_matrix):
    row = confusion_matrix[label, :]
    return confusion_matrix[label, label] / row.sum()
def precision_macro_average(confusion_matrix):
    rows, columns = confusion_matrix.shape
    sum_of_precisions = 0
    for label in range(rows):
        sum_of_precisions += precision(label, confusion_matrix)
    return sum_of_precisions / rows
def recall_macro_average(confusion_matrix):
    rows, columns = confusion_matrix.shape
    sum_of_recalls = 0
    for label in range(columns):
        sum_of_recalls += recall(label, confusion_matrix)
    return sum_of_recalls / columns
```

```
print("precision total:", precision_macro_average(cm))
print("recall total:", recall_macro_average(cm))
```

```
precision total: nan
recall total: 0.5
```

```
def accuracy(confusion_matrix):
    diagonal_sum = confusion_matrix.trace()
    sum_of_all_elements = confusion_matrix.sum()
    return diagonal_sum / sum_of_all_elements
```

```
accuracy(cm)
```

```
0.9
```

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import cross_val_score, train_test_split
from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions
np.random.seed(0)
```

```
clf1 = DecisionTreeClassifier(criterion='entropy', max_depth=1)
clf2 = KNeighborsClassifier(n_neighbors=1)

bagging1 = BaggingClassifier(base_estimator=clf1, n_estimators=10,
max_samples=0.8, max_features=0.8)
bagging2 = BaggingClassifier(base_estimator=clf2, n_estimators=10,
max_samples=0.8, max_features=0.8)
```

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
```

```
import h5py
import numpy as np
import os
import glob
import cv2
import warnings
from matplotlib import pyplot
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
warnings.filterwarnings('ignore')
# tunable-parameters
num_trees = 100
test_size = 0.10
seed      = 9
test_path = "dataset/test"
h5_data   = 'output/data.h5'
h5_labels = 'output/labels.h5'
scoring   = "accuracy"
if not os.path.exists(test_path):
    os.makedirs(test_path)
# create all the machine learning models
models = []
models.append(('LR', LogisticRegression(random_state=seed)))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(random_state=seed)))
models.append(('RF', RandomForestClassifier(n_estimators=num_trees,
random_state=seed)))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(random_state=seed)))
# variables to hold the results and names
results = []
names   = []
# import the feature vector and trained labels
h5f_data  = h5py.File(h5_data, 'r')
h5f_label = h5py.File(h5_labels, 'r')
global_features_string = h5f_data['dataset_1']
global_labels_string   = h5f_label['dataset_1']
global_features = np.array(global_features_string)
global_labels   = np.array(global_labels_string)

h5f_data.close()
h5f_label.close()
# verify the shape of the feature vector and labels
print("[STATUS] features shape: {}".format(global_features.shape))
print("[STATUS] labels shape: {}".format(global_labels.shape))
print("[STATUS] training started...")
```

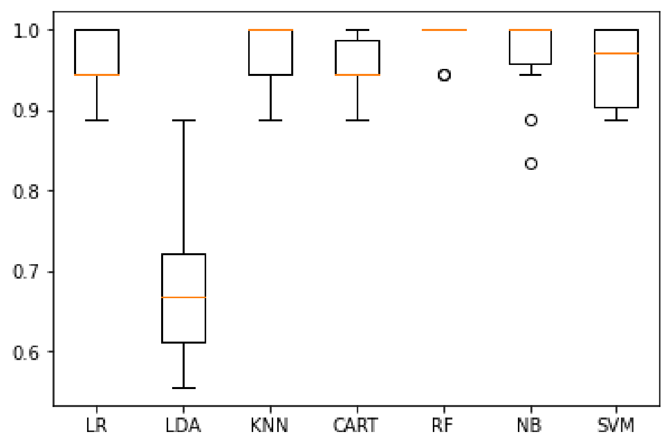
```
[STATUS] features shape: (200, 532)
[STATUS] labels shape: (200,)
[STATUS] training started...
```

```
# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) = 
train_test_split(np.array(global_features),

np.array(global_labels),

test_size=test_size,

random_state=seed)
print("[STATUS] splitted train and test data...")
print("Train data  : {}".format(trainDataGlobal.shape))
print("Test data   : {}".format(testDataGlobal.shape))
print("Train labels: {}".format(trainLabelsGlobal.shape))
print("Test labels : {}".format(testLabelsGlobal.shape))
```

```
[STATUS] splitted train and test data...
Train data  : (180, 532)
Test data   : (20, 532)
Train labels: (180,)
Test labels : (20,)# 10-fold cross validation
for name, model in models:
    kfold = KFold(n_splits=10, random_state=None)
    cv_results = cross_val_score(model, trainDataGlobal, trainLabelsGlobal, 
cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = pyplot.figure()
fig.suptitle('Machine Learning algorithm comparison for species 
identification')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

```
LR: 0.955556 (0.041574)
LDA: 0.683333 (0.108440)
KNN: 0.966667 (0.044444)
CART: 0.950000 (0.038889)
RF: 0.988889 (0.022222)
NB: 0.966667 (0.056656)
SVM: 0.955556 (0.048432)
```

Machine Learning algorithm comparison for species identification




```
from sklearn.metrics import classification_report
# create the model - Random Forests
rf  = RandomForestClassifier(n_estimators=100)
from sklearn.ensemble import AdaBoostClassifier
clf=AdaBoostClassifier(base_estimator=rf)
# fit the training data to the model
clf.fit(trainDataGlobal, trainLabelsGlobal)
#print(clf.fit(trainDataGlobal, trainLabelsGlobal))
clf_pred = clf.predict(trainDataGlobal)
#clf_pred = clf.predict(global_feature.reshape(1,-1))[0]
print(classification_report(trainLabelsGlobal,clf_pred))
#print(confusion_matrix(trainLabelsGlobal,clf_pred))
#print(clf.predict(trainDataGlobal))
#print(clf.predict(global_feature.reshape(1,-1))[0])
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        91
           1       1.00      1.00      1.00        89

    accuracy                           1.00       180
   macro avg       1.00      1.00      1.00       180
weighted avg       1.00      1.00      1.00       180
```

Result – Hence we have successfully performed Random Forest Algorithm over image dataset of two class.

Experiment – 9

Aim – To Perform Ensemble Learning on classified Data.

Platform – Google Colab

What is meant by ensemble learning?

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.)

About Dataset – Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian : "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34]. This database is also available through the UW CS ftp server: ftp ftp.cs.wisc.educd math-prog/cpo-dataset/machine-learn/WDBC/

Also can be found on UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

Attribute Information:

1) ID number

2) Diagnosis (M = malignant, B = benign)

3-32) Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)

b) texture (standard deviation of gray-scale values)

c) perimeter

d) area

e) smoothness (local variation in radius lengths)

f) compactness (perimeter^2 / area - 1.0)

g) concavity (severity of concave portions of the contour)

h) concave points (number of concave portions of the contour)

i) symmetry

j) fractal dimension ("coastline approximation" - 1) The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius. All feature values are recoded with four significant digits.

Missing attribute values: none Class distribution: 357 benign, 212 malignant

```
import pandas as pd
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

Importing Dataset

```
cd /content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P9_Ensemble Learning
```

```
/content/gdrive/MyDrive/Machine_Learning/Colab
Notebooks/ML_Practicals/1_Practical/P9_Ensemble Learning
```

```
ls
```

```
 breast-cancer.csv  'Main_Models (2).ipynb'  'wine (1).csv'
```

```
df = pd.read_csv('breast-cancer.csv')
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_curve, auc
```

Checking Dataset

```
df.head()
```

```
         id  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302        17.99         10.38          122.80     1001.0   
1    842517        20.57         17.77          132.90     1326.0   
2  84300903        19.69         21.25          130.00     1203.0   
3  84348301        11.42         20.38           77.58      386.1   
4  84358402        20.29         14.34          135.10     1297.0   

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710   
1          0.08474           0.07864          0.0869              0.07017   
2          0.10960           0.15990          0.1974              0.12790   
3          0.14250           0.28390          0.2414              0.10520   
4          0.10030           0.13280          0.1980              0.10430   
```

```
   symmetry_mean  ...  texture_worst  perimeter_worst  area_worst  \
0         0.2419  ...          17.33           184.60      2019.0   
1         0.1812  ...          23.41           158.80      1956.0   
2         0.2069  ...          25.53           152.50      1709.0   
3         0.2597  ...          26.50            98.87       567.7   
4         0.1809  ...          16.67           152.20      1575.0   

   smoothness_worst  compactness_worst  concavity_worst  concave points_worst  
\
0            0.1622             0.6656           0.7119                0.2654   
1            0.1238             0.1866           0.2416                0.1860   
2            0.1444             0.4245           0.4504                0.2430   
3            0.2098             0.8663           0.6869                0.2575   
4            0.1374             0.2050           0.4000                0.1625   

   symmetry_worst  fractal_dimension_worst  diagnosis  
0          0.4601                  0.11890          0  
1          0.2750                  0.08902          0  
2          0.3613                  0.08758          0  
3          0.6638                  0.17300          0  
4          0.2364                  0.07678          0  

[5 rows x 32 columns]
```

```
df.shape
```

```
(569, 32)
```

```
vars = ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']

y_cols = 'diagnosis'
x_cols = vars
# x_cols.remove(y_cols)

dfdtype = pd.DataFrame(df.dtypes)
flag_cols = list(dfdtype[dfdtype.iloc[:,0] == 'object'].index)

df['diagnosis'].value_counts()
```

```
1    357
0    212
Name: diagnosis, dtype: int64
```

```
4762/24037
```

```
0.19811124516370596
```

```
zerodf = df[df[y_cols]==0].sample(112)
onedf = df[df[y_cols]== 1]

newdf = pd.concat([zerodf, onedf], axis=0)
newdf[y_cols].value_counts()
```

```
1    357
0    112
Name: diagnosis, dtype: int64
```

```
df[y_cols].value_counts()
```

```
1    357
0    212
Name: diagnosis, dtype: int64
```

```
newdf[y_cols].value_counts()
```

```
1    357
0    112
Name: diagnosis, dtype: int64
```

Statistical Summary of Dataset

```
df.describe()
```

```
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean
```

```
\
count      569.000000     569.000000     569.000000     569.000000
mean         0.096360       0.104341       0.088799       0.048919
std          0.014064       0.052813       0.079720       0.038803
min          0.052630       0.019380       0.000000       0.000000
25%          0.086370       0.064920       0.029560       0.020310
50%          0.095870       0.092630       0.061540       0.033500
75%          0.105300       0.130400       0.130700       0.074000
max          0.163400       0.345400       0.426800       0.201200

       symmetry_mean  ...  texture_worst  perimeter_worst   area_worst  \
count     569.000000  ...     569.000000       569.000000   569.000000
mean        0.181162  ...      25.677223       107.261213   880.583128
std         0.027414  ...       6.146258        33.602542   569.356993
min         0.106000  ...      12.020000        50.410000   185.200000
25%         0.161900  ...      21.080000        84.110000   515.300000
50%         0.179200  ...      25.410000        97.660000   686.500000
75%         0.195700  ...      29.720000       125.400000  1084.000000
max         0.304000  ...      49.540000       251.200000  4254.000000

       smoothness_worst  compactness_worst  concavity_worst  \
count        569.000000         569.000000       569.000000
mean           0.132369           0.254265         0.272188
std            0.022832           0.157336         0.208624
min            0.071170           0.027290         0.000000
25%            0.116600           0.147200         0.114500
50%            0.131300           0.211900         0.226700
75%            0.146000           0.339100         0.382900
max            0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst  \
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
min                0.000000        0.156500                 0.055040
25%                0.064930        0.250400                 0.071460
50%                0.099930        0.282200                 0.080040
75%                0.161400        0.317900                 0.092080
max                0.291000        0.663800                 0.207500

        diagnosis
count  569.000000
mean     0.627417
std      0.483918
min      0.000000
25%      0.000000
50%      1.000000
75%      1.000000
```

```
max      1.000000

[8 rows x 32 columns]
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import neighbors
newdf = newdf.dropna()

# Helper functions to calculate the performance of our models.
import itertools
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    np.set_printoptions(precision=2)
    plt.figure()
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
#         print("Normalized confusion matrix")
#     else:
#         print('Confusion matrix, without normalization')

#     print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
```

```
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig(str(title.split('\n')[0])+'.png')
    plt.show()

def overall_error_rate(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    fn = cnf_matrix[1,0]
    fp = cnf_matrix[0,1]
    tn = cnf_matrix[0,0]
    tp = cnf_matrix[1,1]
    n = len(y_test)
    return (fn+fp)/n
def sensitivity(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    tap = pd.DataFrame(y_test).iloc[:,0].value_counts()[1]
    tp = cnf_matrix[1,1]
    return tp/tap
def false_pos_rate(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    fp = cnf_matrix[0,1]
    tan = pd.DataFrame(y_test).iloc[:,0].value_counts()[0]
    return fp/tan
def specificity(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    tn = cnf_matrix[0,0]
    tan = pd.DataFrame(y_test).iloc[:,0].value_counts()[0]
    return tn/tan
def false_neg_rate(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    fn = cnf_matrix[1,0]
    tap = pd.DataFrame(y_test).iloc[:,0].value_counts()[1]
    return fn/tap
def prop_true_pos(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    try:
        tpp = pd.DataFrame(y_pred).iloc[:,0].value_counts()[1]
    except:
        return 0
    tp = cnf_matrix[1,1]
    return tp/tpp
def prop_true_neg(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    try:
```

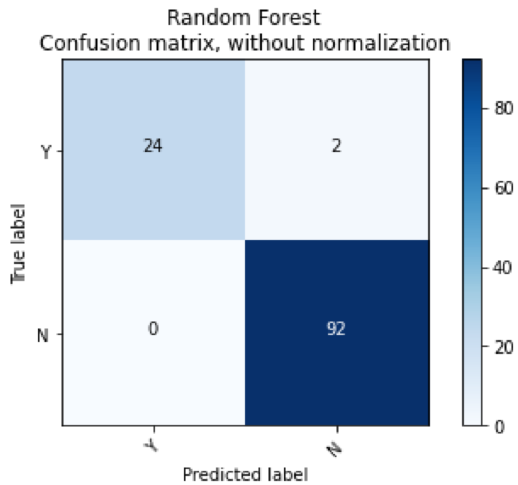
```
        tn = cnf_matrix[0,0]
        tpn = pd.DataFrame(y_pred).iloc[:,0].value_counts()[0]
    except:
        return 0
    return tn/tpn
def recall(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    try:
        tp = cnf_matrix[1,1]
        fn = cnf_matrix[1,0]
        tpn = pd.DataFrame(y_pred).iloc[:,0].value_counts()[0]
    except:
        return 0
    return tp/(fn+tp)
def precision(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    try:
        tp = cnf_matrix[1,1]
        fp = cnf_matrix[0,1]
        tpn = pd.DataFrame(y_pred).iloc[:,0].value_counts()[0]
    except:
        return 0
    return tp/(fp+tp)
def npv(y_pred, y_test):
    cnf_matrix = confusion_matrix(y_test, y_pred)
    try:
        fn = cnf_matrix[1,0]
        fn = cnf_matrix[1,0]
        tn = cnf_matrix[0,0]
        tpn = pd.DataFrame(y_pred).iloc[:,0].value_counts()[0]
    except:
        return 0
    return tn/(tn+fn)
def f1score(y_pred, y_test):
    prec = precision(y_pred, y_test)
    rec = recall(y_pred, y_test)
    f1 = 2 * ((prec * rec)/(prec + rec))
    return f1
def get_descriptive_data(y_pred, y_test):
    print("Accuracy: %f%%" %(round(accuracy_score(y_test, y_pred)*100,2)))
    print("Overall Error Rate: %f%%" %(round(overall_error_rate(y_pred,
y_test)*100,2)))
    print('False Positive Rate: %f%%' %(round(false_pos_rate(y_pred,
y_test)*100,2)))
    print('False Negative Rate: %f%%' %(round(false_neg_rate(y_pred,
y_test)*100,2)))
    print('Specificity: %f%%' %(round(specificity(y_pred, y_test)*100,2)))
    print("Sensitivity: %f%%" %(round(sensitivity(y_pred, y_test)*100,2)))
```

```
    print('Proportion True Positive: %f%%' %(round(prop_true_pos(y_pred, 
y_test)*100,2)))
    print('Proportion True Negative: %f%%' %(round(prop_true_neg(y_pred, 
y_test)*100,2)))
    print("recall: %f%%" %(round(recall(y_pred, y_test)*100,2)))
    print("precision: %f%%" %(round(precision(y_pred, y_test)*100,2)))
    print("FDR: %f%%" %(100-round(precision(y_pred, y_test)*100,2)))
    print("NPV: %f%%" %(round(precision(y_pred, y_test)*100,2)))
    print("FOR: %f%%" %(100-round(npv(y_pred, y_test)*100,2)))
    print("F1SCORE: %f%%" %(100-round(f1score(y_pred, y_test)*100,2)))
```

Single RANDOM FOREST ON 75 25 TRAIN TEST SPLIT

```
# Random Forrest
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols].values, 
                                                    newdf[y_cols].values, 
                                                    test_size=0.25, 
                                                    random_state=1)
rf = RandomForestClassifier(random_state=1, n_jobs=-1)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
get_descriptive_data(y_pred, y_test)

y_score = rf.predict_proba(X_test)[:, 1]
rf_fpr, rf_tpr, _ = roc_curve(y_test, y_score)
rf_roc_auc = auc(rf_fpr, rf_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Random Forest\nConfusion matrix, without 
normalization')
```

```
Accuracy: 98.310000%
Overall Error Rate: 1.690000%
False Positive Rate: 7.690000%
False Negative Rate: 0.000000%
Specificity: 92.310000%
Sensitivity: 100.000000%
Proportion True Positive: 97.870000%
Proportion True Negative: 100.000000%
recall: 100.000000%
precision: 97.870000%
FDR: 2.130000%
NPV: 97.870000%
FOR: 0.000000%
F1SCORE: 1.080000%
```

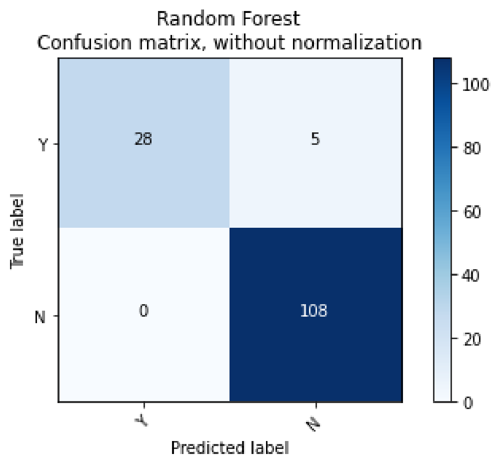


Single RANDOM FOREST ON 70 30 TRAIN TEST SPLIT

```
# Random Forrest
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols].values,
                                                    newdf[y_cols].values,
                                                    test_size=0.3,
                                                    random_state=1)
rf = RandomForestClassifier(random_state=1, n_jobs=-1)

rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
get_descriptive_data(y_pred, y_test)

y_score = rf.predict_proba(X_test)[:, 1]
rf_fpr, rf_tpr, _ = roc_curve(y_test, y_score)
rf_roc_auc = auc(rf_fpr, rf_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Random Forest\nConfusion matrix, without
normalization')
```

```
Accuracy: 96.450000%
Overall Error Rate: 3.550000%
False Positive Rate: 15.150000%
False Negative Rate: 0.000000%
Specificity: 84.850000%
Sensitivity: 100.000000%
Proportion True Positive: 95.580000%
Proportion True Negative: 100.000000%
```

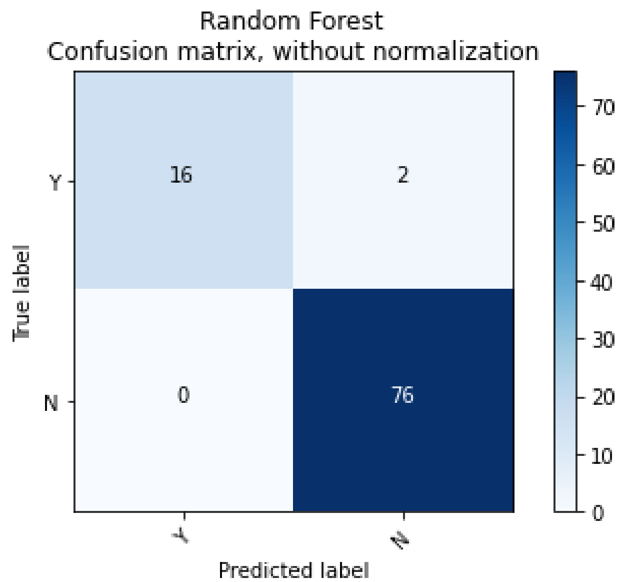
```
recall: 100.000000%
precision: 95.580000%
FDR: 4.420000%
NPV: 95.580000%
FOR: 0.000000%
F1SCORE: 2.260000%
```



Single RANDOM FOREST ON 80 20 TRAIN TEST SPLIT

```
# Random Forrest
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols].values,
                                                    newdf[y_cols].values,
                                                    test_size=0.20,
                                                    random_state=1)
rf = RandomForestClassifier(random_state=1, n_jobs=-1)
np.random.seed(1234)

rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
get_descriptive_data(y_pred, y_test)

y_score = rf.predict_proba(X_test)[:, 1]
rf_fpr, rf_tpr, _ = roc_curve(y_test, y_score)
rf_roc_auc = auc(rf_fpr, rf_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Random Forest\nConfusion matrix, without 
normalization')
```

```
Accuracy: 97.870000%
Overall Error Rate: 2.130000%
False Positive Rate: 11.110000%
False Negative Rate: 0.000000%
Specificity: 88.890000%
Sensitivity: 100.000000%
Proportion True Positive: 97.440000%
Proportion True Negative: 100.000000%
recall: 100.000000%
precision: 97.440000%
FDR: 2.560000%
NPV: 97.440000%
FOR: 0.000000%
F1SCORE: 1.300000%
```

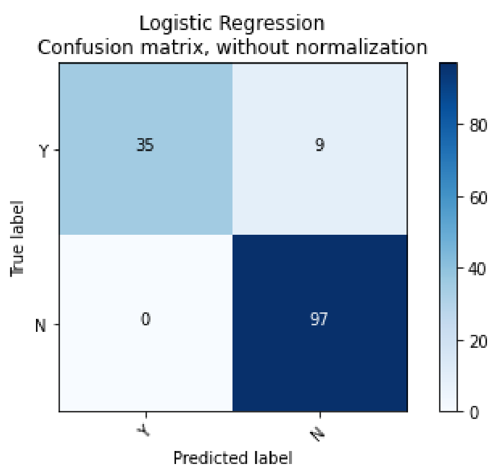


Single Logistic Regression 70 30 train test

```
# Logistic Regression
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.30, random_state=42)
lr = LogisticRegression(random_state=1)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
get_descriptive_data(y_pred, y_test)

y_score = lr.predict_proba(X_test)[:, 1]
lr_fpr, lr_tpr, _ = roc_curve(y_test, y_score)
lr_roc_auc = auc(lr_fpr, lr_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
```

```
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Logistic Regression\nConfusion matrix, without
normalization')
```

```
Accuracy: 93.620000%
Overall Error Rate: 6.380000%
False Positive Rate: 20.450000%
False Negative Rate: 0.000000%
Specificity: 79.550000%
Sensitivity: 100.000000%
Proportion True Positive: 91.510000%
Proportion True Negative: 100.000000%
recall: 100.000000%
precision: 91.510000%
FDR: 8.490000%
NPV: 91.510000%
FOR: 0.000000%
F1SCORE: 4.430000%
```

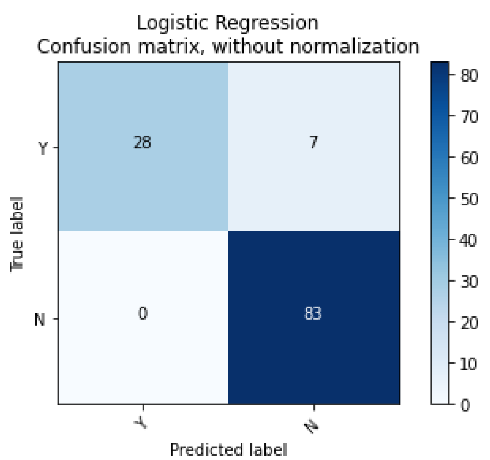


Single Logistic Regression 75 25 train test

```
# Logistic Regression
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.25, random_state=42)
lr = LogisticRegression(random_state=1)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
get_descriptive_data(y_pred, y_test)

y_score = lr.predict_proba(X_test)[:, 1]
lr_fpr, lr_tpr, _ = roc_curve(y_test, y_score)
```

```
lr_roc_auc = auc(lr_fpr, lr_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Logistic Regression\nConfusion matrix, without
normalization')
```

```
Accuracy: 94.070000%
Overall Error Rate: 5.930000%
False Positive Rate: 20.000000%
False Negative Rate: 0.000000%
Specificity: 80.000000%
Sensitivity: 100.000000%
Proportion True Positive: 92.220000%
Proportion True Negative: 100.000000%
recall: 100.000000%
precision: 92.220000%
FDR: 7.780000%
NPV: 92.220000%
FOR: 0.000000%
F1SCORE: 4.050000%
```

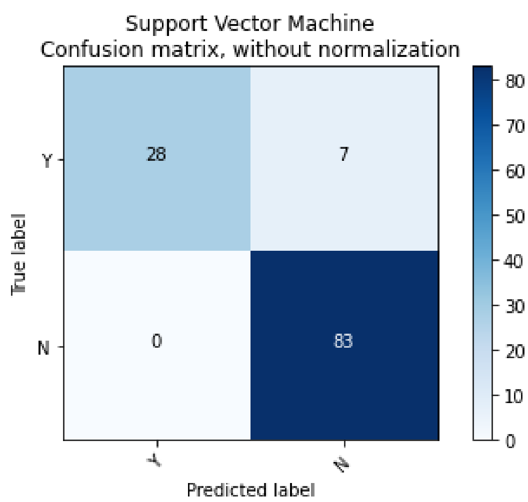


Single SUPPORT VECTOR MACHINE 75 25 train test

```
# Support Vector Machine
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.25, random_state=42)
svm = SVC(random_state=1)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
y_score = svm.decision_function(X_test)
svm_fpr, svm_tpr, _ = roc_curve(y_test, y_score)
```

```
svm_roc_auc = auc(svm_fpr, svm_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Support Vector Machine\nConfusion matrix,
without normalization')
```

```
Accuracy: 94.070000%
Overall Error Rate: 5.930000%
False Positive Rate: 8.430000%
False Negative Rate: 0.000000%
Specificity: 33.730000%
Sensitivity: 237.140000%
Proportion True Positive: 296.430000%
Proportion True Negative: 31.110000%
recall: 100.000000%
precision: 92.220000%
FDR: 7.780000%
NPV: 92.220000%
FOR: 0.000000%
F1SCORE: 4.050000%
```

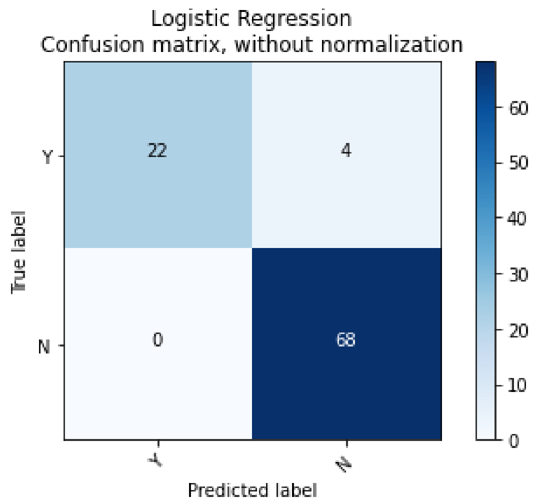


Single Logistic Regression 80 20 train test

```
# Logistic Regression
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.20, random_state=42)
lr = LogisticRegression(random_state=1)
lr.fit(X_train, y_train)
```

```
y_pred = lr.predict(X_test)
get_descriptive_data(y_pred, y_test)

y_score = lr.predict_proba(X_test)[:, 1]
lr_fpr, lr_tpr, _ = roc_curve(y_test, y_score)
lr_roc_auc = auc(lr_fpr, lr_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Logistic Regression\nConfusion matrix, without 
normalization')
```

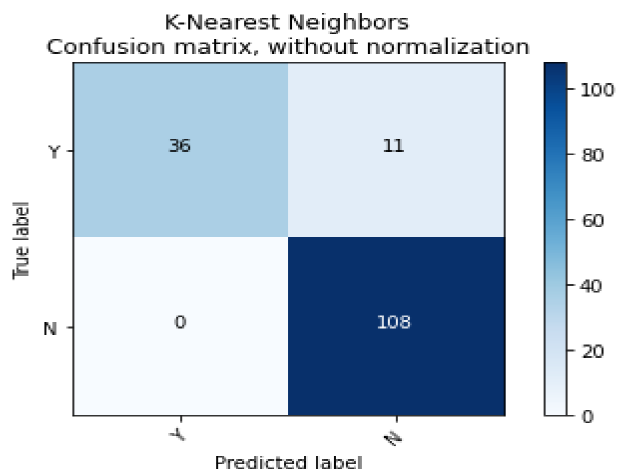
```
Accuracy: 95.740000%
Overall Error Rate: 4.260000%
False Positive Rate: 15.380000%
False Negative Rate: 0.000000%
Specificity: 84.620000%
Sensitivity: 100.000000%
Proportion True Positive: 94.440000%
Proportion True Negative: 100.000000%
recall: 100.000000%
precision: 94.440000%
FDR: 5.560000%
NPV: 94.440000%
FOR: 0.000000%
F1SCORE: 2.860000%
```



Single K-Nearest Neighbour 70 30 trai test

```
# K-Nearest Neighbors
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols], newdf[y_cols], test_size=0.33, random_state=42)
knn = neighbors.KNeighborsClassifier(n_jobs=-1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
y_score = knn.predict_proba(X_test)[:, 1]
knn_fpr, knn_tpr, _ = roc_curve(y_test, y_score)
knn_roc_auc = auc(knn_fpr, knn_tpr)
y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='K-Nearest Neighbors\nConfusion matrix, without normalization')
```

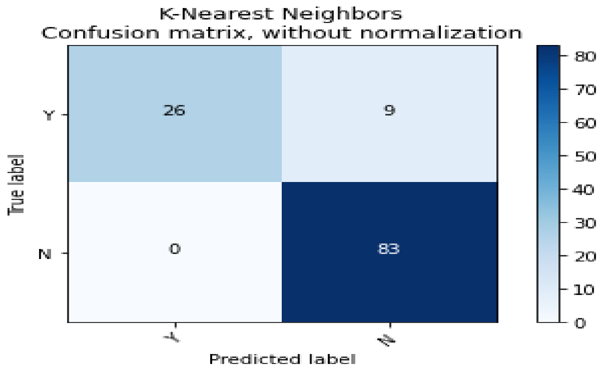
```
Accuracy: 92.900000%
Overall Error Rate: 7.100000%
False Positive Rate: 10.190000%
False Negative Rate: 0.000000%
Specificity: 33.330000%
Sensitivity: 229.790000%
Proportion True Positive: 300.000000%
Proportion True Negative: 30.250000%
recall: 100.000000%
precision: 90.760000%
FDR: 9.240000%
NPV: 90.760000%
FOR: 0.000000%
F1SCORE: 4.850000%
```



Single K-Nearest Neighbour 75 25 train test

```
# K-Nearest Neighbors
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.25, random_state=42)
knn = neighbors.KNeighborsClassifier(n_jobs=-1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

y_score = knn.predict_proba(X_test)[:, 1]
knn_fpr, knn_tpr, _ = roc_curve(y_test, y_score)
knn_roc_auc = auc(knn_fpr, knn_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='K-Nearest Neighbors\nConfusion matrix, without
normalization')
```

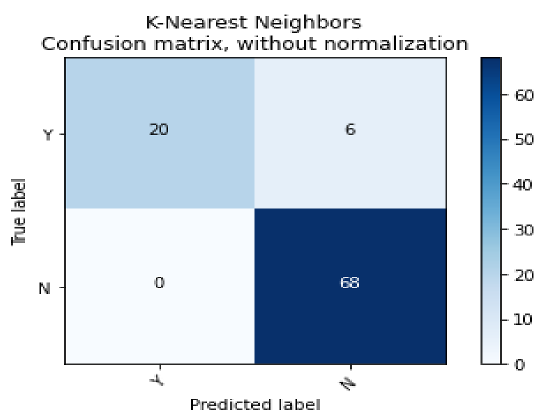
```
Accuracy: 92.370000%
Overall Error Rate: 7.630000%
False Positive Rate: 10.840000%
False Negative Rate: 0.000000%
Specificity: 31.330000%
Sensitivity: 237.140000%
Proportion True Positive: 319.230000%
Proportion True Negative: 28.260000%
recall: 100.000000%
precision: 90.220000%
FDR: 9.780000%
NPV: 90.220000%
FOR: 0.000000%
F1SCORE: 5.140000%
```



Single K-Nearest Neighbour 80 20 train test

```
# K-Nearest Neighbors
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.20, random_state=42)
knn = neighbors.KNeighborsClassifier(n_jobs=-1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

y_score = knn.predict_proba(X_test)[:, 1]
knn_fpr, knn_tpr, _ = roc_curve(y_test, y_score)
knn_roc_auc = auc(knn_fpr, knn_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='K-Nearest Neighbors\nConfusion matrix, without
normalization')
```

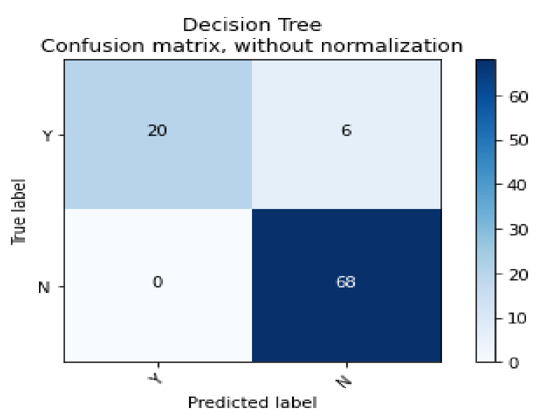
```
Accuracy: 93.620000%
Overall Error Rate: 6.380000%
False Positive Rate: 8.820000%
False Negative Rate: 0.000000%
Specificity: 29.410000%
Sensitivity: 261.540000%
Proportion True Positive: 340.000000%
Proportion True Negative: 27.030000%
recall: 100.000000%
precision: 91.890000%
FDR: 8.110000%
NPV: 91.890000%
FOR: 0.000000%
F1SCORE: 4.230000%
```



Single Decision Tree 80 20 train test

```
# Decision Tree
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.20, random_state=42)
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train,y_train)
y_pred_dtc = dtc.predict(X_test)
y_score = dtc.predict_proba(X_test)[:, 1]
dtc_fpr, dtc_tpr, _ = roc_curve(y_test, y_score)
dtc_roc_auc = auc(dtc_fpr, dtc_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Decision Tree\nConfusion matrix, without
normalization')
```

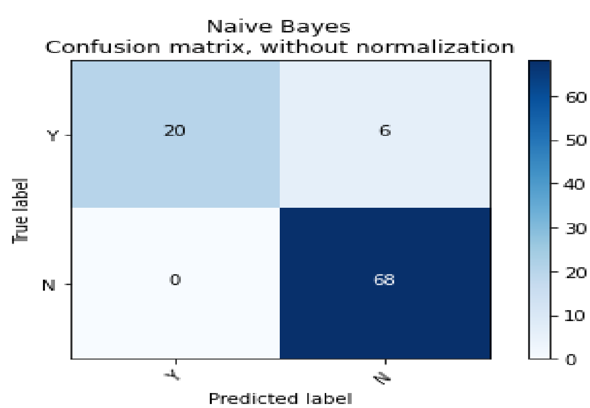
```
Accuracy: 93.620000%
Overall Error Rate: 6.380000%
False Positive Rate: 8.820000%
False Negative Rate: 0.000000%
Specificity: 29.410000%
Sensitivity: 261.540000%
Proportion True Positive: 340.000000%
Proportion True Negative: 27.030000%
recall: 100.000000%
precision: 91.890000%
FDR: 8.110000%
NPV: 91.890000%
FOR: 0.000000%
F1SCORE: 4.230000%
```



Single Naive Bayes 80 20 train test

```
# Naive Bayes
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.20, random_state=42)
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred_dtc = gnb.predict(X_test)
y_score = gnb.predict_proba(X_test)[:, 1]
gnb_fpr, gnb_tpr, _ = roc_curve(y_test, y_score)
gnb_roc_auc = auc(gnb_fpr, gnb_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Naive Bayes\nConfusion matrix, without
normalization')
```

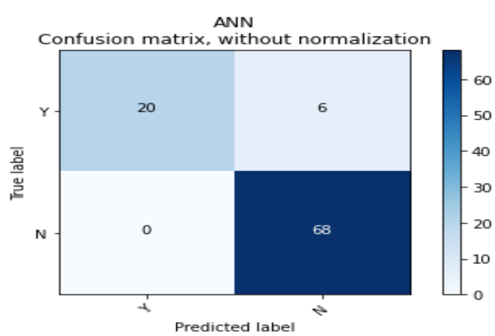
```
Accuracy: 93.620000%
Overall Error Rate: 6.380000%
False Positive Rate: 8.820000%
False Negative Rate: 0.000000%
Specificity: 29.410000%
Sensitivity: 261.540000%
Proportion True Positive: 340.000000%
Proportion True Negative: 27.030000%
recall: 100.000000%
precision: 91.890000%
FDR: 8.110000%
NPV: 91.890000%
FOR: 0.000000%
F1SCORE: 4.230000%
```



Single ANN 80 20 train test

```
# ANN
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.20, random_state=42)
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2),
random_state=1)
clf.fit(X_train, y_train)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
y_pred_dtc = clf.predict(X_test)
y_score = clf.predict_proba(X_test)[:, 1]
clf_fpr, clf_tpr, _ = roc_curve(y_test, y_score)
clf_roc_auc = auc(clf_fpr, clf_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='ANN\nConfusion matrix, without normalization')
```

```
Accuracy: 93.620000%
Overall Error Rate: 6.380000%
False Positive Rate: 8.820000%
False Negative Rate: 0.000000%
Specificity: 29.410000%
Sensitivity: 261.540000%
Proportion True Positive: 340.000000%
Proportion True Negative: 27.030000%
recall: 100.000000%
precision: 91.890000%
FDR: 8.110000%
NPV: 91.890000%
FOR: 0.000000%
F1SCORE: 4.230000%
```



```
# BIRCH
X_train, X_test, y_train, y_test = train_test_split(newdf[x_cols],
newdf[y_cols], test_size=0.25, random_state=42)
from sklearn.cluster import Birch
brc = Birch(branching_factor = 40, n_clusters = 3, threshold = 1.5)
brc = Birch(n_clusters=None)
brc.fit(X_train)
Birch(n_clusters=None)
brc.predict(X_train)
y_pred_brc = brc.predict(X_train)

brc_fpr, brc_tpr, _ = roc_curve(y_test, y_score)
brc_roc_auc = auc(brc_fpr, brc_tpr)

y_pred = pd.Series(y_pred).replace([0,1], ['N','Y'])
y_test = pd.Series(y_test).replace([0,1], ['N','Y'])
class_names = list(y_pred.value_counts().index)
get_descriptive_data(y_pred, y_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='\nBIRCH')

from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

pip install imputer

ERROR: Could not find a version that satisfies the requirement imputer (from
versions: none)
ERROR: No matching distribution found for imputer

import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler

# Convert the DataFrame object into NumPy array otherwise you will not be
able to impute
values = df.values

# Now impute it
imputer = SimpleImputer()
imputedData = imputer.fit_transform(values)

scaler = MinMaxScaler(feature_range=(0, 1))
normalizedData = scaler.fit_transform(imputedData)
```

```
X = normalizedData[:,0:117]
Y = normalizedData[:,117]
```

Bagging Decision Tree

```
kfold = model_selection.KFold(n_splits=15, random_state=None)
dtc = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees,
random_state=None)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9926999999999999
```

Adaboost Decision Tree

```
# AdaBoost Classification

from sklearn.ensemble import AdaBoostClassifier
seed = 7
num_trees = 70
kfold = model_selection.KFold(n_splits=10, random_state=None)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=None)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9999652777777778
```

Voting Ensemble Decision Tree

```
# Voting Ensemble for Classification

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=10, random_state=None)
# create the sub models
np.random.seed(1234)
estimators = []
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

```
0.9999652777777778
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
```

Bagging Random Forest

```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
kfold = model_selection.KFold(n_splits=15, random_state=None)
rf = RandomForestClassifier()
X, Y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
      n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=4)
# define model
model = BaggingClassifier()
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model

results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Adaboost Random Forest

```
# AdaBoost Classification

from sklearn.ensemble import AdaBoostClassifier
seed = 7
num_trees = 70
kfold = model_selection.KFold(n_splits=10, random_state=None)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=None)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9999652777777778
```

Voting Ensemble Random Forest

```
# Voting Ensemble for Classification

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=10, random_state=None)
# create the sub models
estimators = []
model1 = RandomForestClassifier()
```

```
estimators.append(('random', model1))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

```
0.9999305555555555
```

Bagging KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import BaggingClassifier
from statistics import mean
from statistics import *
# define dataset
X, Y = make_classification(n_samples=1000, n_features=20, n_informative=15,
n_redundant=5, random_state=5)
# define the model
model = BaggingClassifier(base_estimator=KNeighborsClassifier())
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9799652777777778
```

ADABOOST KNN

```
# AdaBoost Classification

from sklearn.ensemble import AdaBoostClassifier
kfold = model_selection.KFold(n_splits=10, random_state=None)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=None)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9999652777777778
```

VOTING KNN

```
# Voting Ensemble for Classification

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
```

```
kfold = model_selection.KFold(n_splits=10, random_state=None)
# create the sub models
estimators = []
model1 = KNeighborsClassifier()
estimators.append(('random', model1))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

```
0.8687803804368801
```

BAGGING LOGISTIC

```
from sklearn.ensemble import BaggingClassifier
from statistics import mean
from statistics import *
# define dataset
X, Y = make_classification(n_samples=1000, n_features=20, n_informative=15,
n_redundant=5, random_state=5)
# define the model
model =
BaggingClassifier(base_estimator=LogisticRegression(random_state=1),n_estimat
ors=100,max_features=10,max_samples=100,random_state=1, n_jobs=5)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9926999999999999
```

ADABOOST LOGISTIC

```
# AdaBoost Classification

from sklearn.ensemble import AdaBoostClassifier
kfold = model_selection.KFold(n_splits=10, random_state=None)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=None)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

VOTING LOGISTIC

```
# Voting Ensemble for Classification

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

```
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=10, random_state=None)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('random', model1))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

BAGGING NAIVE BAYES

```
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import GaussianNB
from statistics import mean
from statistics import *
# define dataset
X, Y = make_classification(n_samples=1000, n_features=20, n_informative=15,
n_redundant=5, random_state=5)
# define the model
model = BaggingClassifier(GaussianNB(),n_estimators = 10, max_features =
0.5,random_state = 0, n_jobs = -1)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9926999999999999
```

ADABOOST NAIVE BAYES

```
# AdaBoost Classification

from sklearn.ensemble import AdaBoostClassifier
kfold = model_selection.KFold(n_splits=10, random_state=None)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=None)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9926999999999999
```

VOTING NAIVE BAYES

```
# Voting Ensemble for Classification

from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=10, random_state=None)
# create the sub models
estimators = []
model1 = GaussianNB()
estimators.append(('gaussian', model1))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())

from sklearn.ensemble import BaggingClassifier
from sklearn.neural_network import MLPClassifier
# define dataset
from sklearn.model_selection import cross_val_score
X, Y = make_classification(n_samples=1000, n_features=20, n_informative=15,
n_redundant=5, random_state=5)
model = BaggingClassifier(MLPClassifier(),n_estimators = 10, max_features =
0.5,random_state = 0, n_jobs = -1)
mlp = MLPClassifier(hidden_layer_sizes=(16, 8, 4, 2), max_iter=1001)
clf = BaggingClassifier(mlp, n_estimators=8)
clf.fit(X,Y)
n_scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

plt.scatter(X_train[:,0], X_train[:,1], c=labels, cmap='rainbow', alpha=0.7,
edgecolors='b')

plt.figure(1, figsize=(8,8))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(rf_fpr, rf_tpr,color='b', label='Random Forestz ROC curve (area =
%0.2f)' % rf_roc_auc)
plt.plot(gnb_fpr, gnb_tpr,color='m', label='Naive Bayes ROC curve (area =
%0.2f)' % gnb_roc_auc)
plt.plot(lr_fpr, lr_tpr,color='r', label='Logistic Regression ROC curve (area
= %0.2f)' % lr_roc_auc)
plt.plot(svm_fpr, svm_tpr,color='k', label='Support Vector Machine ROC curve
(area = %0.2f)' % svm_roc_auc)
plt.plot(knn_fpr, knn_tpr,color='g', label='K-Nearest Neighbors ROC curve
(area = %0.2f)' % knn_roc_auc)
plt.plot(dtc_fpr, dtc_tpr,color='c', label='Decision Tree ROC curve (area =
%0.2f)' % dtc_roc_auc)
plt.plot(clf_fpr, clf_tpr,color='y', label='NN ROC curve (area = %0.2f)' %
```

```
clf_roc_auc)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.savefig('roc_curve.png')
plt.show()
```

```
pip install hasy_tools
```

```
Collecting hasy_tools
  Downloading hasy_tools-0.1.1-py3-none-any.whl (14 kB)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from hasy_tools) (1.0.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn->hasy_tools) (1.1.0)
Installing collected packages: hasy-tools
Successfully installed hasy-tools-0.1.1
```

BAGGING SVM

```
from sklearn.svm import LinearSVC
from sklearn.ensemble import BaggingClassifier
import hasy_tools
from sklearn.datasets import make_classification


svm = LinearSVC(random_state=42)
model = BaggingClassifier(base_estimator=svm, n_estimators=31,
random_state=314)
model.fit(X, Y)
X, Y = make_classification(n_samples=10000, n_features=20, n_redundant=0,
      n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=4)
# define model
model = BaggingClassifier()
# define evaluation procedure

print(results.mean())
```

```
0.9999305555555555
```

ADABOOST SVM

```
# AdaBoost Classification

from sklearn.ensemble import AdaBoostClassifier
kfold = model_selection.KFold(n_splits=10, random_state=None)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=None)
```

```
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.9937999999999999
```

VOTING SVM

```
# Voting Ensemble for Classification

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=10, random_state=None)
# create the sub models
estimators = []
model1 = GaussianNB()
estimators.append(('gaussian', model1))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

```
0.9926999999999999
```

NN BAGGING

```
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB

# define dataset
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),
random_state=1)
X, Y = make_classification(n_samples=10000, n_features=20,
n_redundant=0,n_clusters_per_class=1, weights=[0.99], flip_y=0,
random_state=4

clf.fit(X, Y)

print(results.mean())
```

```
0.9926999999999999
```

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from pandas import DataFrame
```

```
X, Y = make_blobs(n_samples=1000, centers=5, n_features=2, cluster_std=2,
random_state=2)
X, Y = make_classification(n_samples=10000, n_features=20, n_redundant=0,
      n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=4)
# define model
model = Sequential()
model.add(Dense(50, input_dim=2, activation='relu'))
model.add(Dense(5, activation='softmax'))
model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# define evaluation procedure

print(results.mean())
```

```
0.9926999999999999
```

Result – Hence we have Performed Ensemble method on Breast Cancer Dataset.

Conclusion – It is Fond That KNN, Naïve Bayes and SVM turned out to be best algorithm with 99% prediction Accuracy.