HEART DISEASE PREDICTION WITH ENSEMBLE LEARNING

A disease is an unnatural medical condition that negatively affects the functional state of an organism and is generally associated with certain signs of illness. As reported by World Health Organization (WHO), Heart Disease and Stroke are the world's biggest killers and have remained the leading causes of death globally in the last 15 years.

In the direction of predicting heart disease, Machine Learning can present remarkable features that simplify the identification of unseen patterns, eventually providing clinical insights that assist physicians in planning and providing care.

In this analysis, the presence of heart disease is predicted by employing Support Vector Machine (SVM), Multinomial Naïve Bayes, Logistic Regression (LR), Decision Tree (DT) & Random Forest (RF), Ensemble combination rules i.e., Majority Voting & Weighted Average Voting and Ensemble classifiers i.e., Bagging, Adaptive Boosting & Gradient Boosting. Parameters such as Accuracy, Precision, Recall and F1-score were estimated to analyze the performance and a comparative study of these classifiers was carried out.

Python Libraries

Python libraries are a collection of functions and methods that allows us to perform many actions without writing the code.

NumPy: NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning.

Pandas: Pandas is a popular Python library for data analysis. Pandas is developed specifically for data extraction and preparation.

Matplotlib: Matpoltlib is a very popular Python library for data visualization. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc.

Scikit-learn: Scikit-learn is one of the most popular ML libraries for classical ML algorithms. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

Seaborn: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

train_test_split: It splits the dataset into a training set and a test set.

```
#import the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import GridSearchCV

from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.externals import joblib
from sklearn.metrics import make_scorer, f1_score, recall_score, precision_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import log_loss
import warnings
warnings.simplefilter(action = 'ignore', category= FutureWarning)


from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
```

Dataset

The Heart Disease dataset has been taken from **Kaggle**. This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. It has a total number of 303 rows and 14 columns among which 165 have a heart disease.

age: age in years

sex: (1 = male; 0 = female)

cp: chest pain type

trestbps: resting blood pressure (in mm Hg on admission to the hospital)

chol: serum cholestoral in mg/dl

fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

restecg: resting electrocardiographic results

thalach: maximum heart rate achieved

exang: exercise induced angina (1 = yes; 0 = no)

oldpeak: ST depression induced by exercise relative to rest

slope: the slope of the peak exercise ST segment

ca: number of major vessels (0-3) colored by flourosopy

thal: thalassemia (1 = normal; 2 = fixed defect; 3 = reversable defect)

target: (1= heart disease or 0= no heart disease)

Read the data

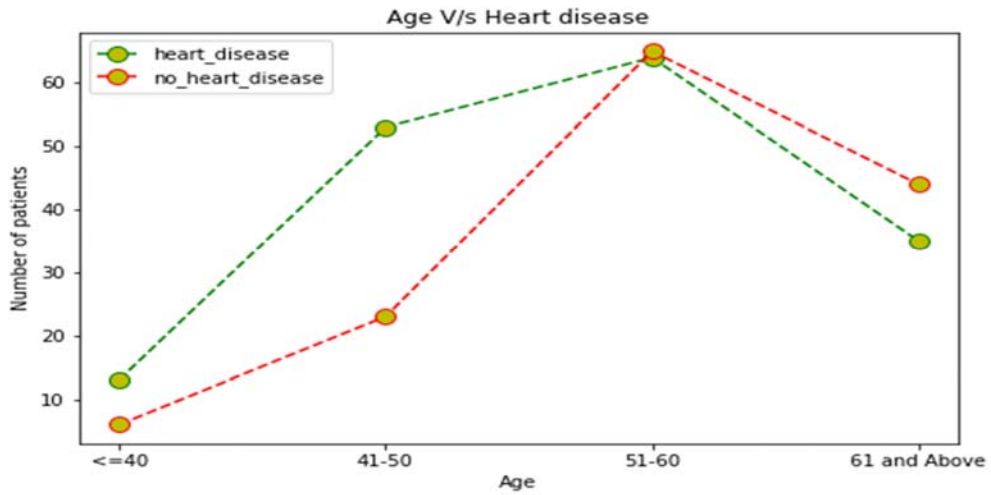
```
#read the csv dataset
data = pd.read_csv("file:/C:/.....................heart.csv", encoding='ANSI')
data.columns
data.head()

#Total number of rows and columns
data.shape
```

```
Out[2]:
   age  sex  cp  trestbps  chol  fbs  ...  exang  oldpeak  slope  ca  thal  target
0   63    1   3       145   233    1  ...      0      2.3      0   0     1       1
1   37    1   2       130   250    0  ...      0      3.5      0   0     2       1
2   41    0   1       130   204    0  ...      0      1.4      2   0     2       1
3   56    1   1       120   236    0  ...      0      0.8      2   0     2       1
4   57    0   0       120   354    0  ...      1      0.6      2   0     2       1

[5 rows x 14 columns]
Out[3]: (303, 14)
```

Graph Plotting

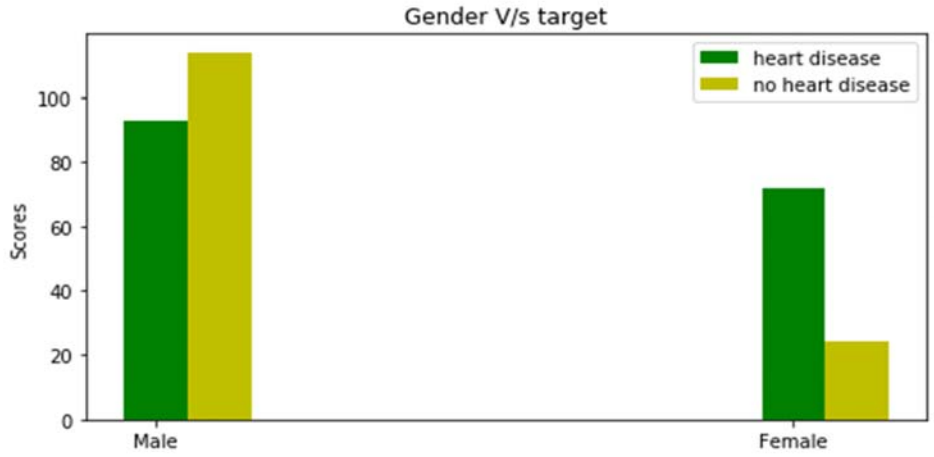
A line graph and bar graph are plotted to demonstrate the interrelationship between Age & Heart disease and Gender & Heart disease, respectively. From the line graph, it can be concluded that in the age group ranging from 41-60 years, the rate of heart disease is the highest.

```
# Plot a line graph for Age V/s heart disease
plt.subplots(figsize =(8,5))
classifiers = ['<=40', '41-50', '51-60','61 and Above']
heart_disease = [13, 53, 64, 35]
no_heart_disease = [6, 23, 65, 44]

l1 = plt.plot(classifiers, heart_disease , color='g', marker='o', linestyle ='dashed', markerfacecolor='y', markersize=10)
l2 = plt.plot(classifiers, no_heart_disease, color='r',marker='o', linestyle ='dashed', markerfacecolor='y', markersize=10 )

plt.xlabel('Age')
plt.ylabel('Number of patients')
plt.title('Age V/s Heart disease')
plt.legend((l1[0], l2[0]), ('heart_disease', 'no_heart_disease'))
plt.show()
```



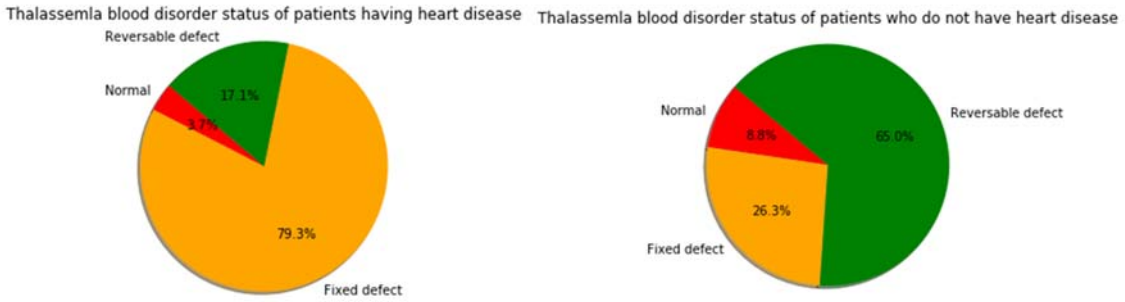
The number of males and females in the dataset is 207 and 96, respectively. Out of 207 males 93 have a heart disease and out of 96 females 72 are suffering from a heart disease.

```
# Plot a bar graph for Gender V/s target
N = 2
ind = np.arange(N)
width = 0.1
fig, ax = plt.subplots(figsize =(8,4))

heart_disease = [93, 72]
rects1 = ax.bar(ind, heart_disease, width, color='g')
no_heart_disease = [114, 24]
rects2 = ax.bar(ind+width, no_heart_disease, width, color='y')

ax.set_ylabel('Scores')
ax.set_title('Gender V/s target')
ax.set_xticks(ind)
ax.set_xticklabels(('Male','Female'))
ax.legend((rects1[0], rects2[0]), ('heart disease', 'no heart disease'))

plt.show()
```



The following pie charts represent the Thalassemia blood disorder status (Normal, Fixed defect and Reversable defect) of patients.

```
#Pie charts for thal:Thalassemla
# Having heart disease
labels= 'Normal', 'Fixed defect', 'Reversable defect'
sizes=[6, 130, 28]
colors=['red', 'orange', 'green']

plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.title('Thalassemla blood disorder status of patients having heart disease')
plt.show()

# Not having heart disease
labels= 'Normal', 'Fixed defect', 'Reversable defect'
sizes=[12, 36, 89]
colors=['red', 'orange', 'green']

plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.title('Thalassemla blood disorder status of patients who do not have heart disease')
plt.show()
```

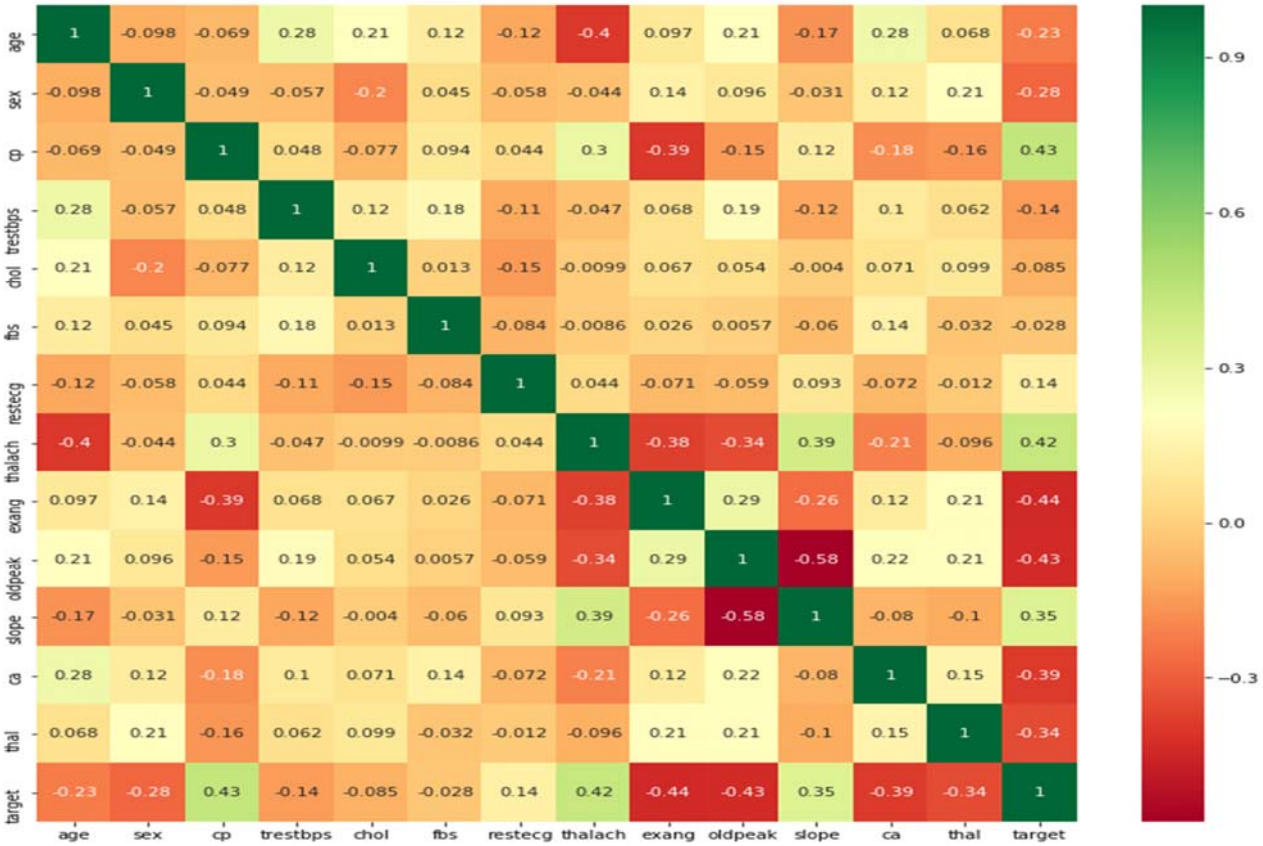


Feature Selection

Irrelevant or partially relevant features can negatively impact the model performance. So, in order to achieve better accuracy for our model we identified the highly important features from the dataset by implementing a feature selection method i.e., Correlation Matrix with Heatmap.

Correlation states how the features are related to each other or the target variable. Correlation can be positive (increase in one value of feature increases the value of the target variable) or negative (increase in one value of feature decreases the value of the target variable).

```
## Feature selection
#get correlation of each feature in dataset

corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(13,13))

#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



Cp, thalach, exang, oldpeak, ca and thal were the highly correlated features with the target variable and we removed the less important features.

```
data=data.drop(['sex', 'fbs', 'restecg', 'slope', 'chol', 'age', 'trestbps'], axis=1)

target=data['target']
data = data.drop(['target'],axis=1)
data.head()
```

```
Out[7]:
   cp  thalach  exang  oldpeak  ca  thal
0   3      150      0      2.3   0     1
1   2      187      0      3.5   0     2
2   1      172      0      1.4   0     2
3   1      178      0      0.8   0     2
4   0      163      1      0.6   0     2
```

Split the dataset

We split the dataset in the ratio 70:30 to create training and testing subsets.

```
# We split the data into training and testing set:
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size=0.3, random_state=10)
```

BASE LEARNERS

Base learners are the normal machine learning algorithms. In this analysis SVM, Multinomial NB, Logistic Regression, Decision Tree and Random Forest are regarded as the base learners. Confusion matrix is considered to describe the performance of these classifiers.

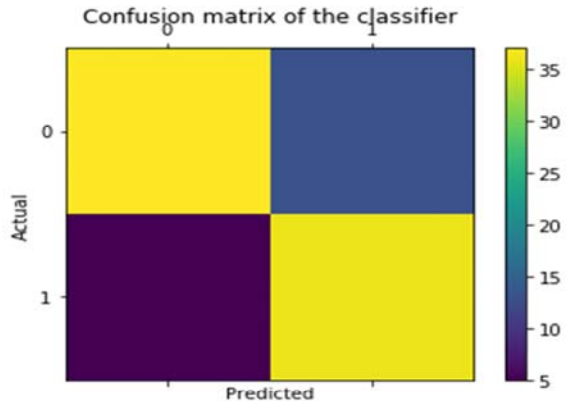
```
## Base Learners
clfs = []
kfolds = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
np.random.seed(1)
```

1. Support Vector Machine (SVM)

```
#Support Vector Machine(SVM)
pipeline_svm = make_pipeline(SVC(probability=True, kernel="linear", class_weight="balanced"))

grid_svm = GridSearchCV(pipeline_svm,
                        param_grid = {'svc__C': [0.01, 0.1, 1]},
                        cv = kfolds,
                        verbose=1,
                        n_jobs=-1)

grid_svm.fit(x_train, y_train)
grid_svm.score(x_test, y_test)
print("\nBest Model: %f using %s" % (grid_svm.best_score_, grid_svm.best_params_))
print('\n')
print('SVM LogLoss {score}'.format(score=log_loss(y_test, grid_svm.predict_proba(x_test))))
clfs.append(grid_svm)
```

```
# save best model to current working directory
joblib.dump(grid_svm, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_grid_svm = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_grid_svm.predict(x_test)
print('SVM accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  labels)
ax.set_yticklabels(['']  labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
SVM accuracy score:  0.8021978021978022


[[37 13]
 [ 5 36]]
```



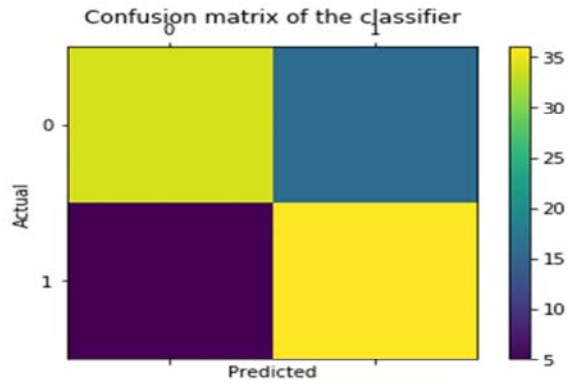

```
              precision    recall  f1-score   support

           0       0.88      0.74      0.80        50
           1       0.73      0.88      0.80        41

    accuracy                           0.80        91
   macro avg       0.81      0.81      0.80        91
weighted avg       0.82      0.80      0.80        91
```

2. Multinomial Naïve Bayes

```
# Multinomial Naive Bayes(NB)
classifierNB=MultinomialNB()
classifierNB.fit(x_train,y_train)
classifierNB.score(x_test, y_test)

print('MultinomialNB LogLoss {score}'.format(score=log_loss(y_test, classifierNB.predict_proba(x_test))))
clfs.append(classifierNB)

# save best model to current working directory
joblib.dump(classifierNB, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_classifierNB = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_classifierNB.predict(x_test)
print('MultinomialNB accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  labels)
ax.set_yticklabels(['']  labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
MultinomialNB accuracy score:  0.7692307692307693
```

```
[[34 16]
 [ 5 36]]
```

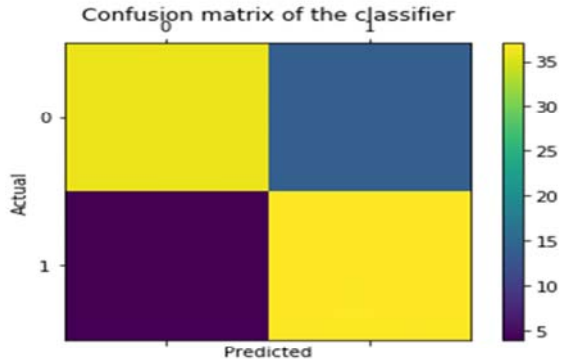



```
              precision    recall  f1-score   support

           0       0.87      0.68      0.76        50
           1       0.69      0.88      0.77        41

    accuracy                           0.77        91
   macro avg       0.78      0.78      0.77        91
weighted avg       0.79      0.77      0.77        91
```

3. Logistic Regression (LR)

```
# Logistic Regression(LR)
classifierLR=LogisticRegression()

classifierLR.fit(x_train,y_train)
classifierLR.score(x_test, y_test)

print('LogisticRegression LogLoss {score}'.format(score=log_loss(y_test, classifierLR.predict_proba(x_test))))
clfs.append(classifierLR)

# save best model to current working directory
joblib.dump(classifierLR, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_classifierLR = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_classifierLR.predict(x_test)
print('Logistic Regression accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  labels)
ax.set_yticklabels(['']  labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
Logistic Regression accuracy score:  0.8021978021978022

[[36 14]
 [ 4 37]]
```

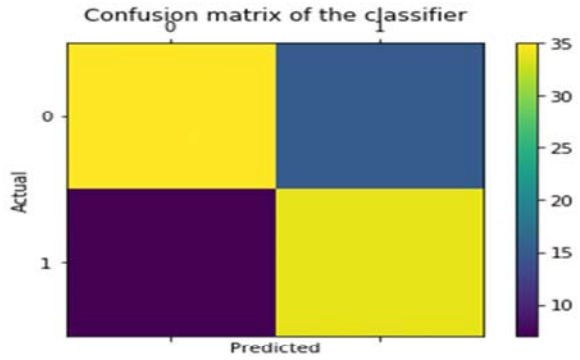



```
              precision    recall  f1-score   support

           0       0.90      0.72      0.80        50
           1       0.73      0.90      0.80        41

    accuracy                           0.80        91
   macro avg       0.81      0.81      0.80        91
weighted avg       0.82      0.80      0.80        91
```

4. Decision Tree (DT)

```
# Decision Tree (DT)
classifierDT=DecisionTreeClassifier(criterion="gini", random_state=50, max_depth=3, min_samples_leaf=5)
classifierDT.fit(x_train,y_train)
classifierDT.score(x_test, y_test)

print('Decision Tree LogLoss {score}'.format(score=log_loss(y_test, classifierDT.predict_proba(x_test))))
clfs.append(classifierDT)

# save best model to current working directory
joblib.dump(classifierDT, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_classifierDT = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_classifierDT.predict(x_test)
print('Decision Tree accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  labels)
ax.set_yticklabels(['']  labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
Decision Tree accuracy score:  0.7582417582417582

[[35 15]
 [ 7 34]]
```



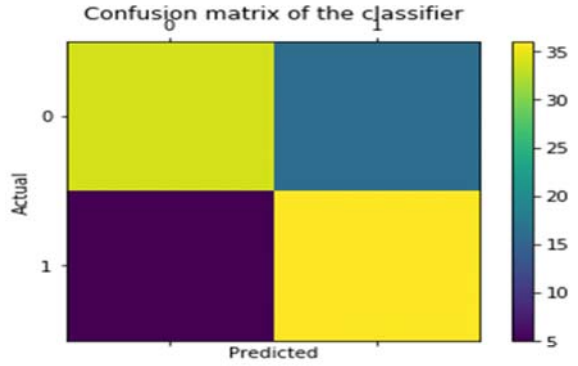
	precision	recall	f1-score	support
0	0.83	0.70	0.76	50
1	0.69	0.83	0.76	41
accuracy			0.76	91
macro avg	0.76	0.76	0.76	91
weighted avg	0.77	0.76	0.76	91

5. Random Forest (RF)

Random Forest is one of the most popular and most powerful machine learning algorithms. It is a type of ensemble machine learning algorithm called Bootstrap Aggregation or bagging.

```
# Random Forest(RF)
classifierRF=RandomForestClassifier()
classifierRF.fit(x_train,y_train)
classifierRF.score(x_test, y_test)
print('RandomForest LogLoss {score}'.format(score=log_loss(y_test, classifierRF.predict_proba(x_test))))
clfs.append(classifierRF)

# save best model to current working directory
joblib.dump(classifierRF, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_classifierRF = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_classifierRF.predict(x_test)
print('Random Forest accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  + labels)
ax.set_yticklabels(['']  + labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
Random Forest accuracy score:  0.7692307692307693


[[34 16]
 [ 5 36]]
```




```
              precision    recall  f1-score   support

           0       0.87      0.68      0.76        50
           1       0.69      0.88      0.77        41

    accuracy                           0.77        91
   macro avg       0.78      0.78      0.77        91
weighted avg       0.79      0.77      0.77        91
```

```
print('Accuracy of svm: {}'.format(grid_svm.score(x_test, y_test)))
print('\n')

print('Accuracy of naive bayes: {}'.format(classifierNB.score(x_test, y_test)))
print('\n')

print('Accuracy of logistic regression: {}'.format(classifierLR.score(x_test, y_test)))
print('\n')

print('Accuracy of decision tree: {}'.format(classifierDT.score(x_test, y_test)))
print('\n')

print('Accuracy of random forest: {}'.format(classifierRF.score(x_test, y_test)))
print('\n')
```

```
Accuracy of svm: 0.8021978021978022
Accuracy of naive bayes: 0.7692307692307693
Accuracy of logistic regression: 0.8021978021978022
Accuracy of decision tree: 0.7582417582417582
Accuracy of random forest: 0.7692307692307693
```

ENSEMBLE LEARNING

Ensemble learning combines diverse set of individual models or classifiers together to improvise the stability and predictive power of the model.

Ensemble Combination Rules

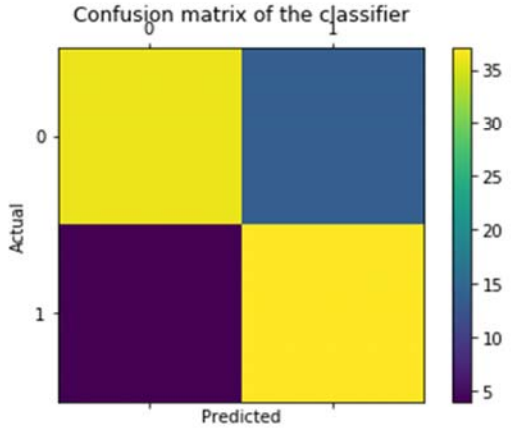
The predictions made by each base learner is given as an input to Majority voting and Weighted Average voting ensemble combination methods.

1. **Majority Voting:** Every base classifier makes a prediction for each test instance and the final output prediction is the one that receives the majority of votes.

```
# Ensemble Majority Voting Classifier
from sklearn.ensemble import VotingClassifier
#create a dictionary of our models
estimators=[('svm', grid_svm), ('nb', classifierNB), ('lr', classifierLR), ('dt', classifierDT),('rf', classifierRF)]

#create our voting classifier, inputting our models
majority_voting = VotingClassifier(estimators, voting='hard')

#fit model to training data
majority_voting.fit(x_train, y_train)
#test our model on the test data
majority_voting.score(x_test, y_test)

# save best model to current working directory
joblib.dump(majority_voting, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model max v = joblib.load("heart disease.pkl" )
```

```
# get predictions from best model above
y_preds = model_max_v.predict(x_test)
print('majority_voting_accuracy: ',majority_voting.score(x_test, y_test))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  + labels)
ax.set_yticklabels(['']  + labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
majority_voting_accuracy:  0.8021978021978022


[[36 14]
 [ 4 37]]
```

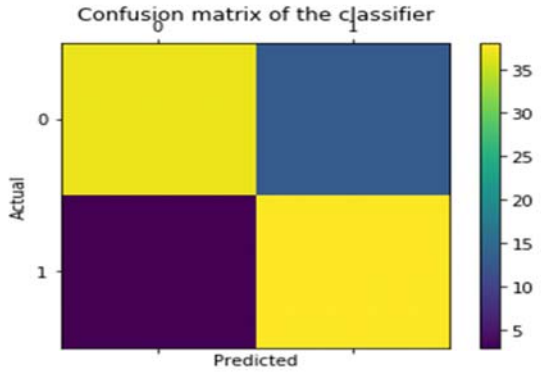



```
              precision    recall  f1-score   support

           0       0.90      0.72      0.80        50
           1       0.73      0.90      0.80        41

    accuracy                           0.80        91
   macro avg       0.81      0.81      0.80        91
weighted avg       0.82      0.80      0.80        91
```

2. **Weighted Average Voting:** All classifiers are assigned different weights defining the importance of each classifier for prediction.

```
#Ensemble Weighted average classifier
#finding the optimum weights
from scipy.optimize import minimize
predictions = []
for clff in clfs:
    predictions.append(clff.predict_proba(x_test))

def log_loss_func(weights):
    final_prediction = 0
    for weight, prediction in zip(weights, predictions):
            final_prediction += weight*prediction

    return log_loss(y_test, final_prediction)

#the algorithms need a starting value, right not we chose 0.5 for all weights
#its better to choose many random starting points and run minimize a few times
starting_values = [0.5]*len(predictions)

#adding constraints  and a different solver as suggested by user 16universe
cons = ({'type':'eq','fun':lambda w: 1-sum(w)})
#our weights are bound between 0 and 1
bounds = [(0,1)]*len(predictions)

res = minimize(log_loss_func, starting_values, method='SLSQP', bounds=bounds, constraints=cons)

print('ensamble score: {best_score}'.format(best_score=res['fun']))
print('Best Weights: {weights}'.format(weights=res['x']))

weighted_avg = VotingClassifier(estimators, voting='soft',weights=res['x']).fit(x_train, y_train)
print('The accuracy weighted average classifier is :', weighted_avg.score(x_test,y_test))

# save best model to current working directory
joblib.dump(weighted_avg, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_w_avg = joblib.load("heart_disease.pkl" )
# get predictions from best model above
y_preds = model_w_avg.predict(x_test)
print('weighted_average_accuracy: ',weighted_avg.score(x_test, y_test))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
weighted_average_accuracy:  0.8241758241758241

[[37 13]
 [ 3 38]]
```



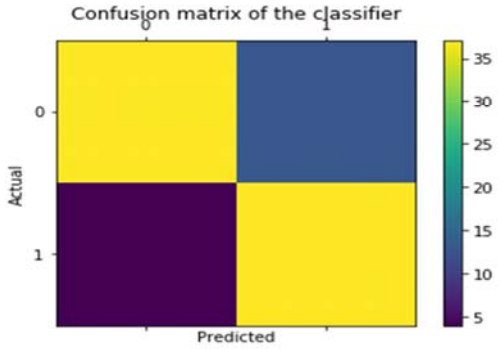
	precision	recall	f1-score	support
0	0.93	0.74	0.82	50
1	0.75	0.93	0.83	41
accuracy			0.82	91
macro avg	0.84	0.83	0.82	91
weighted avg	0.84	0.82	0.82	91

Ensemble Classifiers

1. **Bagging:** Bagging or Bootstrap aggregating creates several subsets of data from training sample chosen randomly with replacement. Each collection of subset data is used to train their decision trees. As a result, we get an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree classifier.

```
# Bagging
classifierBa= BaggingClassifier(max_samples=0.5, max_features=1.0, n_estimators=50)
classifierBa.fit(x_train,y_train)
classifierBa.score(x_test, y_test)

# save best model to current working directory
joblib.dump(classifierBa, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_bagging = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_bagging.predict(x_test)
print('bagging_accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  labels)
ax.set_yticklabels(['']  labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
bagging_accuracy score:  0.8131868131868132

[[37 13]
 [ 4 37]]
```



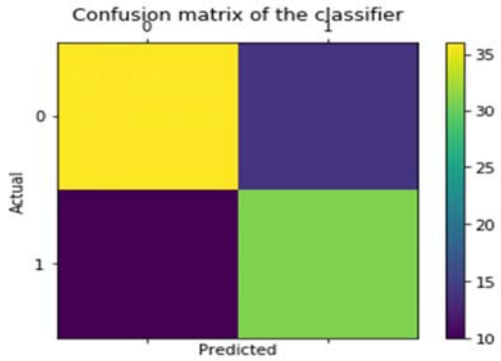
	precision	recall	f1-score	support
0	0.90	0.74	0.81	50
1	0.74	0.90	0.81	41
accuracy			0.81	91
macro avg	0.82	0.82	0.81	91
weighted avg	0.83	0.81	0.81	91

2. **Boosting:** Boosting is used to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analyzing data for errors. Consecutive trees (random sample) are fit and at every step, the goal is to improve the accuracy from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing model.

- **Adaptive Boosting (AdaBoost)**

```
## Boosting
#1.AdaBoost Classifier
classifierAdaBoost= AdaBoostClassifier(n_estimators=500)
classifierAdaBoost.fit(x_train,y_train)
classifierAdaBoost.score(x_test, y_test)

# save best model to current working directory
joblib.dump(classifierAdaBoost, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_Ada_boost = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_Ada_boost.predict(x_test)
print('Ada_boost_accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  labels)
ax.set_yticklabels(['']  labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
Ada_boost_accuracy score:  0.7362637362637363

[[36 14]
 [10 31]]
```



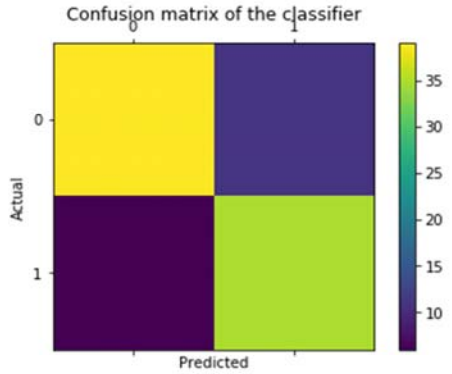
	precision	recall	f1-score	support
0	0.78	0.72	0.75	50
1	0.69	0.76	0.72	41
accuracy			0.74	91
macro avg	0.74	0.74	0.74	91
weighted avg	0.74	0.74	0.74	91

- **Gradient Boosting**

```
#2. GradientBoosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
classifierGBo= GradientBoostingClassifier(n_estimators=500, learning_rate=1.0, max_depth=1)

classifierGBo.fit(x_train,y_train)
classifierGBo.score(x_test, y_test)

# save best model to current working directory
joblib.dump(classifierGBo, "heart_disease.pkl")

# load from file and predict using the best configs found in the CV step
model_Gradient_boosting = joblib.load("heart_disease.pkl" )

# get predictions from best model above
y_preds = model_Gradient_boosting.predict(x_test)
print('Gradient_boosting_accuracy score: ',accuracy_score(y_test, y_preds))
print('\n')

import pylab as plt
labels=[0,1]
cmx=confusion_matrix(y_test,y_preds, labels)
print(cmx)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cmx)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels(['']  labels)
ax.set_yticklabels(['']  labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print('\n')
print(classification_report(y_test, y_preds))
```

```
Gradient_boosting_accuracy score:  0.8131868131868132
```

```
[[39 11]
 [ 6 35]]
```




```
              precision    recall  f1-score   support

           0       0.87      0.78      0.82        50
           1       0.76      0.85      0.80        41

    accuracy                           0.81        91
   macro avg       0.81      0.82      0.81        91
weighted avg       0.82      0.81      0.81        91
```

```
print('\n')
print('Majority Voting accuracy score: ',majority_voting.score(x_test, y_test))
print('Weighted Average accuracy score: ',weighted_avg.score(x_test, y_test))
print('Bagging_accuracy score: ',classifierBa.score(x_test, y_test))
print('Ada_boost_accuracy score: ',classifierAdaBoost.score(x_test, y_test))
print('Gradient_boosting_accuracy score: ',classifierGBo.score(x_test, y_test))
```

```
Majority Voting accuracy score:  0.8021978021978022
Weighted Average accuracy score:  0.8241758241758241
Bagging_accuracy score:  0.8131868131868132
Ada_boost_accuracy score:  0.7362637362637363
Gradient_boosting_accuracy score:  0.8131868131868132
```

Conclusion

This work attempts to predict the presence of heart disease by utilizing Base learners, Ensemble combination rules and Ensemble classifiers. It was noticed that Accuracy of Weighted Average Classifier (82.417%) was the highest.

Thank You!