

**A PROJECT REPORT**  
**ON**  
**HEART DISEASE PREDICTION USING MACHINE LEARNING**

Submitted in partial fulfillment for the requirement of the award of

TRAINING

IN

Data Analytics, Machine Learning and AI using Python



*Submitted By*

**VIMALAADITHAN B S**

(Puducherry Technological University)

## **Abstract:**

In recent times, Heart Disease prediction is one of the most complicated tasks in the medical field. In the modern era, approximately one person dies per minute due to a cardiac problem. Data science plays a crucial role in processing a huge amount of data in the field of healthcare. As heart disease prediction is a complex task, there is a need to automate the prediction process to avoid risks associated with it and alert the patient well in advance. The proposed work predicts the chances of heart disease and classifies patient's risk levels by implementing different data mining techniques such as Naive Bayes, Decision Tree, Logistic Regression, and Random Forest. Thus, this paper presents a comparative study by analyzing the performance of different machine learning algorithms. The trial results verify that the Random Forest algorithm has achieved the highest accuracy of 84% compared to other Machine Learning algorithms implemented.

**Keywords**— Decision Tree, Naive Bayes, Logistic Regression, Random Forest, Heart Disease Prediction, Machine Learning, Data visualization.

## **Introduction:**

According to the World Health Organization, every year 12 million deaths occur worldwide due to heart disease. The load of cardiovascular disease is rapidly increasing all over the world over the past few years. Heart Disease is even highlighted as a silent killer which leads to the death of the person without obvious symptoms. The early diagnosis of heart disease plays a vital role in making decisions on lifestyle changes in high-risk patients and in turn, reduces the complications. This project aims to predict future heart disease by analyzing data of patients who classify whether they have heart disease or not using machine-learning algorithms.

## **Problem:**

The major and important challenge in heart disease is its detection. There are instruments available that can predict heart disease but either they are too expensive to afford or lack the efficiency to calculate the chance of heart disease. Early detection of cardiac diseases can decrease the mortality rate and its overall complications. However, it is not possible to monitor patients every day in all cases accurately and consultation of a patient for 24 hours by a doctor is not available since it requires more sapience, time, and expertise.

Since we have a good amount of data in today's world, we can use various machine learning algorithms to analyze the data for hidden patterns. The hidden patterns can be used for health diagnosis in medicinal data.

## Objective:

The main objectives of developing this project are to,

- **Predict** whether a patient should be diagnosed with heart disease. This is a **binary** outcome.
- **Positive** (+) = 1, patient diagnosed with heart disease.
- **Negative** (-) = 0, patient not diagnosed with heart disease.
- Experiment with various **Classification Models** & see which yields the greatest **accuracy**.
- Examine **trends & correlations** within our data
- Determine which **features** are **most important** to Positive/Negative Heart Disease diagnosis.

## Dataset and Its Attributes:

We have used the already processed UCI Cleveland dataset available on the Kaggle website for our analysis. The complete description of the 14 attributes used in the proposed work is as mentioned below:

1. **age** – Age of the patient.
2. **sex**: 1= Male, 0= Female (*Binary*)
3. **cp** – chest pain type (4 values -*Ordinal*):
  - a. Value 1: typical angina
  - b. Value 2: atypical angina
  - c. Value 3: non-anginal pain
  - d. Value 4: asymptomatic
4. **trestbtp** – resting blood pressure.
5. **chol** – serum cholesterol in mg/dl.
6. **fbs** – fasting blood sugar > 120 mg/dl (*Binary*) (1 = true; 0 = false)
7. **restecg** – resting electrocardiography results (values 0,1,2)
8. **thalach** – maximum heart rate achieved.
9. **exang** – exercise-induced angina (*Binary*) (1 = yes; 0 = no)
10. **oldpeak** – ST depression induced by exercise relative to rest.
11. **slope** – the slope of the peak exercise ST segment (*Ordinal*)
  - Value 1: up sloping
  - Value 2: flat
  - Value 3: down sloping
12. **ca** – number of major vessels (0–3, *Ordinal*) colored by fluoroscopy.
13. **thal** – Thalassemia defect (*Ordinal*):
  - 1 = fixed defect;
  - 2 = reversible defect;
  - 3 = normal;
14. **target** – represents the diagnosis of heart disease (*binary*)
  - 1 = patient has heart disease;
  - 0 = patient does not have heart disease

Note: Our data has 3 types of data:

**Continuous Data:** which is quantitative data that can be measured.

**Ordinal Data:** Categorical data that has an order to it (0,1,2,3, etc)

**Binary Data:** data whose unit can take on only two possible states (0 & 1)

## Methodologies:

### 1. Loading and Visualizing the dataset:

First, let's import all the libraries required for our project such as NumPy, pandas, matplotlib, sklearn, etc, and load the dataset and see how the data is distributed among various attributes. This would help us get an idea of what we are working with.

```
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
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
[ ] # loading the dataset
```

```
heart_data = pd.read_csv('heart_data.csv')
heart_data.head()
```

|   | age | sex | cp | trestbps | chol | fb | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|-----|-----|----|----------|------|----|---------|---------|-------|---------|-------|----|------|--------|
| 0 | 63  | 1   | 3  | 145      | 233  | 1  | 0       | 150     | 0     | 2.3     | 0     | 0  | 1    | 1      |
| 1 | 37  | 1   | 2  | 130      | 250  | 0  | 1       | 187     | 0     | 3.5     | 0     | 0  | 2    | 1      |
| 2 | 41  | 0   | 1  | 130      | 204  | 0  | 0       | 172     | 0     | 1.4     | 2     | 0  | 2    | 1      |
| 3 | 56  | 1   | 1  | 120      | 236  | 0  | 1       | 178     | 0     | 0.8     | 2     | 0  | 2    | 1      |
| 4 | 57  | 0   | 0  | 120      | 354  | 0  | 1       | 163     | 1     | 0.6     | 2     | 0  | 2    | 1      |

Now, let see the shape of the data and summarize the count, mean, standard deviation, min, and max for numeric variables.

```
heart_data.shape
```

```
(303, 14)
```

It has 303 rows and 14 columns.

|       | age        | sex        | cp         | trestbps   | chol       | fbs        | restecg    | thalach    | exang      | oldpeak    | slope      | ca         | thal       | target     |
|-------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| count | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 |
| mean  | 54.366337  | 0.683168   | 0.966997   | 131.623762 | 246.264026 | 0.148515   | 0.528053   | 149.646865 | 0.326733   | 1.039604   | 1.399340   | 0.729373   | 2.313531   | 0.544554   |
| std   | 9.082101   | 0.466011   | 1.032052   | 17.538143  | 51.830751  | 0.356198   | 0.525860   | 22.905161  | 0.469794   | 1.161075   | 0.616226   | 1.022606   | 0.612277   | 0.498835   |
| min   | 29.000000  | 0.000000   | 0.000000   | 94.000000  | 126.000000 | 0.000000   | 0.000000   | 71.000000  | 0.000000   | 0.000000   | 0.000000   | 0.000000   | 0.000000   | 0.000000   |
| 25%   | 47.500000  | 0.000000   | 0.000000   | 120.000000 | 211.000000 | 0.000000   | 0.000000   | 133.500000 | 0.000000   | 0.000000   | 1.000000   | 0.000000   | 2.000000   | 0.000000   |
| 50%   | 55.000000  | 1.000000   | 1.000000   | 130.000000 | 240.000000 | 0.000000   | 1.000000   | 153.000000 | 0.000000   | 0.800000   | 1.000000   | 0.000000   | 2.000000   | 1.000000   |
| 75%   | 61.000000  | 1.000000   | 2.000000   | 140.000000 | 274.500000 | 0.000000   | 1.000000   | 166.000000 | 1.000000   | 1.600000   | 2.000000   | 1.000000   | 3.000000   | 1.000000   |
| max   | 77.000000  | 1.000000   | 3.000000   | 200.000000 | 564.000000 | 1.000000   | 2.000000   | 202.000000 | 1.000000   | 6.200000   | 2.000000   | 4.000000   | 3.000000   | 1.000000   |

summarizes the count, mean, standard deviation, min, and max for numeric variables.

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
#   Column      Non-Null Count  Dtype
---  -
0   age         303 non-null    int64
1   sex         303 non-null    int64
2   cp          303 non-null    int64
3   trestbps    303 non-null    int64
4   chol        303 non-null    int64
5   fbs         303 non-null    int64
6   restecg     303 non-null    int64
7   thalach     303 non-null    int64
8   exang       303 non-null    int64
9   oldpeak     303 non-null    float64
10  slope       303 non-null    int64
11  ca          303 non-null    int64
12  thal        303 non-null    int64
13  target      303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB

```

Display the information of the dataset and null values if any for each column. We have perfect data that doesn't have any null value.

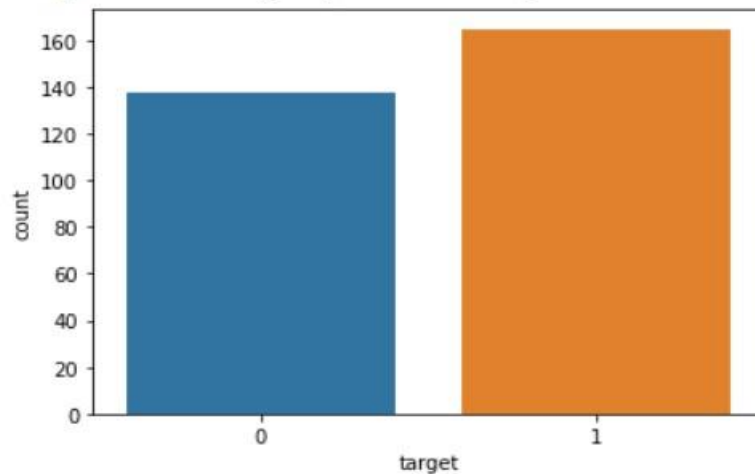
Let's see if there is a good proportion between our positive & negative binary target variables. We will be able to train the model more accurately if the target variable is distributed almost equally.

```

1    165
0    138
Name: target, dtype: int64

```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fd402faf290>



It appears we have a good balance between the two binary outputs.

## 2. Exploratory Data Analysis:

**Correlation Matrix** - let us see the correlations between all variables.

|          | age       | sex       | cp        | trestbps  | chol      | fbs       | restecg   | thalach   | exang     | oldpeak   | slope     | ca        | thal      | target    |
|----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| age      | 1.000000  | -0.098447 | -0.068653 | 0.279351  | 0.213678  | 0.121308  | -0.116211 | -0.398522 | 0.096801  | 0.210013  | -0.168814 | 0.276326  | 0.068001  | -0.225439 |
| sex      | -0.098447 | 1.000000  | -0.049353 | -0.056769 | -0.197912 | 0.045032  | -0.058196 | -0.044020 | 0.141664  | 0.096093  | -0.030711 | 0.118261  | 0.210041  | -0.280937 |
| cp       | -0.068653 | -0.049353 | 1.000000  | 0.047608  | -0.076904 | 0.094444  | 0.044421  | 0.295762  | -0.394280 | -0.149230 | 0.119717  | -0.181053 | -0.161736 | 0.433798  |
| trestbps | 0.279351  | -0.056769 | 0.047608  | 1.000000  | 0.123174  | 0.177531  | -0.114103 | -0.046698 | 0.067616  | 0.193216  | -0.121475 | 0.101389  | 0.062210  | -0.144931 |
| chol     | 0.213678  | -0.197912 | -0.076904 | 0.123174  | 1.000000  | 0.013294  | -0.151040 | -0.009940 | 0.067023  | 0.053952  | -0.004038 | 0.070511  | 0.098803  | -0.085239 |
| fbs      | 0.121308  | 0.045032  | 0.094444  | 0.177531  | 0.013294  | 1.000000  | -0.084189 | -0.008567 | 0.025665  | 0.005747  | -0.059894 | 0.137979  | -0.032019 | -0.028046 |
| restecg  | -0.116211 | -0.058196 | 0.044421  | -0.114103 | -0.151040 | -0.084189 | 1.000000  | 0.044123  | -0.070733 | -0.058770 | 0.093045  | -0.072042 | -0.011981 | 0.137230  |
| thalach  | -0.398522 | -0.044020 | 0.295762  | -0.046698 | -0.009940 | -0.008567 | 0.044123  | 1.000000  | -0.378812 | -0.344187 | 0.386784  | -0.213177 | -0.096439 | 0.421741  |
| exang    | 0.096801  | 0.141664  | -0.394280 | 0.067616  | 0.067023  | 0.025665  | -0.070733 | -0.378812 | 1.000000  | 0.288223  | -0.257748 | 0.115739  | 0.206754  | -0.436757 |
| oldpeak  | 0.210013  | 0.096093  | -0.149230 | 0.193216  | 0.053952  | 0.005747  | -0.058770 | -0.344187 | 0.288223  | 1.000000  | -0.577537 | 0.222682  | 0.210244  | -0.430696 |
| slope    | -0.168814 | -0.030711 | 0.119717  | -0.121475 | -0.004038 | -0.059894 | 0.093045  | 0.386784  | -0.257748 | -0.577537 | 1.000000  | -0.080155 | -0.104764 | 0.345877  |
| ca       | 0.276326  | 0.118261  | -0.181053 | 0.101389  | 0.070511  | 0.137979  | -0.072042 | -0.213177 | 0.115739  | 0.222682  | -0.080155 | 1.000000  | 0.151832  | -0.391724 |
| thal     | 0.068001  | 0.210041  | -0.161736 | 0.062210  | 0.098803  | -0.032019 | -0.011981 | -0.096439 | 0.206754  | 0.210244  | -0.104764 | 0.151832  | 1.000000  | -0.344029 |
| target   | -0.225439 | -0.280937 | 0.433798  | -0.144931 | -0.085239 | -0.028046 | 0.137230  | 0.421741  | -0.436757 | -0.430696 | 0.345877  | -0.391724 | -0.344029 | 1.000000  |

```

corr = heart_data.corr()
plt.subplots(figsize=(15,10))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap=sns.diverging_palette(220, 20, as_cmap=True))
sns.heatmap(corr, xticklabels=corr.columns,
            yticklabels=corr.columns,
            annot=True,
            cmap=sns.diverging_palette(220, 20, as_cmap=True))

```



We can see there is a **positive correlation** between chest pain (cp) & the target variable. This makes sense since the greater amount of chest pain results in a greater chance of having heart disease. Cp (chest pain), is an ordinal feature with 4 values: Value 1: typical angina, Value 2: atypical angina, Value 3: non-anginal pain, Value 4: asymptomatic.

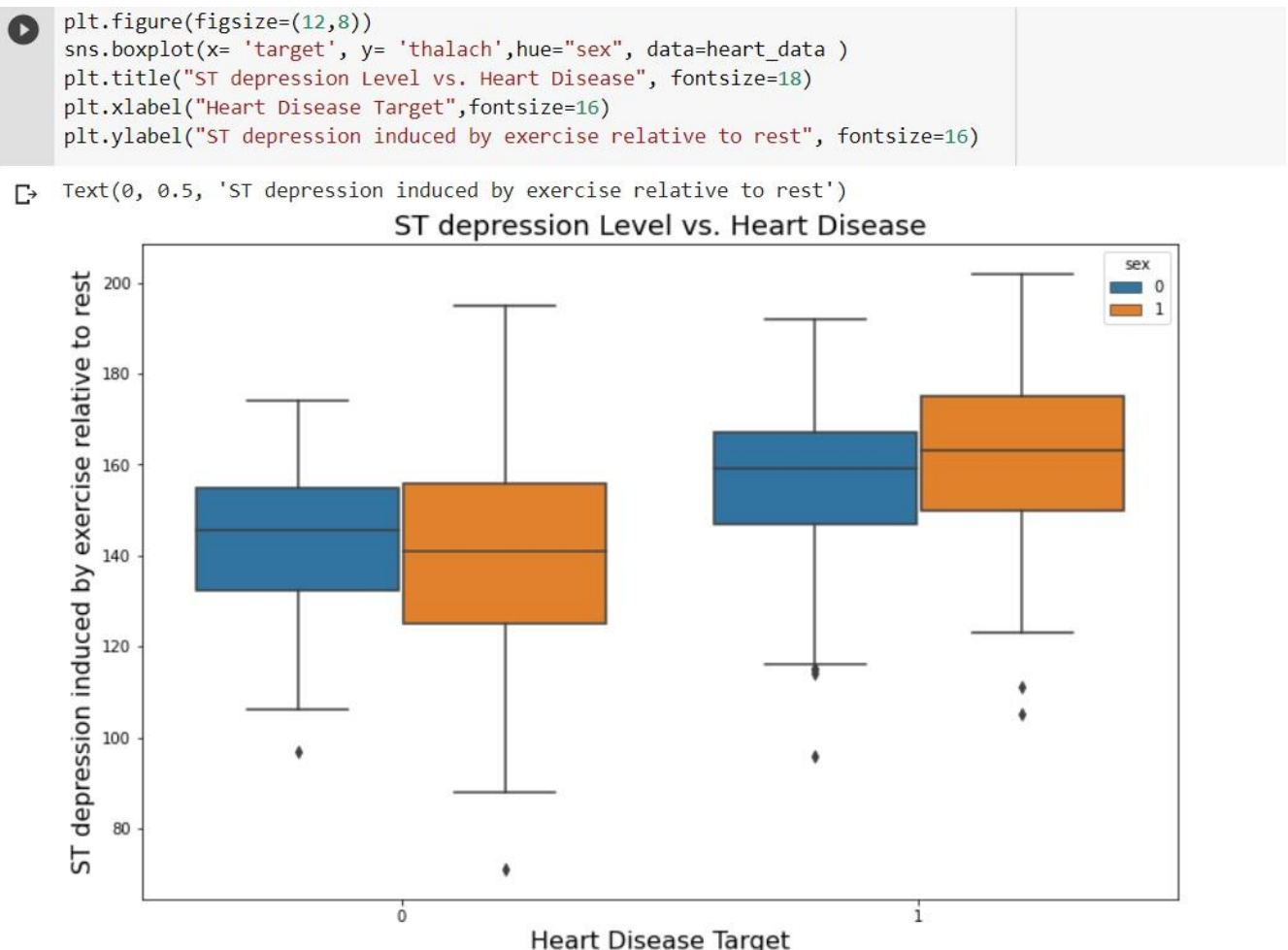
In addition, we see a **negative correlation** between exercise-induced angina (exang) and our target variable. This makes sense because when you exercise, your heart requires more blood, but narrowed arteries slow down blood flow.

From this, we can see how each column and attribute of the data is correlated with the target variable.

## Box Plots

The advantages of showing the Box plots are that it shows the *basic statistics* of the data, as well as its *distribution*. These plots are often used to compare the distribution of a given variable across some categories.

It shows the median, IQR, & Tukey's fence. (Minimum, first quartile (Q1), median, third quartile (Q3), and maximum). In addition, it can provide us with outliers in our data.



Positive patients exhibit a heightened median for ST depression level, while negative patients have lower levels. In addition, we don't see many differences between male & female target outcomes, except for the fact that males have slightly larger ranges of ST Depression.

From comparing positive and negative patients we can see there are vast differences in means for many of our 13 attributes. From examining the details, we can observe that positive patients experience heightened maximum heart rate achieved (thalach) average. In addition, positive patients exhibit about 1/3rd the amount of ST depression induced by exercise relative to rest (oldpeak).

### 3. Prepare Data for Modelling:

To prepare data for modeling,

- **Assign** all the 13 attributes (i.e., feature variables) to X, & the last column (i.e., target variable) to our classification predictor, Y.
- **Split** the data set into the Training set and Test set.
- **Normalizing** the data will transform the data so that its distribution will have a mean of 0 and a standard deviation of 1.

```
[9] # Splitting of Feature and Target variable

X = heart_data.drop(columns='target', axis=1) # Feature variable
Y = heart_data['target']                    # Target variable

# Dividing the dataset into training and test data

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.18, stratify=Y, random_state=1)

[10] from sklearn.preprocessing import StandardScaler
      sc = StandardScaler()
      X_train = sc.fit_transform(X_train)
      X_test = sc.transform(X_test)

[11] print(X.shape, X_train.shape, X_test.shape)

(303, 13) (248, 13) (55, 13)
```

### 4. Training the Model:

Now we'll Train various Classification Models on the Training set & see which yields the highest accuracy. We will compare the training data with different Machine Learning Algorithms such as Logistic Regression, SVM (Support Vector Machine), Naive Bayes Classifier, Decision Trees, and Random Forest. All these algorithms are supervised, learning models. The training dataset is the dataset that is used to train a model. The testing dataset is used to check the performance of the trained model.

For each of the algorithms, the performance is computed and analyzed based on different metrics used such as accuracy, precision, recall, and F-measure scores as described further.

## Model 1: Logistic Regression:

Logistic Regression is a classification algorithm mostly used for binary classification problems. In logistic regression instead of fitting a straight line or hyperplane, the logistic regression algorithm uses the logistic function to squeeze the output of a linear equation between 0 and 1. There are 13 independent variables that make logistic regression good for classification.

```
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, Y_train) # Training the model with training data
```

```
[21] X_train_pred = model.predict(X_train)
training_accuracy = accuracy_score(X_train_pred, Y_train)
print("Accuracy of Training data: ",training_accuracy) # Accuracy of training data
```

Accuracy of Training data: 0.8709677419354839

```
[23] X_test_pred = model.predict(X_test)
test_accuracy = accuracy_score(X_test_pred, Y_test)

print("Accuracy of Test data: ",test_accuracy,"\n") # Accuracy of test data
print(classification_report(Y_test, X_test_pred)) # output accuracy
```

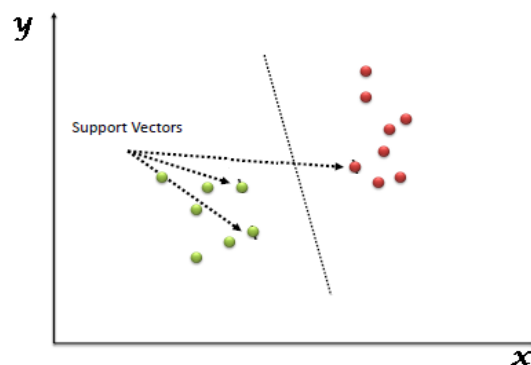
Accuracy of Test data: 0.8181818181818182

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.80      | 0.80   | 0.80     | 25      |
| 1            | 0.83      | 0.83   | 0.83     | 30      |
| accuracy     |           |        | 0.82     | 55      |
| macro avg    | 0.82      | 0.82   | 0.82     | 55      |
| weighted avg | 0.82      | 0.82   | 0.82     | 55      |

Accuracy 82%

## Model 2: SVM (Support Vector Machine):

"Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is several features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

```
from sklearn.metrics import classification_report
from sklearn.svm import SVC

model2 = SVC(random_state=1) # get instance of model
model2.fit(X_train, Y_train) # Train/Fit model

Y_pred2 = model2.predict(X_test) # get y predictions
test_accuracy = accuracy_score(Y_pred2, Y_test)

print("Accuracy of Test data: ", test_accuracy, "\n")
print(classification_report(Y_test, Y_pred2)) # output accuracy
```

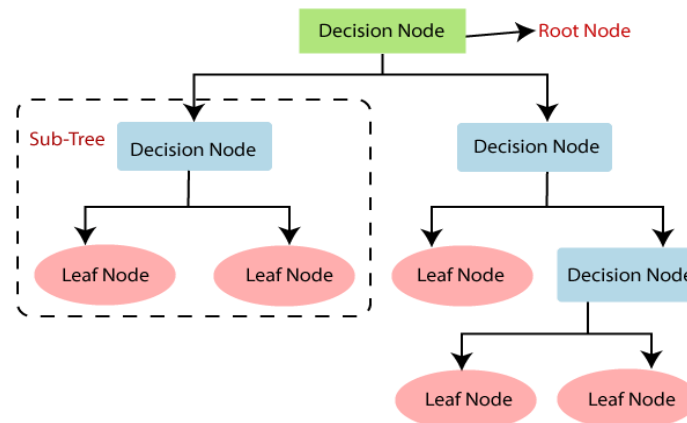
➡ Accuracy of Test data: 0.8

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.82      | 0.72   | 0.77     | 25      |
| 1            | 0.79      | 0.87   | 0.83     | 30      |
| accuracy     |           |        | 0.80     | 55      |
| macro avg    | 0.80      | 0.79   | 0.80     | 55      |
| weighted avg | 0.80      | 0.80   | 0.80     | 55      |

Accuracy 80%

## Model 3: Decision Trees:

The Decision Tree algorithm is in the form of a flowchart where the inner node represents the dataset attributes and the outer branches are the outcome. Decision Tree is chosen because they are fast, reliable, easy to interpret, and very little data preparation is required. In the Decision Tree, the prediction of class labels originates from the root of the tree. The value of the root attribute is compared to the record's attribute. As the result of the comparison, the corresponding branch is followed to that value, and a jump is made to the next node.



```
[31] from sklearn.tree import DecisionTreeClassifier

model3 = DecisionTreeClassifier(random_state=1) # get instance of model
model3.fit(X_train, Y_train) # Train/Fit model

Y_pred3 = model3.predict(X_test) # get y predictions
test_accuracy = accuracy_score( Y_pred3, Y_test)

print("Accuracy of Test data: ",test_accuracy,"\n")
print(classification_report(Y_test, Y_pred3)) # output accuracy
```

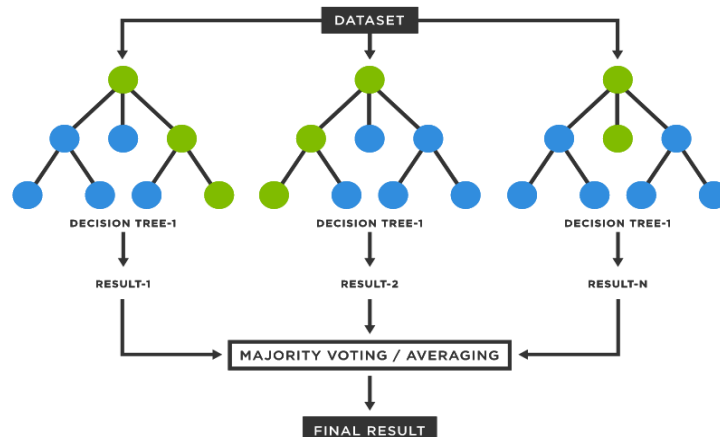
Accuracy of Test data: 0.7818181818181819

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.74      | 0.80   | 0.77     | 25      |
| 1            | 0.82      | 0.77   | 0.79     | 30      |
| accuracy     |           |        | 0.78     | 55      |
| macro avg    | 0.78      | 0.78   | 0.78     | 55      |
| weighted avg | 0.78      | 0.78   | 0.78     | 55      |

Accuracy 78%

## Model 4: Random Forest

Random Forest algorithms are used for classification as well as regression. It creates a tree for the data and makes a prediction based on that. Random Forest algorithm can be used on large datasets and can produce the same result even when large sets of record values are missing. The generated samples from the decision tree can be saved so that they can be used on other data. In the random forest there are two stages, firstly create a random forest then make a prediction using a random forest classifier created in the first stage.



```
[45] from sklearn.ensemble import RandomForestClassifier

model4 = RandomForestClassifier()# get instance of model
model4.fit(X_train, Y_train) # Train/Fit model

Y_pred4 = model4.predict(X_test) # get y predictions
test_accuracy = accuracy_score( Y_pred4, Y_test)

print("Accuracy of Test data: ",test_accuracy,"\n")
print(classification_report(Y_test, Y_pred4)) # output accuracy
```

Accuracy of Test data: 0.8363636363636363

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.83      | 0.80   | 0.82     | 25      |
| 1            | 0.84      | 0.87   | 0.85     | 30      |
| accuracy     |           |        | 0.84     | 55      |
| macro avg    | 0.84      | 0.83   | 0.83     | 55      |
| weighted avg | 0.84      | 0.84   | 0.84     | 55      |

Accuracy 84% !

## Model 5: Naives Bayes Classifier

Naïve Bayes algorithm is based on the Bayes rule. The independence between the attributes of the dataset is the main assumption and the most important in making a classification. It is easy and fast to predict and holds best when the assumption of independence holds. Bayes' theorem calculates the posterior probability of an event (A) given some prior probability of event B represented by  $P(A/B)$  as shown in equation 1:

$$P(A|B) = (P(B|A) P(A)) / P(B)$$

```
[47] from sklearn.naive_bayes import GaussianNB

model5 = GaussianNB() # get instance of model
model5.fit(X_train, Y_train) # Train/Fit model

Y_pred5 = model5.predict(X_test) # get y predictions
test_accuracy = accuracy_score( Y_pred5, Y_test)

print("Accuracy of Test data: ",test_accuracy,"\n")
print(classification_report(Y_test, Y_pred5)) # output accuracy
```

Accuracy of Test data: 0.8

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.79      | 0.76   | 0.78     | 25      |
| 1            | 0.81      | 0.83   | 0.82     | 30      |
| accuracy     |           |        | 0.80     | 55      |
| macro avg    | 0.80      | 0.80   | 0.80     | 55      |
| weighted avg | 0.80      | 0.80   | 0.80     | 55      |

Accuracy 80% !

While comparing all the training models, we can conclude that **the Random Forest** algorithm yields the **highest accuracy**. With an accuracy of **84%**.

**Precision:** defines how many are correctly classified among that class.

**Recall:** defines how many of this class you find over the whole number of elements of this class.

**F1-score:** harmonic mean of precision and recall values.

F1 score reaches its best value at 1 and worst value at 0.

$$F1 \text{ Score} = 2 \times ((\text{precision} \times \text{recall}) / (\text{precision} + \text{recall}))$$

**Support:** number of samples of the **true** response that lie in that class.

## 5. Making the Confusion Matrix

```
✓ [48] from sklearn.metrics import confusion_matrix, accuracy_score
0s c_m = confusion_matrix(Y_test, Y_pred4)
    print(c_m)
    accuracy_score(Y_test, Y_pred4)

[[20  5]
 [ 4 26]]
0.8363636363636363
```

Any accuracy above 70% is considered good.

20 is the number of True Positives in our data, while 26 is the number of True Negatives.

5 & 4 are the number of errors.

- There are 9 Type 1 errors (False Positives)- You predicted positive and it's false.
- There are 3 Type 2 errors (False Negatives)- You predicted negative and it's false.

**Accuracy = number of Correctly Predicted values / number of Total predicted values.**

TP - number of true positives

FN - number of false negatives

FP - number of false positives

TN – number of true negatives.

**Accuracy** =  $(TP + TN) / (TP + TN + FP + FN)$ .

**Accuracy** =  $(20+26) / (20+26+5+4) = 0.8363$

**Accuracy = 83.63%**

## 6. Feature Importance:

Feature Importance provides a score that indicates how helpful each feature was in our model.

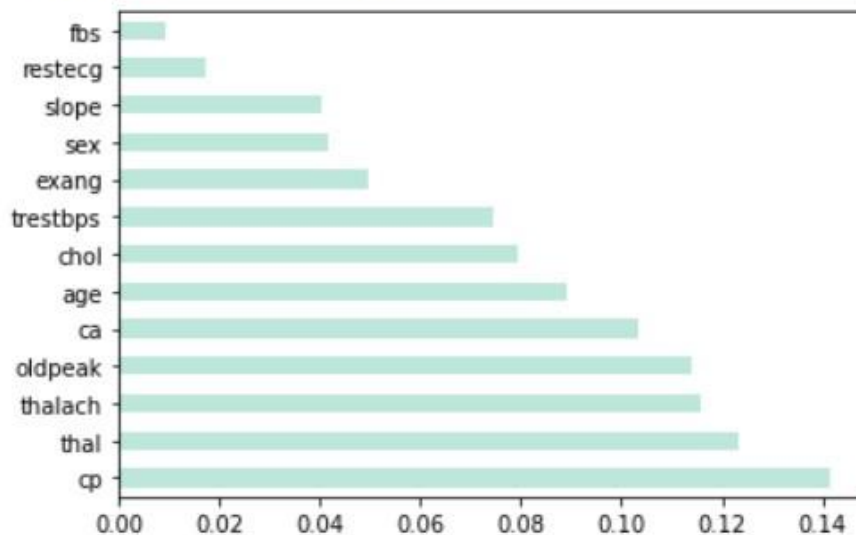
The higher the Feature Score, the more that feature is used to make key decisions & thus the more important it is.

✓  
0s

```
importance = model4.feature_importances_  
  
for i,v in enumerate(importance):  
    print('Feature: %0d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.08930  
Feature: 1, Score: 0.04178  
Feature: 2, Score: 0.14139  
Feature: 3, Score: 0.07459  
Feature: 4, Score: 0.07947  
Feature: 5, Score: 0.00925  
Feature: 6, Score: 0.01723  
Feature: 7, Score: 0.11586  
Feature: 8, Score: 0.04980  
Feature: 9, Score: 0.11417  
Feature: 10, Score: 0.04052  
Feature: 11, Score: 0.10350  
Feature: 12, Score: 0.12314
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7ffa81bb9490>



From the Feature Importance graph above, we can conclude that the top 4 significant features were chest pain type (cp), maximum heart rate achieved (thalach), number of major vessels (ca), and ST depression induced by exercise relative to rest (oldpeak).

## 7. Predictions:

**Scenario:** A patient develops cardiac symptoms and we input his vitals into the Machine Learning Algorithm.

- A **34-year-old** male, with a **chest pain** value of **2** (atypical angina), with **resting blood pressure** of **110**.
- he has **serum cholesterol** of **240 mg/dl**.
- He has **fasting blood sugar** > **124 mg/dl**.
- He has a **resting electrocardiographic result** of **1**.
- The patient's **maximum heart rate** achieved is **163**.
- Also, he was **exercise-induced angina**.
- His **ST depression** induced by exercise relative to rest value was **2.2**.
- The **slope** of the peak exercise ST segment is **flat**.
- He has **no major vessels** colored by fluoroscopy, and in addition, his **thalassemia defect** is **reversible**.

Based on this information, we can classify this patient whether he has a positive or negative diagnosis of heart disease.

```
✓ [75] input = model4.predict(sc.transform([[34,1,2,110,240,1,1,163,1,2.2,2,0,2]]))  
1s print(input)
```

```
[1]
```

1 - Positive Diagnosis of Heart Disease

## Conclusion:

1. Out of the 13 features we examined, the top 4 significant features that helped us classify between a positive & negative Diagnosis were chest pain type (cp), maximum heart rate achieved (thalach), number of major vessels (ca), and ST depression induced by exercise relative to rest (oldpeak).

2. Our machine learning algorithm can now classify patients with heart disease. Now we can properly diagnose patients, & get them the help they need to recover. By diagnosing detecting these features early, we may prevent worse symptoms from arising later.

3. Our Random Forest algorithm yields the highest accuracy, 84%. Any accuracy above 70% is considered good, 80% is the ideal accuracy!

## Project Works:

Find my entire project work on the following links:

Source code: [https://github.com/vimal-11/cardiac\\_disease\\_prediction-ML](https://github.com/vimal-11/cardiac_disease_prediction-ML)

Web-application: <https://cardiac-disease-prediction.herokuapp.com>

## References:

1. <https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/>
2. <https://www.cdc.gov/ncbddd/thalassemia/>
3. <https://www.kaggle.com/ronitf/heart-disease-uci>
4. <https://archive.ics.uci.edu/ml/datasets/Heart+Disease>
5. [https://www.researchgate.net/publication/319393368\\_Heart\\_Disease\\_Diagnosis\\_and\\_Prediction\\_Using\\_Machine\\_Learning\\_and\\_Data\\_Mining\\_Techniques\\_A\\_Review](https://www.researchgate.net/publication/319393368_Heart_Disease_Diagnosis_and_Prediction_Using_Machine_Learning_and_Data_Mining_Techniques_A_Review)
6. <https://devcenter.heroku.com/>
7. <https://www.tibco.com/reference-center/what-is-a-random-forest>
8. AH Chen, SY Huang, PS Hong, CH Cheng, and EJ Lin, 2011, "HDPS: Heart Disease Prediction System", Computing in Cardiology, ISSN: 0276-6574, pp.557- 560.
9. Uma. K, M. Hanumathappa, "Heart Disease Prediction Using Classification Techniques with Feature Selection Method", Adarsh Journal of Information Technology, Volume-5, Issue-2, pp.22-29, 2016.
10. V.Subha, M.Revathi, D.Murugan, "Comparative Analysis of Support Vector Machine Ensembles for Heart Disease Prediction", International Journal of Computer Science & Communication Networks, Volume-5(6), pp.386-390.