Project – 2

Heart disease classification using Machine Learning

Submitted by Team 24

Seacom Engineering College

Trainer: **Mr. Gurvansh Singh** (Knowledge Solutions India)

Submitted by **SNEHA DEY**

Table of content

ABSTRACT

In today's era deaths due to heart disease has become a major issue approximately one person dies per minute due to heart disease. This is considering both male and female category and this ratio may vary according to the region also this ratio is considered for the people of various age group. This does not indicate that the people with other age group will not be affected by heart diseases. This problem may start in early age group also and predict the cause and disease is a major challenge nowadays. Here in this project, have discussed various algorithms and tools used for prediction of heart diseases. Huge amount of patient related data is maintained on monthly basis. The store data can be useful for source of predicting the occurrence of feature disease. Some of machine learning techniques are used to predict the heart disease, Such as Support Vector Machines, K – nearest neighbour classifier (K-NN), SVM with PCA, K-NN with PCA. With the same approach in mind, I, the student of Team 24, have taken up as my final project.

OBJECTIVES

- Improve cardiovascular health and quality of life through prevention, detection, and treatment of risk factors for heart attack and stroke.
- Early identification and treatment of heart attacks and strokes.
- Prevention of repeat cardiovascular events and reduction in deaths from cardiovascular disease.

In all the above cases, a notification will be sent to the patient and she/he will be informed of the situation and necessary action could be taken as soon as possible.

LEARNING OBJECTIVES

After completing of this project, we have knowledge on the following topics:

- ✓ Python
- ✓ Machine Learning

SOFTWARE USED

Anaconda: Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment.

Jupyter Notebook: The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. To support data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Classification Task

From the perspective of automatic learning, heart disease detection can be seen as a classification or clustering problem. On the other hand, we formed a model on the vast set of presence and absence file data; we can reduce this problem to classification. For known families, this problem can be reduced to one classification only - having a limited set of classes, including the heart disease sample, it is easier to identify the right class, and the result would be more accurate than with clustering algorithms. In this section, the theoretical context is given on all the methods used in this research. For the purpose of comparative analysis, five Machine Learning algorithms are discussed. The different Machine Learning (ML) algorithms are K-Nearest Neighbor (KNN), Support Vector Machine (SVM) are the reason to choose these algorithms is based on their popularity.

PROBLEM STATEMENT

Previous research studies have examined the application of machine learning techniques for the prediction and classification of Heart disease. However, these studies focus on the particular impacts of specific machine learning techniques and not on the optimization of these techniques using optimized methods. In addition, few researchers attempt to use hybrid optimization methods for an optimized classification of machine learning. The most proposed studies in the literature exploit optimized techniques such as Particle Swarm Optimization and Ant Colony Optimization with a specific ML technique such as SVM, KNN or Random Forest. In this work the Fast Correlation-Based Feature Selection (FCBF) method applied as a first step (pre-treatment). When all continuous attributes are discretized, the attribute selection attributes relevant to mining, from among all the original attributes, are selected. Feature selection, as a pre-processing step to machine learning, is effective in reducing dimensionality, eliminating irrelevant data, increasing learning accuracy and improving understanding of results. In the second step, PSO and ACO are applied to select the relevant characteristics of the data set. The best subset of characteristics selected by the characteristic selection methods improves the accuracy of the classification. Therefore, the third step applies classification methods to diagnose heart disease and measures the classification accuracy to evaluate the performance of characteristic selection methods. The main objective of this article is the prediction heart disease using different classification algorithms such as K-Nearest Neighbor, Support Vector Machine, Naïve Bayes, Random Forest and a Multilayer Perception | Artificial Neural Network optimized by Particle Swarm Optimization (PSO) combined with Ant Colony Optimization (ACO) approaches. The weak data-mining tool is used to analyze data from a heart disease. The main contributions of this paper are:

- Extraction of classified accuracy useful for heart disease prediction
- Remove redundant and irrelevant features with Fast Correlation-Based Feature selection (FCBF) method.
- Optimizations with Particle Swarm Optimization PSO then we consider the result of PSO the initial values of Ant Colony Optimization ACO approaches.
- Comparison of different data mining algorithms on the heart disease dataset.
- Identification of the best performance-based algorithm for heart disease prediction.

IMPORT LIBRARIES

We imported all the libraries for the project:

1. Numpy: To work with arrays.
2. Pandas: To work with csv files and dataframes.
3. Matplotlib: To create charts using pyplot, define parameters using rcParams and color them with cm.rainbow.
4. train_test_split: To split the dataset into training and testing data.
5. StandardScalar: To scale all the features, so that the machine learning model better adapts to the dataset.
6. K-nearest neighbors: It is a non-parametric method used for classification and regression.
7. Confusion matrix: It is also known as an error matrix and is a specific table layout that allows visualization of the performance of the algorithm.
8. warnings: To ignore all warnings which might be showing up in the notebook due to past/future depreciation of a feature.

Next, I imported all the necessary Machine Learning algorithms.

IMPORT DATASET

After downloading the dataset, I saved it to my working directory with the name dataset.csv.

Next, I used read_csv () to read the dataset and save it to the dataset variable.

Before any analysis, just wanted to take a look at the data. So, have used the info() method. Next, I used describe() method.

Understanding the data:

Confusion Matrix: To begin with, let's see the correlation matrix of features and try to analyze it. The figure size is defined to 12 x 8 by using rcParams. Then, I used pyplot to show the correlation matrix. Using xticks and yticks, I've added names to the correlation matrix. colorbar() shows the colorbar for the matrix.

HISTOGRAM

The best part about this type of plot is that it just takes a single command to draw the plots and it provides so much information in return .
Just use dataset.hist()

Let's take a look at the plots, It shows how each feature and label is distributed along different ranges, which further confirms the need for the scaling. Next, wherever you see discrete bars, it basically means that each of these is actually a categorical variable. We will need to handle these categorical variables before applying Machine Learning. Our target labels have two classes, 0 for no disease and 1 for disease.

TARGET CLASS PREDICTION

Bar Plot for Target Class:
It's really essential that the dataset we are working on should be approximately balanced. An extremely imbalanced dataset can render the whole model training useless and thus, will be of no use.
For x-axis I used the unique() values from the target column and then set their name using xticks. For y-axis, I used value_count() to get the values for each class. I colored the bars as green and red.
From the plot, we can see that the classes are almost balanced and we are good to proceed with data processing.

DATA PROCESSING

To work with categorical variables, we should break each categorical column into dummy columns with 1s and 0s.

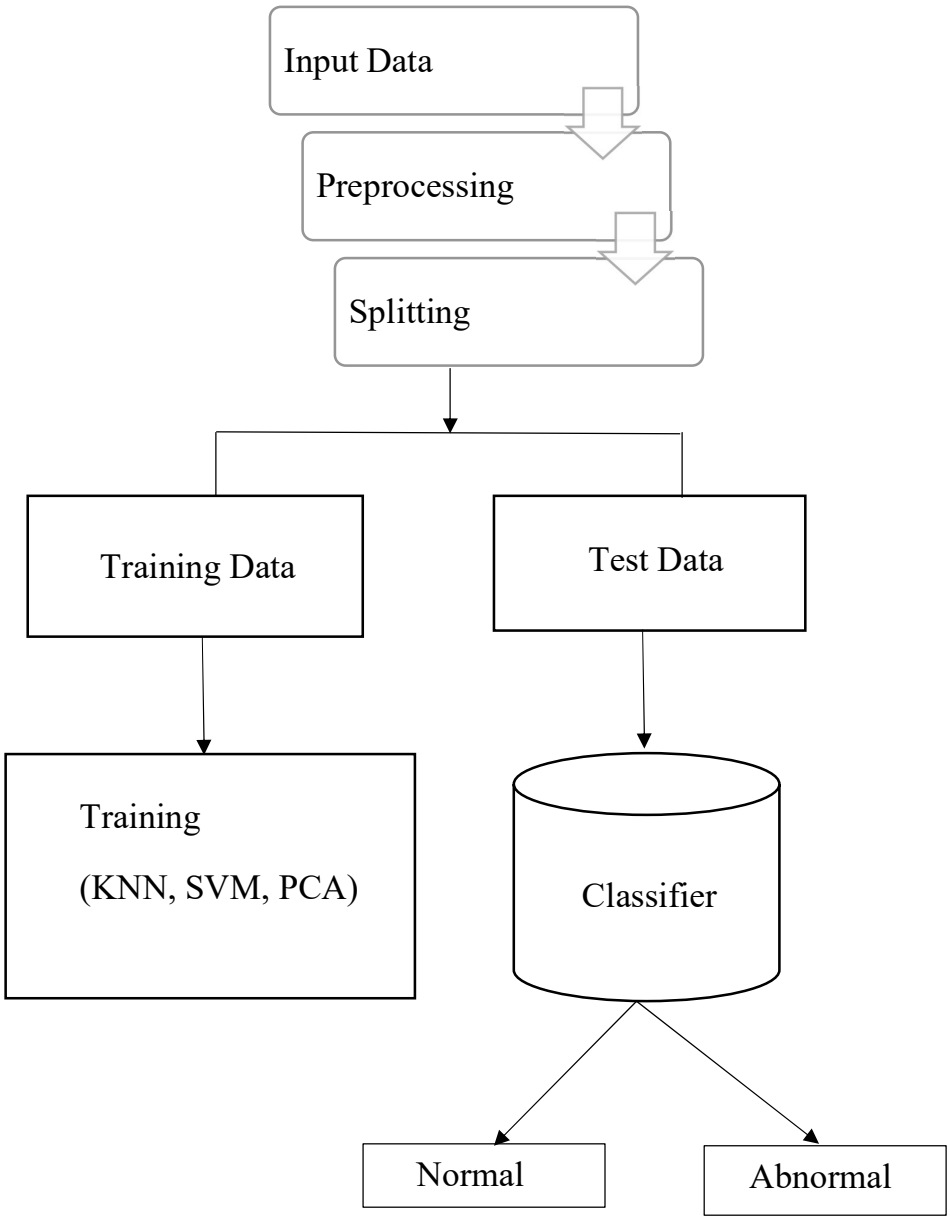
Let's say we have a column Gender, with values 1 for Male and 0 for Female. It needs to be converted into two columns with the value 1 where the column would be true and 0 where it will be false. Take a look at the Gist below.

To get this done, we use the get_dummies() method from pandas. Next, we need to scale the dataset for which we will use the StandardScaler. The fit_transform() method of the scaler scales the data and we update the columns.

The dataset is now ready. We can begin with training our models.

METHODOLOGY

The block drawing for organization of heart disease databank is shown in figure:



MACHINE LEARNING METHOD

In this project, I took 4 algorithms and varied their various parameters and compared the final models. I split the dataset into 67% training data and 33% testing data.

K Neighbours Classifier

This classifier looks for the classes of K nearest neighbours of a given data point and based on the majority class, it assigns a class to this data point. However, the number of neighbours can be varied. I varied them from 1 to 20 neighbours and calculated the test score in each case.
Then, I plot a line graph of the number of neighbors and the test score achieved in each case.

Support Vector Classifier

This classifier aims at forming a hyperplane that can separate the classes as much as possible by adjusting the distance between the data points and the hyperplane. There are several kernels based on which the hyperplane is decided. I tried four kernels namely, *linear*, *poly*, *rbf*, and *sigmoid*.
Once I had the scores for each, I used the rainbow method to select different colors for each bar and plot a bar graph of the scores achieved by each.

CODE

 # Importing libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Machine Learning
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

 # Importing dataset

```
dataset=pd.read_csv('heart.csv')
print(dataset)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      63    1   3       145   233    1        0      150      0      2.3
1      37    1   2       130   250    0        1      187      0      3.5
2      41    0   1       130   204    0        0      172      0      1.4
3      56    1   1       120   236    0        1      178      0      0.8
4      57    0   0       120   354    0        1      163      1      0.6
..    ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298    57    0   0       140   241    0        1      123      1      0.2
299    45    1   3       110   264    0        1      132      0      1.2
300    68    1   0       144   193    1        1      141      0      3.4
301    57    1   0       130   131    0        1      115      1      1.2
302    57    0   1       130   236    0        0      174      0      0.0

      slope  ca  thal  target
0         0   0     1       1
1         0   0     2       1
2         2   0     2       1
3         2   0     2       1
4         2   0     2       1
..      ...  ..   ...     ...
298       1   0     3       0
299       1   0     3       0
300       1   2     3       0
301       1   1     3       0
302       1   1     2       0

[303 rows x 14 columns]
```

```
dataset.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	that	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

dataset.info()

```
<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
```

dataset.describe()

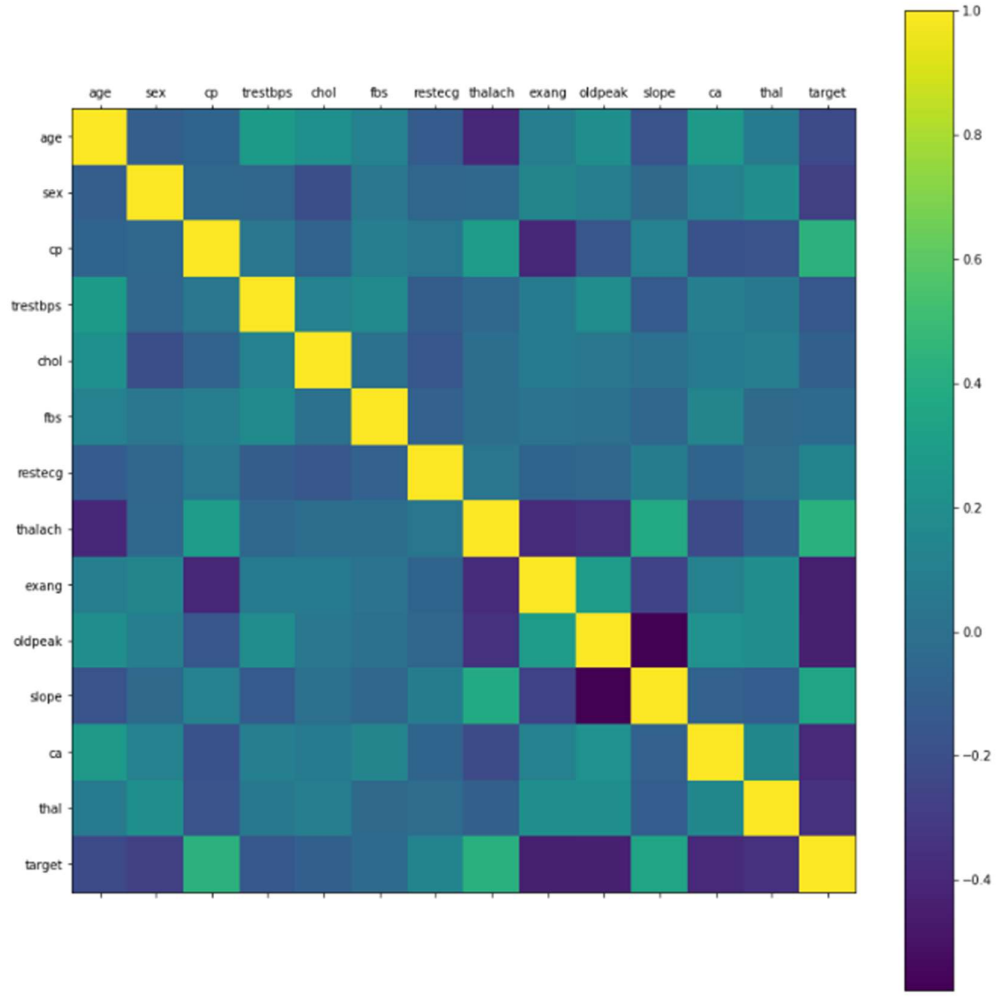
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	that	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	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

The method revealed that the range of each variable is different. The maximum value of age is 77 but for chol it is 564. Thus, feature scaling must be performed on the dataset.

Confusion Matrix

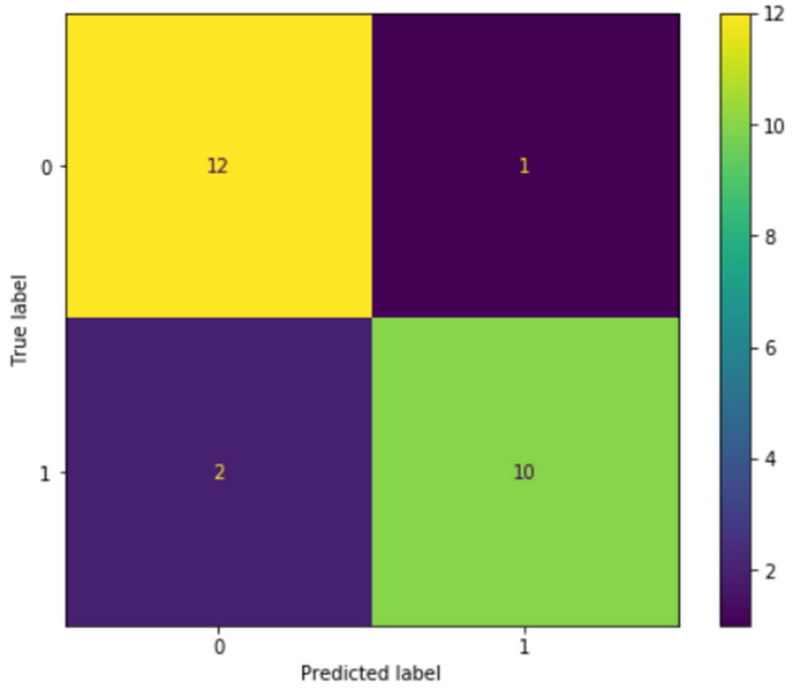
```
rcParams['figure.figsize'] = 20, 14

plt.matshow(dataset.corr())

plt.yticks(np.arange(dataset.shape[1]), dataset.columns)

plt.xticks(np.arange(dataset.shape[1]), dataset.columns)

plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x22861232448>
```



It's easy to see that there is no single feature that has a very high correlation with our target value. Also, some of the features have a negative correlation with the target value and some have positive.

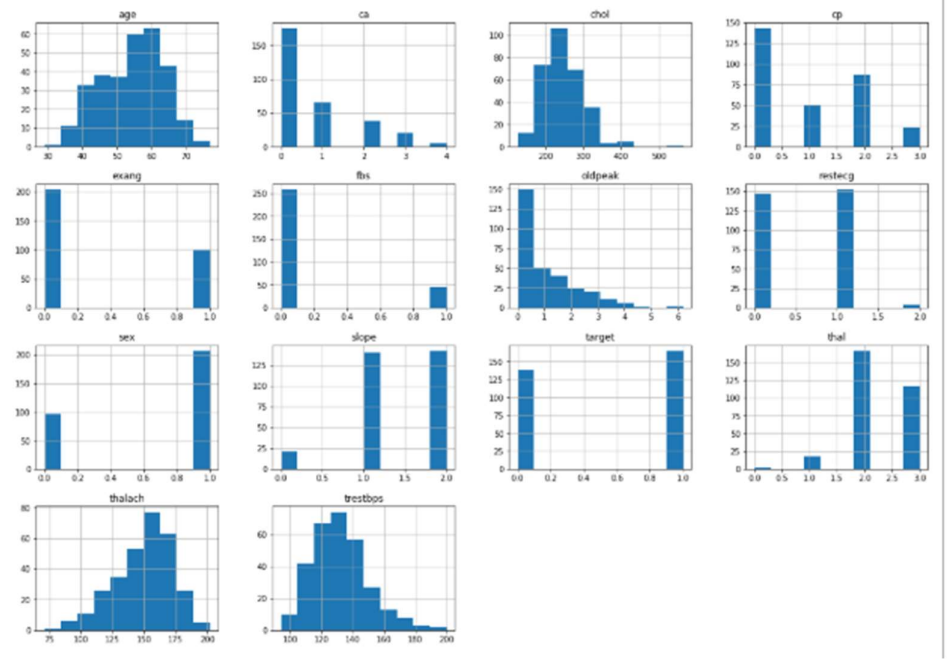
```
from sklearn.datasets import make_classification
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
X, y = make_classification(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = SVC(random_state=0)
clf.fit(X_train, y_train)
SVC(random_state=0)
plot_confusion_matrix(clf, X_test, y_test)
fig= plt.figure(figsize=(6,4))
plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```

# **Histogram**
dataset.hist()

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002285F395308>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000228615400C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022861577908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002286150B888>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000228615E89C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022861618A88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002286144A548>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002286148B708>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000022861458808>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002286163C9C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000228616A2A48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000228619BAAC8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000228619F3BC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022861A2DD08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022861A64E08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000022861A9DFC8>]],
      dtype=object)
```



Let's take a look at the plots. It shows how each feature and label is distributed along different ranges, which further confirms the need for scaling. Next, wherever you see discrete bars, it basically means that each of these is actually a categorical variable. We will need to handle these categorical variables before applying Machine Learning. Our target labels have two classes, 0 for no disease and 1 for disease.

Predict the Target class

```
features=['target']

x=dataset[features]

print(x.head())
```

```
   target
0       1
1       1
2       1
3       1
4       1
```

```
y=dataset.target

print(y.head())
```

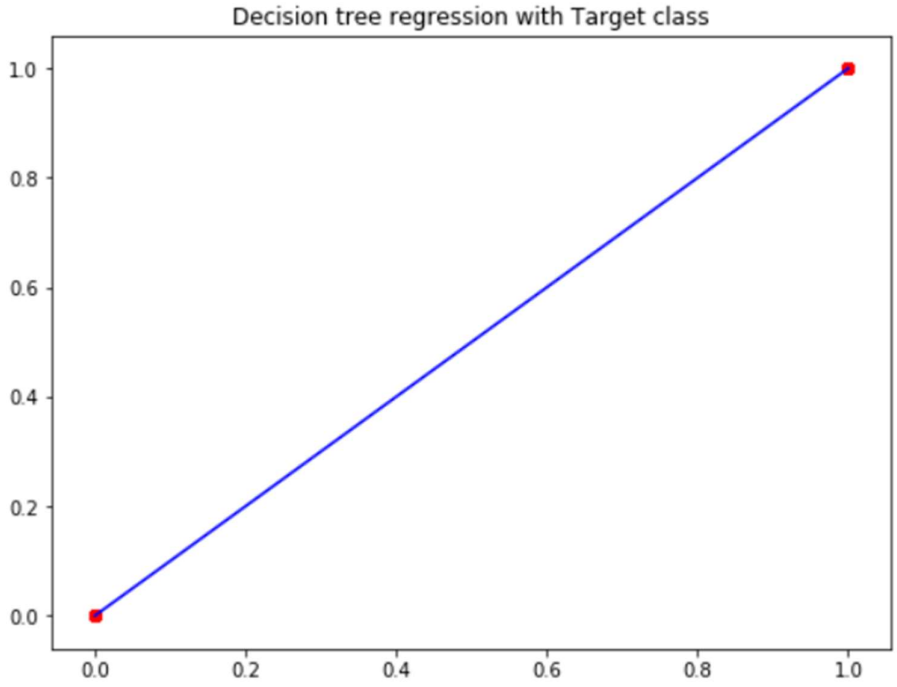
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
from sklearn.tree import DecisionTreeRegressor

model=DecisionTreeRegressor(random_state=1)

model.fit(x,y)

predict=model.predict(x)

print(predict)
```

```
[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. 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. 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. 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.]
```

```
y_pred = model.predict(np.array([[5]]))
print(y_pred)
[1.]
```

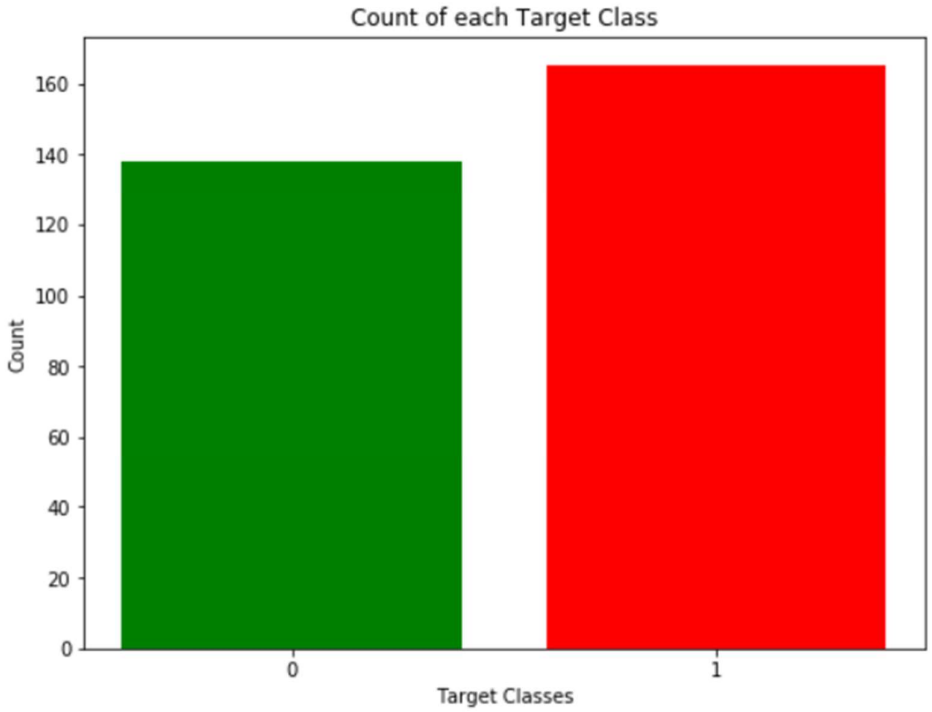
```
plt.scatter(x,y,color = 'red')
plt.plot(x,model.predict(x),color = 'blue')
plt.title('Decision tree regression with Target class')
plt.show()
```



Bar Plot for Target Class

```
rcParams['figure.figsize'] = 8,6

plt.bar(dataset['target'].unique(), dataset['target'].value_counts(), color = ['red', 'green'])

plt.xticks([0, 1])

plt.xlabel('Target Classes')

plt.ylabel('Count')

plt.title('Count of each Target Class')
```

```
Text(0.5, 1.0, 'Count of each Target Class')
```



For x-axis I used the unique() values from the target column and then set their name using xticks. For y-axis, I used value_count() to get the values for each class. I colored the bars as green and red.

From the plot, we can see that the classes are almost balanced and we are good to proceed with data processing.

Bar plot for Count of male and female

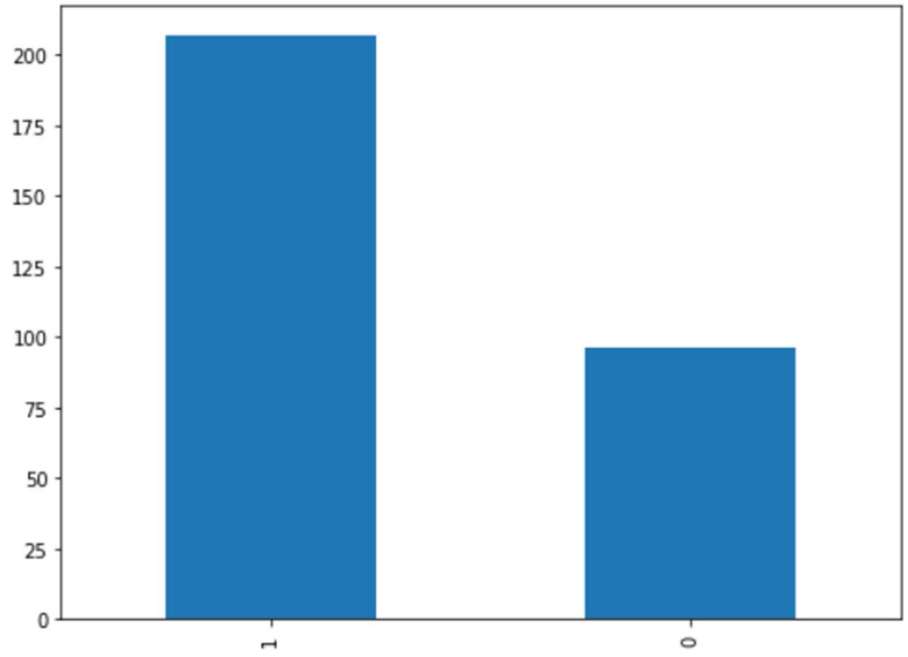
 # As per dataset, 0 – female and 1 – male

dataset['sex'].value_counts()

```
1    207
0     96
Name: sex, dtype: int64
```

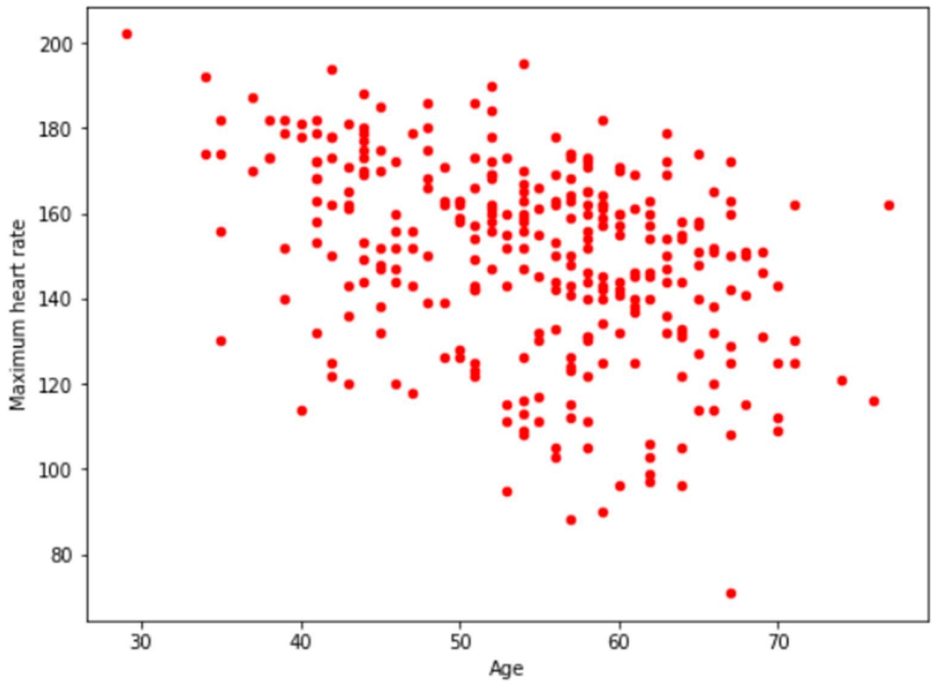
dataset['sex'].value_counts().plot(kind='bar')

```
<matplotlib.axes._subplots.AxesSubplot at 0x22861e1a1c8>
```



Scatter plot between Age and Maximum heart rate

```
dataset.plot(kind='scatter',x='age',y='thalach',color='red')

plt.xlabel('Age')

plt.ylabel('Maximum heart rate')

plt.show()
```



 # Data Processing

```
dataset = pd.get_dummies(dataset, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])

standardScaler = StandardScaler()

columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

dataset[columns_to_scale] = standardScaler.fit_transform(dataset[columns_to_scale])
```

The dataset is now ready. We can begin with training our models.

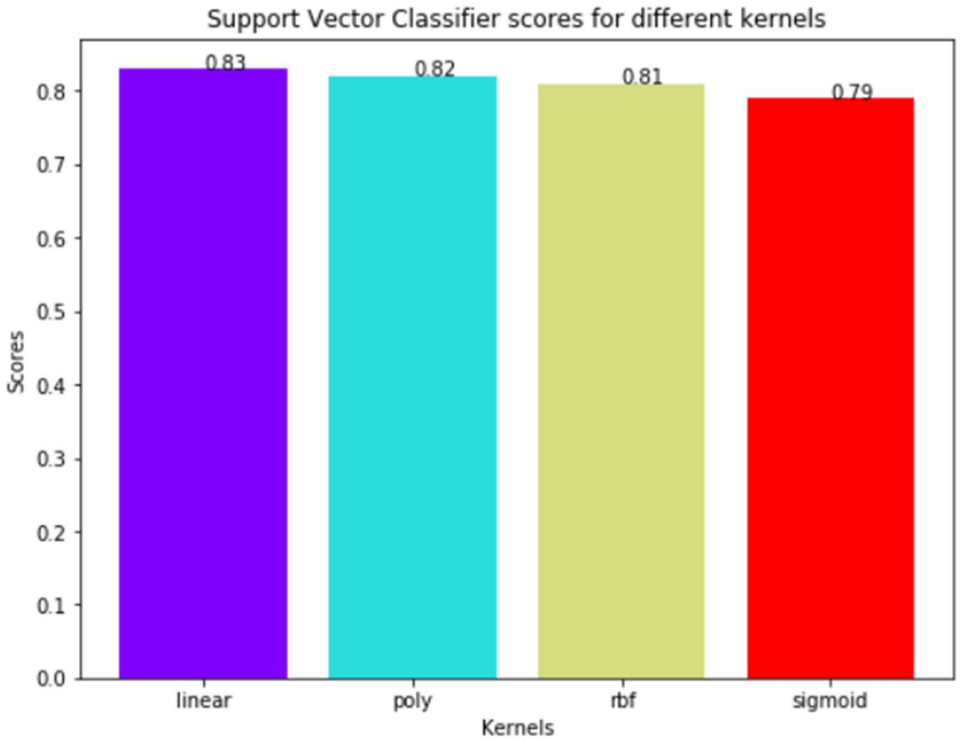
 # Splitting dataset

```
y = dataset['target']

X = dataset.drop(['target'], axis = 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 0)
```

In this project, I took 4 algorithms and varied their various parameters and compared the final models. I split the dataset into 67% training data and 33% testing data.

Support Vector Machine

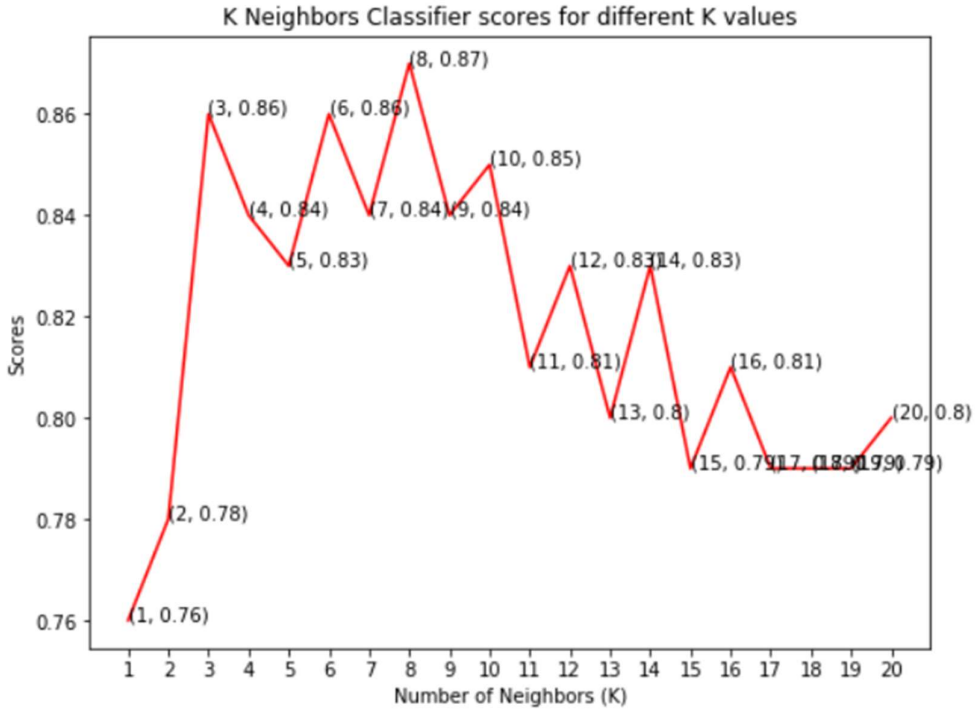
```
svc_scores = []

kernels = ['linear', 'poly', 'rbf', 'sigmoid']

for  i  in  range ( len ( kernels )):

    svc_classifier = SVC(kernel = kernels[i])

    svc_classifier.fit(X_train, y_train)

    svc_scores.append(svc_classifier.score(X_test, y_test))

colors = rainbow(np.linspace(0, 1, len(kernels)))

plt.bar(kernels, svc_scores, color = colors)

for i in range(len(kernels)):

    plt.text(i, svc_scores[i], svc_scores[i])

plt.xlabel('Kernels')

plt.ylabel('Scores')

plt.title('Support Vector Classifier scores for different kernels')
```



 # K – nearest neighbor classifier

```
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    knn_classifier.fit(X_train, y_train)
    knn_scores.append(knn_classifier.score(X_test, y_test))


plt.plot([k for k in range(1, 21)], knn_scores, color = 'red')
for i in range(1,21):
    plt.text(i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks([i for i in range(1, 21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('K Neighbors Classifier scores for different K values')
```

```
Text(0.5, 1.0, 'K Neighbors Classifier scores for different K values')
```



 # SVM with PCA

#PCA

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)

X_train = pca.fit_transform(X_train)

X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

print(explained_variance)
```

```
[0.24423267 0.15198552]
```

```
### Training the svm model on training set

from sklearn.svm import SVC

classifier = SVC(kernel='linear',random_state=0)

classifier.fit(X_train,y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
    verbose=False)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
   decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
   max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
   verbose=False)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001,
    verbose=False)
```

```
### Predict the Test results

y_pred = classifier.predict(X_test)

print(y_pred)
```

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

MAking the Confusion Matrix

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

```
[[35 13]
 [ 9 43]]
```

#Comparing the predictions with the actual results

```
comparison = pd.DataFrame(y_test,columns=['y_test'])
comparison['y_predicted'] = y_pred
comparison.head()
```

	y_test	y_predicted
0	NaN	0
1	NaN	0
2	NaN	1
3	NaN	0
4	NaN	0

#Apply k-fold validation here

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=classifier,X=X_train,y=y_train,cv=10)
accuracies
```

```
array([0.66666667, 0.80952381, 0.76190476, 0.9       , 0.75      ,
       0.9       , 0.95      , 0.7       , 0.7       , 0.85      ])
```

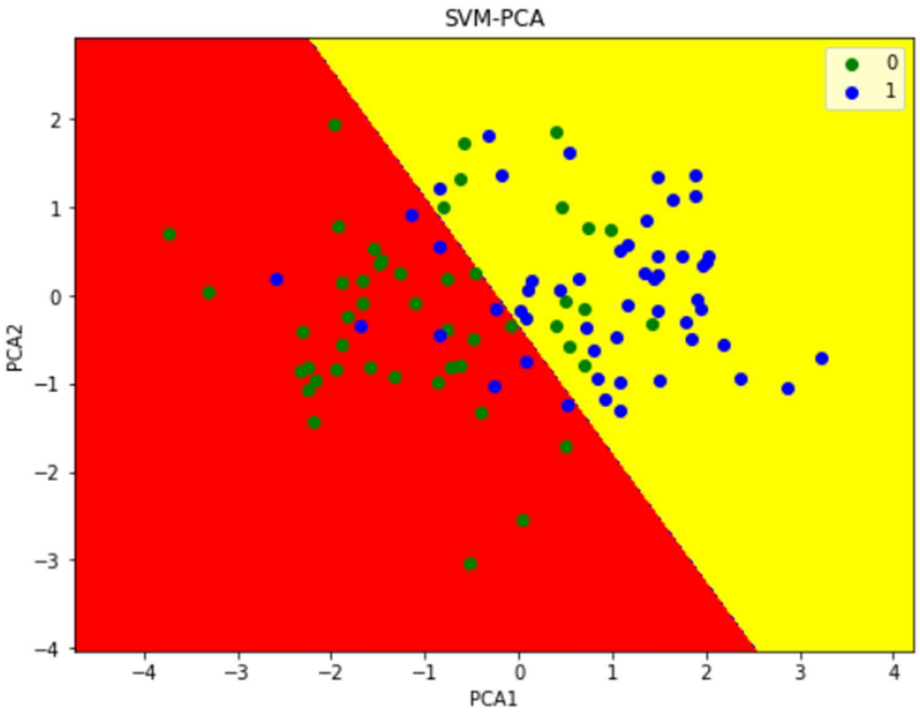
#Applying grid search for optimal parameters and model after k-fold validation

```
from sklearn.model_selection import GridSearchCV
```

```
parameters = [{'C':[0.01,0.1,1,10,50,100,500,1000], 'kernel':['rbf'], 'gamma':
[0.1,0.125,0.15,0.17,0.2]}]

grid_search = GridSearchCV(estimator=classifier, param_grid=parameters,
scoring ='accuracy',cv=10,n_jobs=-1)

grid_search = grid_search.fit(X_train,y_train)

best_accuracy = grid_search.best_score_

best_accuracy
```

```
0.8088095238095239
```

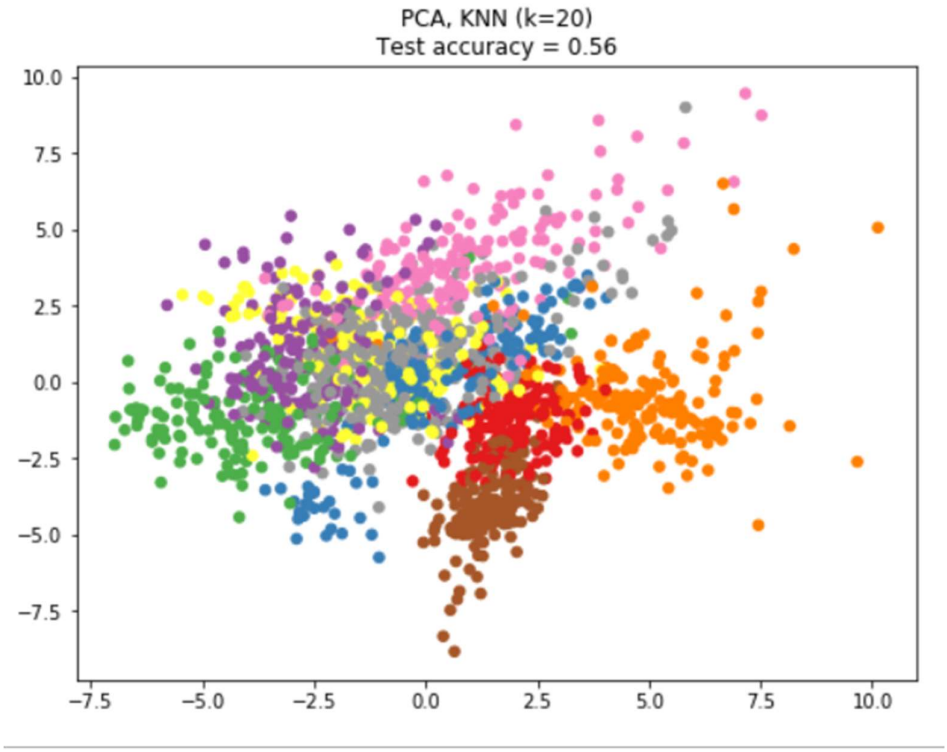
```
### visualization of the test data

from matplotlib.colors import ListedColormap

x_set, y_set = X_test, y_test

x1,x2 = np.meshgrid(np.arange(start = x_set[:,0].min()-1, stop =
x_set[:,0].max()+1, step = 0.01),

                np.arange(start = x_set[:,1].min()-1, stop = x_set[:,1].max()+1, step =
0.01))

plt.contourf(x1,x2,classifier.predict(np.array([x1.ravel(),x2.ravel()]).T).reshape(
x1.shape),

          aplha = 0.75, cmap = ListedColormap(('red','blue','yellow')))

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(('green','blue','red'))(i), label = j)

plt.title('SVM-PCA')

plt.xlabel('PCA1')

plt.ylabel('PCA2')

plt.legend()

plt.show()
```


SVM-PCA
0
1
PCA2
PCA1
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4

KNN with PCA

```
from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.decomposition import PCA

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.neighbors import
(KNeighborsClassifier,NeighborhoodComponentsAnalysis)

from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import StandardScaler


n_neighbors = 20

random_state = 0

X, y = datasets.load_digits(return_X_y=True)


# Split into train/test

X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.5,
stratify=y,random_state=random_state)


dim = len(X[0])

n_classes = len(np.unique(y))

pca = make_pipeline(StandardScaler(),PCA(n_components=2,
random_state=random_state))

knn = KNeighborsClassifier(n_neighbors=n_neighbors)

dim_reduction_methods = [('PCA', pca)]


for i, (name, model) in enumerate(dim_reduction_methods):

    plt.figure()
```

```
    model.fit(X_train, y_train)


    knn.fit(model.transform(X_train), y_train)
    acc_knn = knn.score(model.transform(X_test), y_test)
    X_embedded = model.transform(X)
    plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=y, s=30, cmap='Set1')
    plt.title("{}, KNN (k={})\nTest accuracy =
{:.2f}".format(name,n_neighbors,acc_knn))


plt.show()
```



ACCURACY

1. Support Vector Classifier: **83%**
2. K Neighbours Classifier: **87%**
3. SVM with PCA: **80%**
4. KNN with PCA: **56%**

K Neighbours Classifier scored the best score of **87%** with 8 neighbours.

INFERENCE

In the above paper we have studied various classification algorithms that can be used for classification of heart disease databases also we have seen different techniques that can be used for classification and the accuracy obtained by them. This investigation tells us about dissimilar technologies that are used in dissimilar papers with dissimilar count of attributes with different accuracies depending on the tools designed for execution. The accurateness of the structure can be further upgraded by creating various combinations of data mining techniques and by parameter tuning also.

ACKNOWLEDGEMENT

I would like to thank **Mr. Gurvansh Singh** for his valuable suggestions, his relevant remarks and his perpetual advices and **Knowledge Solution India (KSI)** for giving such kind of platform.