Heart disease classification

SSRK KASYAP

Contents

Introduction

- It is difficult to identify heart disease because of several contributory risk factors such as diabetes, high blood pressure, high cholesterol, abnormal pulse rate and many other factors.
- Among various life threatening diseases, heart disease has garnered a great deal of attention in medical research.
- The diagnosis of heart disease is a challenging task, which can offer automated prediction about the heart condition of patient so that further treatment can be made effective.
- the diagnosis of heart disease is usually based on signs, symptoms of the patient. >The severity of the disease is classified based on various methods like K-Nearest Neighbor Algorithm (KNN), Decision Trees (DT), Genetic algorithm (GA), and Naive Bayes(NB).
- The nature of heart disease is complex and hence, the disease must be handled carefully. Not doing so may affect the heart or cause premature death.

Problem Statement

Heart Disease prediction using Machine Learning

Motivation

- A major challenge facing healthcare organizations is the provision of quality services at affordable costs.
- Quality service implies diagnosing patients correctly and administering treatments that are effective.
- Poor clinical decisions can lead to disastrous consequences which are therefore unacceptable
- Hospitals must also minimize the cost of clinical tests.
- They can achieve this results by employing appropriate computer based information and decision support system

Objectives

- The main objective of this research is to develop a heart prediction system, the system can discover and extract hidden knowledge associated with diseases from heart data set.
- This system aims to exploit machine learning techniques on medical data set to assist in the prediction of the heart disease.
- Reduce the cost of medical tests.
- To help avoid human biases.

Algorithms Used

- Logistic Regression
- Naive Bayes
- K-nearest Neighbor
- Decision Tree
- Support Vector Machine
- Random Forest

Applications

Medical Institutions:-

To teach medical students how the heart attack been measured, or how to identify that the person is suffering from heart disease.

Hospitals:

To detect that is the person having heart disease or not.

Summary

Heart disease prediction is challenging and very important in medical field. However, the mortality rate can be drastically controlled if the disease is detected at early stage and preventive measures are adopted as soon as possible.The proposed hybrid HRFLM approach is combined the characteristics of random forest(RF) and linear method(IM).HRFLM proved to be quite accurate in the prediction of heart disease.

THANK YOU

```
!pip install mlxtend
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: mlxtend in /usr/local/lib/python3.10/dist-packages (0.14.0)
Requirement already satisfied: scipy>=0.17 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.10.1)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.22.4)
Requirement already satisfied: pandas>=0.17.1 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.5.3)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (1.2.2)
Requirement already satisfied: matplotlib>=1.5.1 in /usr/local/lib/python3.10/dist-packages (from mlxtend) (3.7.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from mlxtend) (67.7.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (8.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.5.1->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.17.1->mlxtend) (2022.7.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->mlxtend) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->mlxtend) (3.1.0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=1.5.1->mlxter
```

Packages Required

```
!pip install pandas_profiling
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas_profiling in /usr/local/lib/python3.10/dist-packages (3.6.6)
Requirement already satisfied: ydata-profiling in /usr/local/lib/python3.10/dist-packages (from pandas_profiling) (4.3.1)
Requirement already satisfied: scipy<1.11,>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (1
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profilir
Requirement already satisfied: matplotlib<4,>=3.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (3
Requirement already satisfied: pydantic<2,>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (1
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (6
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (
Requirement already satisfied: visions[type_image_path]==0.7.5 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_
Requirement already satisfied: numpy<1.24,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (0.1.
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (6
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (4.65
Requirement already satisfied: seaborn<0.13,>=0.10.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
Requirement already satisfied: multimethod<2,>=1.4 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling
Requirement already satisfied: typeguard<3,>=2.13.2 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
Requirement already satisfied: imagehash==4.3.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (4.3
Requirement already satisfied: wordcloud>=1.9.1 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (1.9
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling) (1.8.1)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling->pandas_prc
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling->pandas_profili
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0.7.5->ydata-prc
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0.7.5->ydata-prc
Requirement already satisfied: tangled-up-in-unicode>=0.0.4 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]==0
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profiling->pa
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->pa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->pandas
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->p
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->p
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->par
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling->pa
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<4,>=3.2->ydata-profiling
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<2.1,>1.1->ydata-profiling->p
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-packages (from phik<0.13,>=0.11.1->ydata-profiling->panc
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<2,>=1.8.1->ydata-profi
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profili
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling-
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-prc
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling->panda
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels<1,>=0.13.2->ydata-profiling->pa
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels<1,>=0.13.2->ydata-profilir
```

```
import six
import sys
sys.modules['sklearn.externals.six'] = six
```

```
#loading dataset
import pandas as pd
import numpy as np
#visualisation
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
#EDA
from collections import Counter
import pandas_profiling as pp
# data preprocessing
from sklearn.preprocessing import StandardScaler
# data splitting
from sklearn.model_selection import train_test_split
# data modeling
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

```
data = pd.read_csv("/content/heart.csv")
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

Missing Value Detection

```
data.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
```

```
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Descriptive statistics

```
data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	c
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.

EDA

```
pp.ProfileReport(data)
```

Summarize dataset: 100% 48/48 [00:08<00:00, 2.62it/s, Completed]

Generate report structure: 100% 1/1 [00:07<00:00, 7.34s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.08s/it]

Model prepration

```
y = data["target"]
X = data.drop('target',axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state = 0)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Number of observations 1025 **Categorical** 9

```
print(y_test.unique())
Counter(y_train)
```

```
[1 0]
Counter({1: 419, 0: 401})
```

Duplicate rows (%) 29.5%

```
m1 = 'Logistic Regression'
lr = LogisticRegression()
model = lr.fit(X_train, y_train)
lr_predict = lr.predict(X_test)
lr_conf_matrix = confusion_matrix(y_test, lr_predict)
lr_acc_score = accuracy_score(y_test, lr_predict)
print("confussion matrix")
print(lr_conf_matrix)
print("\n")
print("Accuracy of Logistic Regression:",lr_acc_score*100,'\n')
print(classification_report(y_test,lr_predict))
```

```
confussion matrix
[[ 77  21]
 [  7 100]]


Accuracy of Logistic Regression: 86.34146341463415

              precision    recall  f1-score   support

           0       0.92      0.79      0.85        98
           1       0.83      0.93      0.88       107

    accuracy                           0.86       205
   macro avg       0.87      0.86      0.86       205
weighted avg       0.87      0.86      0.86       205
```

```
m2 = 'Naive Bayes'
nb = GaussianNB()
nb.fit(X_train,y_train)
nbpred = nb.predict(X_test)
nb_conf_matrix = confusion_matrix(y_test, nbpred)
nb_acc_score = accuracy_score(y_test, nbpred)
print("confussion matrix")
print(nb_conf_matrix)
print("\n")
print("Accuracy of Naive Bayes model:",nb_acc_score*100,'\n')
print(classification_report(y_test,nbpred))
```

```
confussion matrix
[[79 19]
 [11 96]]


Accuracy of Naive Bayes model: 85.36585365853658
```

```
              precision    recall  f1-score   support

           0       0.88      0.81      0.84        98
           1       0.83      0.90      0.86       107

    accuracy                           0.85       205
   macro avg       0.86      0.85      0.85       205
weighted avg       0.86      0.85      0.85       205
```

```
m3 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=20, random_state=2,max_depth=5)
rf.fit(X_train,y_train)
rf_predicted = rf.predict(X_test)
rf_conf_matrix = confusion_matrix(y_test, rf_predicted)
rf_acc_score = accuracy_score(y_test, rf_predicted)
print("confussion matrix")
print(rf_conf_matrix)
print("\n")
print("Accuracy of Random Forest:",rf_acc_score*100,'\n')
print(classification_report(y_test,rf_predicted))
```

```
confussion matrix
[[ 89   9]
 [  2 105]]


Accuracy of Random Forest: 94.6341463414634

              precision    recall  f1-score   support

           0       0.98      0.91      0.94        98
           1       0.92      0.98      0.95       107

    accuracy                           0.95       205
   macro avg       0.95      0.94      0.95       205
weighted avg       0.95      0.95      0.95       205
```

```
m4 = 'Extreme Gradient Boost'
xgb = XGBClassifier(learning_rate=0.01, n_estimators=25, max_depth=15,gamma=0.6, subsample=0.52,colsample_bytree=0.6,seed=27,
                    reg_lambda=2, booster='dart', colsample_bylevel=0.6, colsample_bynode=0.5)
xgb.fit(X_train, y_train)
xgb_predicted = xgb.predict(X_test)
xgb_conf_matrix = confusion_matrix(y_test, xgb_predicted)
xgb_acc_score = accuracy_score(y_test, xgb_predicted)
print("confussion matrix")
print(xgb_conf_matrix)
print("\n")
print("Accuracy of Extreme Gradient Boost:",xgb_acc_score*100,'\n')
print(classification_report(y_test,xgb_predicted))
```

```
confussion matrix
[[ 87  11]
 [  5 102]]


Accuracy of Extreme Gradient Boost: 92.19512195121952

              precision    recall  f1-score   support

           0       0.95      0.89      0.92        98
           1       0.90      0.95      0.93       107

    accuracy                           0.92       205
   macro avg       0.92      0.92      0.92       205
weighted avg       0.92      0.92      0.92       205
```

```
m5 = 'K-NeighborsClassifier'
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
knn_predicted = knn.predict(X_test)
knn_conf_matrix = confusion_matrix(y_test, knn_predicted)
knn_acc_score = accuracy_score(y_test, knn_predicted)
print("confussion matrix")
print(knn_conf_matrix)
print("\n")
```

```
print("Accuracy of K-NeighborsClassifier:",knn_acc_score*100,'\n')
print(classification_report(y_test,knn_predicted))
```

```
confussion matrix
[[84 14]
 [11 96]]


Accuracy of K-NeighborsClassifier: 87.8048780487805

              precision    recall  f1-score   support

           0       0.88      0.86      0.87        98
           1       0.87      0.90      0.88       107

    accuracy                           0.88       205
   macro avg       0.88      0.88      0.88       205
weighted avg       0.88      0.88      0.88       205
```

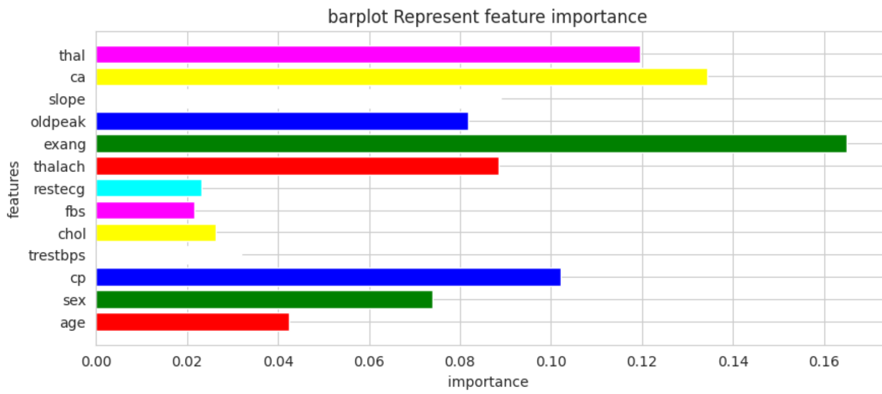
```
m6 = 'DecisionTreeClassifier'
dt = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 6)
dt.fit(X_train, y_train)
dt_predicted = dt.predict(X_test)
dt_conf_matrix = confusion_matrix(y_test, dt_predicted)
dt_acc_score = accuracy_score(y_test, dt_predicted)
print("confussion matrix")
print(dt_conf_matrix)
print("\n")
print("Accuracy of DecisionTreeClassifier:",dt_acc_score*100,'\n')
print(classification_report(y_test,dt_predicted))
```

```
confussion matrix
[[95  3]
 [ 8 99]]


Accuracy of DecisionTreeClassifier: 94.6341463414634

              precision    recall  f1-score   support

           0       0.92      0.97      0.95        98
           1       0.97      0.93      0.95       107

    accuracy                           0.95       205
   macro avg       0.95      0.95      0.95       205
weighted avg       0.95      0.95      0.95       205
```

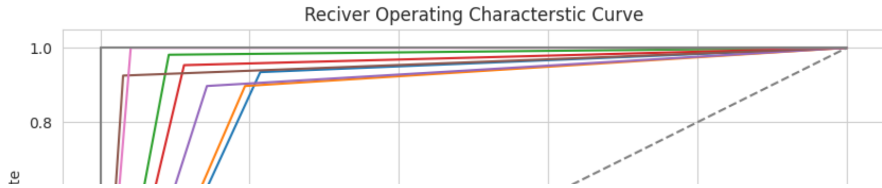
```
m7 = 'Support Vector Classifier'
svc =  SVC(kernel='rbf', C=2)
svc.fit(X_train, y_train)
svc_predicted = svc.predict(X_test)
svc_conf_matrix = confusion_matrix(y_test, svc_predicted)
svc_acc_score = accuracy_score(y_test, svc_predicted)
print("confussion matrix")
print(svc_conf_matrix)
print("\n")
print("Accuracy of Support Vector Classifier:",svc_acc_score*100,'\n')
print(classification_report(y_test,svc_predicted))
```

```
confussion matrix
[[ 94   4]
 [  0 107]]


Accuracy of Support Vector Classifier: 98.04878048780488

              precision    recall  f1-score   support

           0       1.00      0.96      0.98        98
           1       0.96      1.00      0.98       107

    accuracy                           0.98       205
   macro avg       0.98      0.98      0.98       205
weighted avg       0.98      0.98      0.98       205
```

```
imp_feature = pd.DataFrame({'Feature': ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal'], 'Importance': xgb.feature_importances_})
plt.figure(figsize=(10,4))
plt.title("barplot Represent feature importance ")
plt.xlabel("importance ")
plt.ylabel("features")
colors = ["red","green","blue","white","yellow","magenta","cyan"]
plt.barh(imp_feature['Feature'],imp_feature['Importance'],color = colors)
plt.show()
```




```
lr_false_positive_rate,lr_true_positive_rate,lr_threshold = roc_curve(y_test,lr_predict)
nb_false_positive_rate,nb_true_positive_rate,nb_threshold = roc_curve(y_test,nbpred)
rf_false_positive_rate,rf_true_positive_rate,rf_threshold = roc_curve(y_test,rf_predicted)
xgb_false_positive_rate,xgb_true_positive_rate,xgb_threshold = roc_curve(y_test,xgb_predicted)
knn_false_positive_rate,knn_true_positive_rate,knn_threshold = roc_curve(y_test,knn_predicted)
dt_false_positive_rate,dt_true_positive_rate,dt_threshold = roc_curve(y_test,dt_predicted)
svc_false_positive_rate,svc_true_positive_rate,svc_threshold = roc_curve(y_test,svc_predicted)


sns.set_style('whitegrid')
plt.figure(figsize=(10,5))
plt.title('Reciver Operating Characterstic Curve')
plt.plot(lr_false_positive_rate,lr_true_positive_rate,label='Logistic Regression')
plt.plot(nb_false_positive_rate,nb_true_positive_rate,label='Naive Bayes')
plt.plot(rf_false_positive_rate,rf_true_positive_rate,label='Random Forest')
plt.plot(xgb_false_positive_rate,xgb_true_positive_rate,label='Extreme Gradient Boost')
plt.plot(knn_false_positive_rate,knn_true_positive_rate,label='K-Nearest Neighbor')
plt.plot(dt_false_positive_rate,dt_true_positive_rate,label='Desion Tree')
plt.plot(svc_false_positive_rate,svc_true_positive_rate,label='Support Vector Classifier')
plt.plot([0,1],ls='--')
plt.plot([0,0],[1,0],c='.5')
plt.plot([1,1],c='.5')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.legend()
plt.show()
```

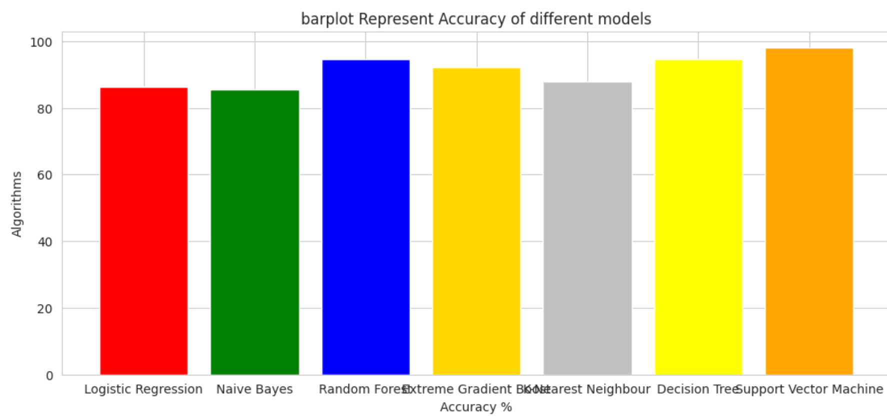


Model Evaluation


```
model_ev = pd.DataFrame({'Model': ['Logistic Regression','Naive Bayes','Random Forest','Extreme Gradient Boost',
                    'K-Nearest Neighbour','Decision Tree','Support Vector Machine'], 'Accuracy': [lr_acc_score*100,
                    nb_acc_score*100,rf_acc_score*100,xgb_acc_score*100,knn_acc_score*100,dt_acc_score*100,svc_acc_score*100]})
model_ev
```

	Model	Accuracy
0	Logistic Regression	86.341463
1	Naive Bayes	85.365854
2	Random Forest	94.634146
3	Extreme Gradient Boost	92.195122
4	K-Nearest Neighbour	87.804878
5	Decision Tree	94.634146
6	Support Vector Machine	98.048780

```
colors = ['red','green','blue','gold','silver','yellow','orange',]
plt.figure(figsize=(12,5))
plt.title("barplot Represent Accuracy of different models")
plt.xlabel("Accuracy %")
plt.ylabel("Algorithms")
plt.bar(model_ev['Model'],model_ev['Accuracy'],color = colors)
plt.show()
```



0s completed at 9:42 PM

Heart Disease Classification Report

Introduction

Heart disease has risen to become one of the leading causes of death all over the world. Ac- cording to the World Health Organization, cardiac illnesses claim the lives of 17.7 million people each year, accounting for 31% of all fatalities worldwide. Heart disease has become the top cause of death in India as well. As a result, it is essential to be able to forecast heart-related disorders in a reliable and precise manner. Data on various health-related concerns is compiled by medical institutions all over the world. These data can be used to gain significant information utilizing a variety of machine learning techniques. However, the amount of data collected is enormous, and it is frequently noisy.

We analyze the various machine learning algorithms and find the best to predict the presence or absence of heart disease. The tar- get we will be exploring is binary classification which is 0 to show the absence of heart disease and 1 to show the presence of heart disease.

We are going to use various machine learning algorithms to predict the target. We will be using a number of different features about a person to predict whether they have heart dis- ease or not. The dependent variable is whether or not a patient has heart disease, while the independent variables are the patient's many medical characteristics. The various machine learning algorithms used for our model will be Logistic Regression, K-Nearest Neighbours, and Random Forest. We will compare the scores of all these models by splitting our data into training and testing in an approximate 80:20 ratio. We will also tune the hyper parameters for all these models to yield the best results. And finally conclude the best prediction model for our heart disease dataset.

Methodology Implementation

We have collected data from various reliable sources from the internet. After analyzing various factors, we have reached a conclusion that 13 independent variables will determine 1 target variable. To do this we will have to split the target variable from the rest. If we can reach 96% accuracy at predicting whether or not a patient has heart disease during the proof of concept, we'll pursue this project.

Training and Testing Dataset

The train and split procedure is used to divide the data into two halves.

1. Train split
2. Test split

The model designed will first train on the train split where it tries to learn the patterns in the data. Then based on the patterns it has learnt it will be tested on the test split. In this entire process choosing the test split size is also very important. A rule of thumb is to use 80% of your data to train on and the other 20% to test on.

Machine learning Models

Machine learning models are majorly classified as supervised and unsupervised. If the model is supervised, it is divided into two categories: regression and classification. We will focus on the following machine learning models:

1. ***Logistic Regression:*** It is a basic classification algorithm which predicts the probability of a target variable.

2. ***K-nearest Neighbors:*** It's a ma- chine learning algorithm that's supervised. The idea behind nearest neighbor methods is to find a predetermined number of training samples that are closest in distance to the new point and use them to predict the mark. It makes no assumptions about the data and is typically used for classification tasks where little to no prior knowledge of the data distribution is available. Finding the k closest data points in the training set to the data point for which a target value is unavailable and assigning the average value of the identified data points to it is the aim of this algorithm.

3. ***Random Forest:*** Random forest is a supervised machine learning algorithm that can be used to solve problems in both classification and regression. It builds decision trees out of data samples, then gets predictions from each of them before voting on the best solution.

4. Naive Bayes: The Naïve Bayes classifier is a supervised machine learning algorithm, which is used for classification tasks, like text classification. It is also part of a family of generative learning algorithms, meaning that it seeks to model the distribution of inputs of a given class or category. Unlike discriminative classifiers, like logistic regression, it does not learn which features are most important to differentiate between classes.

5. Decision Tree: Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

6. Support Vector: Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.

we will find the other metrics for the logistic regression model:

A. ROC Curve

The metric compares the true positive rate with the false positive rate.

The True Positive Rate (TPR) is defined as follows:

$$TPR=\frac{TP}{TP+FN}$$

The False Positive Rate (FPR) is defined :

$$FPR=\frac{FP}{FP+TN}$$

It also provides us with AUC scores which denotes the area underneath the ROC curve

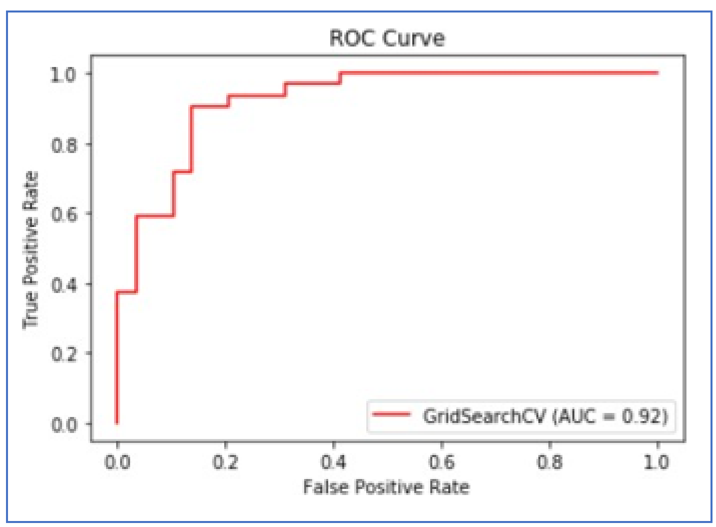


Fig 8-ROC Curve

B. Confusion Matrix

A confusion matrix is a table that is used to describe the output of a classification model/classifier by comparing the true values of the training and test datasets. It is divided into four parts, each of which is defined as follows:

1. True positives (TP): These are cases in which we expected yes (they have the disease) and they do.
2. Real negatives (TN): We predicted they wouldn't have the disorder, and they don't.
3. False positives (FP): We expected that they will have the disease, but they don't. (This is often referred to as a "Type I error.")
4. False negatives (FN): We expected that they will not have the disorder, but they do. (This is often referred to as a "Type II error.")

C. Classification Report

The Classification report is used to find the quality of predictions from a classification algorithm. It helps us to find how many predictions are correct and how many are wrong. More specifically, it gives us an understanding of True negatives and False Negatives, True Positives and False Positives, and uses them to predict the metrics of a classification. The main metrics found by the Classification report are accuracy, precision, recall, and f1- score.

D. Feature Importance

It refers to the techniques that assign a score to the input attributes/features with respect to the fact that which feature has the highest contribution in predicting the results for a given ma- chine learning model. For finding it we will use the coef_ attribute .The coef_ attribute is the coefficient of the features in the decision function. We can note that negative coef_ attribute denotes the presence of negative correlation.

Future Scope

In the future, the work could be improved by creating a web application premised on the logistic regression algorithm and by using a larger dataset than the one used in this study, which would help to provide better outcomes and aid health professionals in predicting heart disease efficiently and effectively.

Conclusion

With the rising number of deaths due to heart disease, it is becoming increasingly important to build a system that can effectively and accurately forecast heart disease. The motivation for the study was to find the most efficient ML algorithm for detection of heart diseases. This study compares the accuracy score of KNN, Logistic Regression and Random Forest for predicting heart disease using UCI machine learning repository dataset. The result of this study indicates that the Logistic regression algorithm is the most efficient algorithm with an accuracy score of 89% for prediction of heart disease. Accuracy of the algorithms in ma- chine learning depends upon the dataset that is used for training and testing purposes.