Akshita Mehta

Follow 31 Followers About

Machine Learning on Heart Disease Dataset

A Akshita Mehta Oct 12, 2019 · 7 min read

> *" Health is a state of complete physical, social and mental well being and not merely the absence of disease or infirmity. Health is thus a level of functional efficiency of living beings and a general condition of a person's mind, body and spirit, meaning it is free from illness, injury and pain. It is a resource of everyday life and a positive concept emphasizing physical capabilities."*
>
> ***Good Health is the Best Wealth!!!***

In this Medium article, we will learn about different classification techniques being applied to a heart disease which predicts whether a person is suffering from heart disease or not using Kaggle Heart Disease Dataset

About the Dataset

This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer-valued 0 = no disease and 1 = disease.

Attribute Information

- age : age in years
- sex : (1 = male; 0 = female)
- cp : chest pain type
- trestbps : resting blood pressure (in mm Hg on admission to the hospital)
- chol : serum cholestoral in mg/dl
- fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg : resting electrocardiographic results
- thalach : maximum heart rate achieved
- exang : exercise induced angina (1 = yes; 0 = no)
- oldpeak : ST depression induced by exercise relative to rest
- slope : the slope of the peak exercise ST segment
- ca : number of major vessels (0–3) colored by flourosopy
- thal : 1 = normal; 2 = fixed defect; 3 = reversable defect
- target : 1 = disease; 0 = no disease

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
52	1	0	125	212	0	1	168	0	1	2	2	3	0
53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
61	1	0	148	203	0	1	161	0	0	2	1	3	0
62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
58	0	0	100	248	0	0	122	0	1	1	0	2	1
58	1	0	114	318	0	2	140	0	4.4	0	3	1	0
55	1	0	160	289	0	0	145	1	0.8	1	1	3	0
46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
54	1	0	122	286	0	0	116	1	3.2	1	2	2	0
71	0	0	112	149	0	1	125	0	1.6	1	0	2	1
43	0	0	132	341	1	0	136	1	3	1	0	3	0
34	0	1	118	210	0	1	192	0	0.7	2	0	2	1
51	1	0	140	298	0	1	122	1	4.2	1	3	3	0
52	1	0	128	204	1	1	156	1	1	1	0	0	0
34	0	1	118	210	0	1	192	0	0.7	2	0	2	1
51	0	2	140	308	0	0	142	0	1.5	2	1	2	1
54	1	0	124	266	0	0	109	1	2.2	1	1	3	0
50	0	1	120	244	0	1	162	0	1.1	2	0	2	1
58	1	2	140	211	1	0	165	0	0	2	0	2	1
60	1	2	140	185	0	0	155	0	3	1	0	2	0

A snippet of csv file

Importing Libraries

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
```

Reading CSV File

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

Encoding Categorical Data

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction

```
from sklearn.preprocessing import OneHotEncoder
#cp
oneHotEncoder = OneHotEncoder(categorical_features=[2],
n_values='auto')
oneHotEncoder.fit(X)
X = oneHotEncoder.transform(X).toarray()
X = X[:, 1:]
#restecg
oneHotEncoder = OneHotEncoder(categorical_features=[8],
n_values='auto')
oneHotEncoder.fit(X)
X = oneHotEncoder.transform(X).toarray()
X = X[:, 1:]
#slope
oneHotEncoder = OneHotEncoder(categorical_features=[13],
n_values='auto')
oneHotEncoder.fit(X)
X = oneHotEncoder.transform(X).toarray()
X = X[:, 1:]
#ca
oneHotEncoder = OneHotEncoder(categorical_features=[15],
n_values='auto')
oneHotEncoder.fit(X)
X = oneHotEncoder.transform(X).toarray()
X = X[:, 1:]
#thal
oneHotEncoder = OneHotEncoder(categorical_features=[19],
n_values='auto')
oneHotEncoder.fit(X)
```

```
X = oneHotEncoder.transform(X).toarray()
X = X[:, 1:]

from sklearn.preprocessing import StandardScaler
scalerX = StandardScaler()
X = scalerX.fit_transform(X)
```

Splitting the Dataset

```
from sklearn.model_selection import train_test_split
XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size=0.3,
random_state=0)
```

What is Classification?

Classification is a technique to categorize our data into a desired and distinct number of classes where we can assign labels to each class.

Now we will be applying different classification techniques using the existing libraries in python.

Logistic Regression

> *Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.*

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(XTrain,yTrain)
yPred = classifier.predict(XTest)
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("Logistic Regression :")
print("Accuracy = ", accuracy)
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
```

```
Logistic Regression :
Accuracy =  0.8896103896103896
Mean Squared Error: 0.11038961038961038
R score: 0.5569282843240955
Mean Absolute Error: 0.11038961038961038
```

Decision Tree Classifier

> *Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.*

```
from sklearn.tree import DecisionTreeClassifier as DT
classifier = DT(criterion='entropy', random_state=0)
classifier.fit(XTrain,yTrain)
yPred = classifier.predict(XTest)
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("Decision Tree Classifier :")
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
print("Accuracy = ", accuracy)
```

```
Decision Tree Classifier :
Mean Squared Error: 0.01948051948051948
R score: 0.9218108737042522
Mean Absolute Error: 0.01948051948051948
Accuracy =  0.9805194805194806
```

Perceptron

> *Perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.*

```
from sklearn.linear_model import Perceptron
classifier = Perceptron(tol=1e-3, random_state=0)
classifier.fit(XTrain,yTrain)
yPred = classifier.predict(XTest)
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("Perceptron :")
print("Accuracy = ", accuracy)
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
```

```
Perceptron :
Accuracy =  0.8409090909090909
Mean Squared Error: 0.1590909090909091
R score: 0.361455468584726
Mean Absolute Error: 0.1590909090909091
```

K Nearest Neighbors

> *In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.*

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5, p=2,
metric='minkowski')
classifier.fit(XTrain,yTrain)
yPred = classifier.predict(XTest)
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("K Nearest Neighbors :")
print("Accuracy = ", accuracy)
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
```

```
K Nearest Neighbors :
Accuracy =  0.8831168831168831
Mean Squared Error: 0.11688311688311688
R score: 0.5308652422255129
Mean Absolute Error: 0.11688311688311688
```

Support Vector Machine

> *SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and*

> *then based on these transformations it finds an optimal boundary between the possible outputs.*

```
from sklearn.svm import SVC
classifier = SVC(kernel='linear',random_state=0)
classifier.fit(XTrain,yTrain)
yPred = classifier.predict(XTest)
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("Support Vector Machine :")
print("Accuracy = ", accuracy)
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
```

```
Support Vector Machine :
Accuracy =  0.8928571428571429
Mean Squared Error: 0.10714285714285714
R score: 0.5699598053733869
Mean Absolute Error: 0.10714285714285714
```

Gaussian Naive Bayes

> *A Gaussian Naive Bayes algorithm is a special type of NB algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a gaussian distribution i.e, normal distribution.*

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(XTrain,yTrain)
yPred = classifier.predict(XTest)
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("Gaussian Naive Bayes :")
print("Accuracy = ", accuracy)
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
```

```
Gaussian Naive Bayes :
Accuracy =  0.8733766233766234
Mean Squared Error: 0.1266233766233766
R score: 0.49177067907763905
Mean Absolute Error: 0.1266233766233766
```

Random Forest Classifier

> *It consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.*

```
from sklearn.ensemble import RandomForestClassifier as RF
classifier = RF(n_estimators=10, criterion='entropy', random_state=0)
classifier.fit(XTrain,yTrain)
yPred = classifier.predict(XTest)
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("Random Forest Classifier :")
print("Accuracy = ", accuracy)
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
```

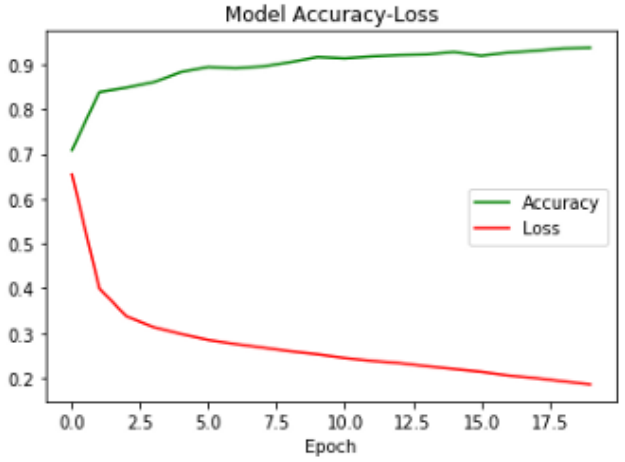
```
Random Forest Classifier :
Accuracy =  0.9902597402597403
Mean Squared Error: 0.00974025974025974
R score: 0.960905436852126
Mean Absolute Error: 0.00974025974025974
```

Artificial Neural Network

> *An artificial neural network is an interconnected group of nodes, inspired by a simplification of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.*

```
from keras.models import Sequential
from keras.layers import Dense

#Initialising ANN
classifier = Sequential()

#Adding the first hidden layer or the input layer
classifier.add(Dense(activation='relu',
                     kernel_initializer='uniform',
                     input_dim=22,
                     units=12))
#Adding the second hidden layer
classifier.add(Dense(activation='relu',
                     kernel_initializer='uniform',
                     units=12))
#Adding the output layer
classifier.add(Dense(activation='sigmoid',
                     kernel_initializer='uniform',
                     units=1))
```

```
#Compiling the ANN
classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
print(classifier.summary())

#Fitting the ANN
history = classifier.fit(XTrain, yTrain, batch_size=5, epochs=20,
verbose=1)
from matplotlib import pyplot as plt
plt.plot(history.history['acc'],'green')
plt.plot(history.history['loss'],'red')
plt.title('Model Accuracy-Loss')
plt.xlabel('Epoch')
plt.legend(['Accuracy','Loss'])
plt.show()

#Predicting the Test set Results
yPred = classifier.predict(XTest)
yPred = (yPred>0.5) #Since output is probability
mse = mean_squared_error(yTest,yPred)
r = r2_score(yTest,yPred)
mae = mean_absolute_error(yTest,yPred)
accuracy = accuracy_score(yTest,yPred)
print("Artificial Neural Network Classifier :")
print("Accuracy = ", accuracy)
print("Mean Squared Error:",mse)
print("R score:",r)
print("Mean Absolute Error:",mae)
```




```
Artificial Neural Network Classifier :
Accuracy =  0.9415584415584416
Mean Squared Error: 0.05844155844155844
R score: 0.7654326211127565
Mean Absolute Error: 0.05844155844155844
```

Rather than trying each algorithm separately, we will try a new method and display it using pretty tables

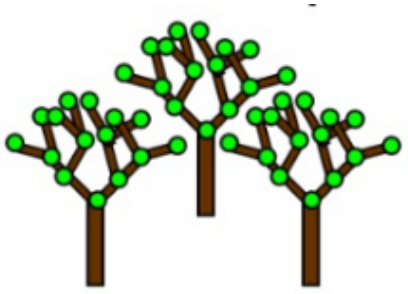
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Model","Accuracy", "Mean Squared Error", "R²
score","Mean Absolute Error"]
models = [
```

```
    LogisticRegression(),
    KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski'),
    SVC(kernel='linear',random_state=0),
    GaussianNB(),
    DT(criterion='entropy', random_state=0),
    RF(n_estimators=10, criterion='entropy', random_state=0),
    Perceptron(tol=1e-3, random_state=0)
]
for model in models:
    model.fit(XTrain, yTrain)
    yPred = model.predict(XTest)
    accuracy = accuracy_score(yTest,yPred)
    mse = mean_squared_error(yTest,yPred)
    r = r2_score(yTest,yPred)
    mae = mean_absolute_error(yTest,yPred)

table.add_row([type(model).__name__, format(accuracy,
'.3f'),format(mse, '.3f'),format(r, '.3f'),format(mae, '.3f')])

table.add_row(["Artificial Neural Network Classifier",0.954
,0.045,0.817,0.045])
print(table)
```

Model	Accuracy	Mean Squared Error	R² score	Mean Absolute Error
LogisticRegression	0.890	0.110	0.557	0.110
KNeighborsClassifier	0.883	0.117	0.531	0.117
SVC	0.893	0.107	0.570	0.107
GaussianNB	0.873	0.127	0.492	0.127
DecisionTreeClassifier	0.981	0.019	0.922	0.019
RandomForestClassifier	0.990	0.010	0.961	0.010
Perceptron	0.841	0.159	0.361	0.159
Artificial Neural Network Classifier	0.954	0.045	0.817	0.045

CONCLUSION

From the above output, we see that **RandomForestClassifier** works best for our dataset.



And if you want to get the whole code for the algorithms tested above, you can find it here :)

Blog by:

Akshita Mehta

Github LinkedIn

Machine Learning Classification Heart Disease Dataset