IBM machine learning supervised machine learning-classification peer graded assignment.

-R N V Siva Karthik

Contents

Problem description:

The problem statement shodden was to detect the presence of heart disease given the details of subject under study. The dataset was obtained from UCI ML repository, it contains 303 instances with 14 features.

Dataset details:

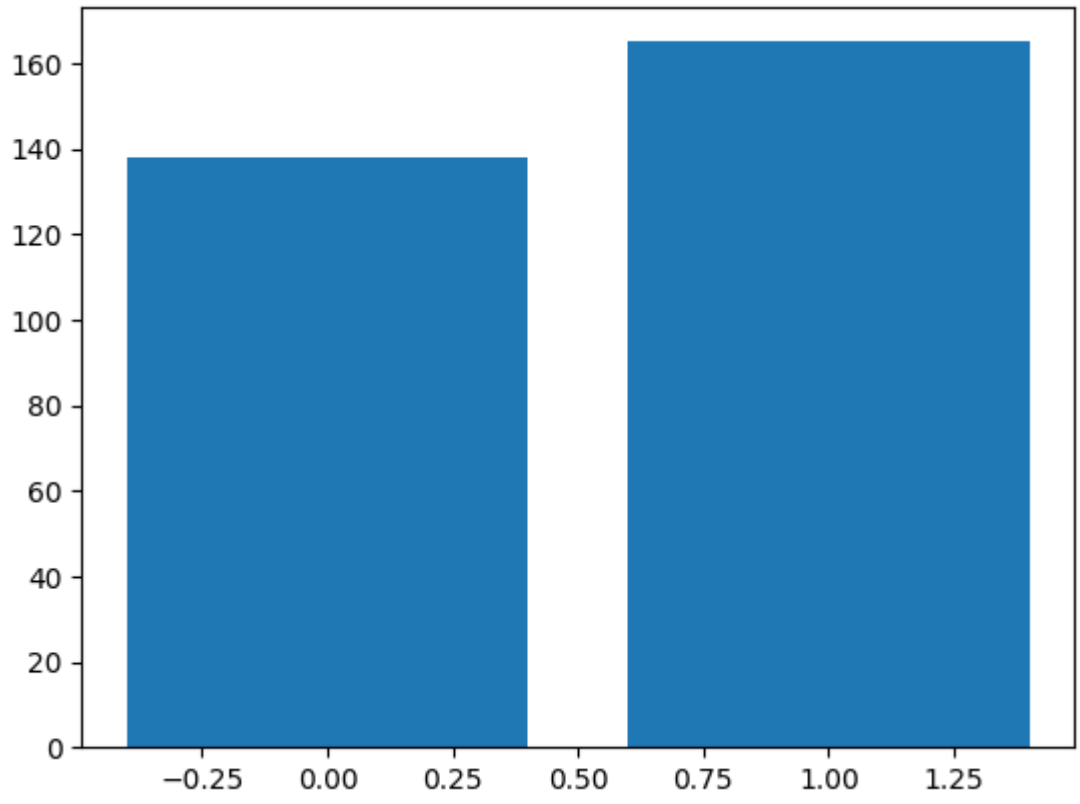
The dataset chosen was Heart diseases prediction dataset from the UCI machine learning data set repository. It contains the following fields:

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal'],
      dtype='object')
```

```
 #   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
dtypes: float64(1), int64(12)
memory usage: 30.9 KB
```

Data preprocessing:

when we plot the bar graph of the occurrence of each class label in target attribute we can find that there are about 140 positive labels and about 150 negative labels.



This eliminates the need of using measures for unbalanced classes problem.

There is no need of encoding as the data is already in either integer or float data types

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
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
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
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
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
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
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
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
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

We can observe that the data is also almost scaled that is most of the data is having min value as 0 the fields like age, trestbps, thalach, chol doesn't have it as 0 and it doesn't always have to be that way. Since we have already gone through the course and learnt that scaling doesn't have an effect while using tree based algorithms we shall also skip this step, we will be using logistic regression but, we will see that age is a very correlated factor in determining and so is cholesterol levels and other such variables and scaling them down to zero will minimize their importance.

```
x=data.drop(columns=['target'],axis=1)
y=data['target']
```
✓ 0.0s

The following code converts the data read from csv file into features data (x) and target data (y)

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=40)
```
✓ 0.0s

Here we use train test split method to split the x & y data sets into a pair of training and testing datasets xtrain, xtest, ytrain, ytest.

Building the logistic regression model:

I have chosen the base model to be the logistic regression. Using the above train data sets I fit this model.

The following are the results obtained:

```
accuracy: 0.9120879120879121
precision: 0.9607843137254902
recall: 0.8909090909090909
f1 score: 0.9245283018867925
confusion matrix:
 [[34  2]
 [ 6 49]]
```

As we can see the results are very nice, an accuracy of about 91%.

Building the Random Forest Classification model:

I have used GridSearchCV method to go through various n_estimators, samples, max_depth values to get the best model.

The following results were obtained:

```
accuracy: 0.8681318681318682
precision: 0.8823529411764706
recall: 0.8823529411764706
f1 score: 0.8823529411764706
confusion matrix:
 [[34  6]
 [ 6 45]]
```

As we can see the results were not quite as accurate as logistic regression model but considerably good.

The following were the best parameters:

```
{'ran__max_depth': 5, 'ran__max_samples': 0.2, 'ran__n_estimators': 100}
```

Building a k nearest neighbour method:

This method performed poor than I expected, I ran it with gridsearchCV method with n_neighbors as a hyperparameter .

The following results were obtained:

```
accuracy: 0.7362637362637363
precision: 0.7362637362637363
recall: 0.7362637362637363
f1 score: 0.7362637362637363
confusion matrix:
 [[29 13]
 [11 38]]
```

As we can see the accuracy is only about 73%

Building the SVM model:

Cross validation was used to tune the hyperparameters c, gamma, kernal of 'rbf' was used.

The following results were obtained from SVM model:

```
accuracy: 0.8241758241758241
precision: 0.8241758241758241
recall: 0.8241758241758241
f1 score: 0.8241758241758241
confusion matrix:
 [[30  6]
 [10 45]]
```

As we can see it is not as good as logistic regression or randon forest but not as bad as knn.

Building the Gradient Boosting Classifier model:

Cross validation was used to tune the hyperparameters n_estimators and subsample size.

The following results were obtained:

```
accuracy: 0.8571428571428571
precision: 0.9019607843137255
recall: 0.8518518518518519
f1 score: 0.8761904761904761
confusion matrix:
 [[32  5]
 [ 8 46]]
```

As we can see the results are better than SVM.

Recommended model and key findings:

The model that yielded the best results was the logistic regression model. With an accuracy of 91%. The choice of the random_state hyperparameter in the train_test_split method greatly effected the results in this case. By using a serializer like pickle we can save our models for further use also.

Conclusion:

The use of gridsearchcv made my job easier. Since it choose the best hyperparameters and the best model for me that is the least overfitted and has the perfect bias variance balance.