central test for logisticR

March 18, 2024

[3]:

```
# importing necessary libraries
import pandas as pd
import numpy as np
```

[4]:

```
#loading data
data=pd.read_csv("C:\\Users\\SAMUEL K\\Desktop\\banana_quality.csv")
data.head()
```

[4]:

```
       Size    Weight  Sweetness  Softness  HarvestTime  Ripeness   Acidity  \
0 -1.924968  0.468078   3.077832 -1.472177     0.294799  2.435570  0.271290
1 -2.409751  0.486870   0.346921 -2.495099    -0.892213  2.067549  0.307325
2 -0.357607  1.483176   1.568452 -2.645145    -0.647267  3.090643  1.427322
3 -0.868524  1.566201   1.889605 -1.273761    -1.006278  1.873001  0.477862
4  0.651825  1.319199  -0.022459 -1.209709    -1.430692  1.078345  2.812442

  Quality
0    Good
1    Good
2    Good
3    Good
4    Good
```

[5]:

```
# defining x variables
x=data[['Size','Weight','Sweetness','Softness','HarvestTime','Ripeness','Acidity']]
x.head()
```

[5]:

```
       Size    Weight  Sweetness  Softness  HarvestTime  Ripeness   Acidity
0 -1.924968  0.468078   3.077832 -1.472177     0.294799  2.435570  0.271290
1 -2.409751  0.486870   0.346921 -2.495099    -0.892213  2.067549  0.307325
2 -0.357607  1.483176   1.568452 -2.645145    -0.647267  3.090643  1.427322
3 -0.868524  1.566201   1.889605 -1.273761    -1.006278  1.873001  0.477862
4  0.651825  1.319199  -0.022459 -1.209709    -1.430692  1.078345  2.812442
```

[6]:

```
# dropping the unwanted x columns
x=data.drop(columns=["Quality"])
x.head()
```

[6]:
```
        Size    Weight  Sweetness  Softness  HarvestTime  Ripeness   Acidity
0 -1.924968  0.468078   3.077832 -1.472177     0.294799  2.435570  0.271290
1 -2.409751  0.486870   0.346921 -2.495099    -0.892213  2.067549  0.307325
2 -0.357607  1.483176   1.568452 -2.645145    -0.647267  3.090643  1.427322
3 -0.868524  1.566201   1.889605 -1.273761    -1.006278  1.873001  0.477862
4  0.651825  1.319199  -0.022459 -1.209709    -1.430692  1.078345  2.812442
```

[7]:
```python
# defining y variables
y=data["Quality"]
y.head()
```

[7]:
```
0    Good
1    Good
2    Good
3    Good
4    Good
Name: Quality, dtype: object
```

[8]:
```python
#importing necessary libarries
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import recall_score,accuracy_score,precision_score,f1_score
```

[9]:
```python
# splitting data
x_train, x_test, y_train, y_test= train_test_split(x,y,test_size=0.2,␣
 ↪random_state=42)
```

[10]:
```python
#building a model
model=LogisticRegression()
model.fit(x_train,y_train)
```

[10]:
```
LogisticRegression()
```

[11]:
```python
#making aprediction for y
pred=model.predict(x_test)
pred
```

[11]:
```
array(['Bad', 'Good', 'Good', …, 'Good', 'Bad', 'Bad'], dtype=object)
```

[12]:
```python
# calculating for recall score using pos_label
recall=recall_score(y_test,pred ,pos_label="Good")
recall
```

[12]:
```
0.8888888888888888
```

[13]:
```python
#calculating for accuray score
accuracy=accuracy_score(y_test,pred)
```

```
accuracy
```

[13]:

```
0.879375
```

[14]:

```
#calculating precision score
precision=precision_score(y_test,pred,pos_label="Good")
precision
```

[14]:

```
0.8771084337349397
```

[15]:

```
f1=f1_score(y_test,pred,pos_label="Good")
f1
```

[15]:

```
0.8829593693147362
```

[16]:

```
#standardizing the data
scaler=StandardScaler()
x_train_scaled=scaler.fit_transform(x_train)
x_test_scaled=scaler.transform(x_test)
#x_train1
```

[17]:

```
# define the hyperparameters to tune
model = LogisticRegression()
param_grid={'C':[1],
            'penalty':['l1', 'l2', 'elasticnet', None]
           }
param_grid
```

[17]:

```
{'C': [1], 'penalty': ['l1', 'l2', 'elasticnet', None]}
```

[18]:

```
#performing GridSearch cross validation for hyperparameter
grid_search= GridSearchCV(model, param_grid,cv=5,n_jobs=-1)
grid_search
```

[18]:

```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [1], 'penalty': ['l1', 'l2', 'elasticnet', None]})
```

[19]:

```
#fitting grid search into the model
import warnings
warnings.filterwarnings("ignore")
grid_search.fit(x_train,y_train)
```

[19]:

```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [1], 'penalty': ['l1', 'l2', 'elasticnet', None]})
```

[20]:

```
# best parameters
best_params = grid_search.best_params_
best_params
```

```
[20]: {'C': 1, 'penalty': 'l2'}
```

```
[21]: # fitting the best param with x and y train
      best_model= LogisticRegression(**best_params)
      best_model.fit(x_train,y_train)
```

```
[21]: LogisticRegression(C=1)
```

```
[22]: # predicting for y to view if model can predict the values
      y_pred= best_model.predict(x_test)
      y_pred
```

```
[22]: array(['Bad', 'Good', 'Good', …, 'Good', 'Bad', 'Bad'], dtype=object)
```

```
[24]: #calculating for accuray score
      accuracy2=accuracy_score(y_test,y_pred)
      accuracy2
```

```
[24]: 0.879375
```

```
[26]: if accuracy2> accuracy:
          print("The optimized model better than ordinary model")

      elif accuracy2<accuracy:
          print("The ordinary model is greater than optimized model")
      else:
          print("Both models have the same accuracy")
```

```
Both models have the same accuracy
```

```
[ ]:
```