Lab10. Patients Physical Activities Prediction using Boosting

Objectives

In this lab, you will recognize physical activities such as ‘laying’, ‘sitting’ or ‘walking’ using Gradient Boosting, AdaBoost and VotingClassifiers.

Learning Outcomes

After completing this lab, you will be able to

- Create a small dataset with selected rows based on fewer target labels
- Build GradientBoostingClassifier, fit and predict on test data
- Print accuracy and classification report
- Find the best no. of decision trees and learning rate using GridSearch and Cross Validation
- Build AdaBoost classifier model with GridSearchCV, fit and predict
- Select best parameter values for n_estimators and learning_rate
- Build LogisticRegressionCV model, fit, predict and print scores
- Build VotingClassifier using other models, fit, predict and print scores
- Interpret results and parameter values
- Change parameter values and play around with models

Import necessary library

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import precision_score, recall_score,accuracy_score,roc_auc_score,clas
from sklearn.ensemble import GradientBoostingClassifier,AdaBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_val_score
```

Step 1 [Understand Data]

In [2]:

```
hac = pd.read_csv("Human_Activity_Data.csv")
```

In [3]:

```
hac.head()
```

Out[3]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672

5 rows × 562 columns

In [4]:

```
hac.columns
```

Out[4]:

```
Index(['tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y', 'tBodyAcc-mean()-Z',
       'tBodyAcc-std()-X', 'tBodyAcc-std()-Y', 'tBodyAcc-std()-Z',
       'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y', 'tBodyAcc-mad()-Z',
       'tBodyAcc-max()-X',
       ...
       'fBodyBodyGyroJerkMag-skewness()', 'fBodyBodyGyroJerkMag-kurtosis()',
       'angle(tBodyAccMean,gravity)', 'angle(tBodyAccJerkMean),gravityMea
n)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity'],
      dtype='object', length=562)
```

In [5]:

```
hac.shape
```

Out[5]:

```
(10299, 562)
```

In [6]:

```
hacl.dtypes
```

Out[6]:

```
tBodyAcc-mean()-X                       float64
tBodyAcc-mean()-Y                       float64
tBodyAcc-mean()-Z                       float64
tBodyAcc-std()-X                        float64
tBodyAcc-std()-Y                        float64
                                          ...
angle(tBodyGyroJerkMean,gravityMean)    float64
angle(X,gravityMean)                    float64
angle(Y,gravityMean)                    float64
angle(Z,gravityMean)                    float64
Activity                                 object
Length: 562, dtype: object
```

In [7]:

```
hac.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10299 entries, 0 to 10298
Columns: 562 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), object(1)
memory usage: 44.2+ MB
```

In [8]:

```
hac.value_counts()
```

Out[8]:

```
tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-std()-X
tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()-Y  tB
odyAcc-mad()-Z  tBodyAcc-max()-X  tBodyAcc-max()-Y  tBodyAcc-max()-Z  tBod
yAcc-min()-X  tBodyAcc-min()-Y  tBodyAcc-min()-Z  tBodyAcc-sma()  tBodyAcc
-energy()-X  tBodyAcc-energy()-Y  tBodyAcc-energy()-Z  tBodyAcc-iqr()-X  t
BodyAcc-iqr()-Y  tBodyAcc-iqr()-Z  tBodyAcc-entropy()-X  tBodyAcc-entropy
()-Y  tBodyAcc-entropy()-Z  tBodyAcc-arCoeff()-X,1  tBodyAcc-arCoeff()-X,2
tBodyAcc-arCoeff()-X,3  tBodyAcc-arCoeff()-X,4  tBodyAcc-arCoeff()-Y,1  tB
odyAcc-arCoeff()-Y,2  tBodyAcc-arCoeff()-Y,3  tBodyAcc-arCoeff()-Y,4  tBod
yAcc-arCoeff()-Z,1  tBodyAcc-arCoeff()-Z,2  tBodyAcc-arCoeff()-Z,3  tBodyA
cc-arCoeff()-Z,4  tBodyAcc-correlation()-X,Y  tBodyAcc-correlation()-X,Z
tBodyAcc-correlation()-Y,Z  tGravityAcc-mean()-X  tGravityAcc-mean()-Y  tG
ravityAcc-mean()-Z  tGravityAcc-std()-X  tGravityAcc-std()-Y  tGravityAcc-
std()-Z  tGravityAcc-mad()-X  tGravityAcc-mad()-Y  tGravityAcc-mad()-Z  tG
ravityAcc-max()-X  tGravityAcc-max()-Y  tGravityAcc-max()-Z  tGravityAcc-m
in()-X  tGravityAcc-min()-Y  tGravityAcc-min()-Z  tGravityAcc-sma()  tGrav
ityAcc-energy()-X  tGravityAcc-energy()-Y  tGravityAcc-energy()-Z  tGravit
yAcc-iqr()-X  tGravityAcc-iqr()-Y  tGravityAcc-iqr()-Z  tGravityAcc-entrop
```

In [9]:

```
label_encoder = LabelEncoder()
hac["label_Activity"] = label_encoder.fit_transform(hac["Activity"])
```

Step2. [Build a small dataset]

In [10]:

```
hac.Activity.value_counts()
```

Out[10]:

```
LAYING                1944
STANDING              1906
SITTING               1777
WALKING               1722
WALKING_UPSTAIRS      1544
WALKING_DOWNSTAIRS    1406
Name: Activity, dtype: int64
```

In [11]:

```
hac.label_Activity.value_counts()
```

Out[11]:

```
0    1944
2    1906
1    1777
3    1722
5    1544
4    1406
Name: label_Activity, dtype: int64
```

Take first 3000 samples for each 6 activities and build classifier

In [12]:

```
sam  = hac[hac['Activity']=='LAYING'][:500]
sam1 = hac[hac['Activity']=='SITTING'][:500]
sam2 = hac[hac['Activity']=='WALKING'][:500]
sam3 = hac[hac['Activity']=='STANDING'][:500]
sam4 = hac[hac['Activity']=='WALKING_UPSTAIRS'][:500]
sam5 = hac[hac['Activity']=='WALKING_DOWNSTAIRS'][:500]
```

In [13]:

```
hac_new = pd.concat([sam,sam1,sam2,sam3,sam4,sam5])
```

In [14]:

```
hac_new.to_csv("human_activity_clipped3000.csv")
```

In [15]:

```
hac_new = pd.read_csv("human_activity_clipped3000.csv")
```

In [16]:

```
hac_new.head()
```

Out[16]:

	Unnamed: 0	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X
0	51	0.403474	-0.015074	-0.118167	-0.914811	-0.895231	-0.891748	-0.917696
1	52	0.278373	-0.020561	-0.096825	-0.984883	-0.991118	-0.982112	-0.987985
2	53	0.276555	-0.017869	-0.107621	-0.994195	-0.996372	-0.995615	-0.994901
3	54	0.279575	-0.017276	-0.109481	-0.996135	-0.995812	-0.998689	-0.996393
4	55	0.276527	-0.016819	-0.107983	-0.996775	-0.997256	-0.995422	-0.997167

5 rows × 564 columns

In [17]:

```
hac_new.shape
```

Out[17]:

```
(3000, 564)
```

In [18]:

```
hac_new.columns
```

Out[18]:

```
Index(['Unnamed: 0', 'tBodyAcc-mean()-X', 'tBodyAcc-mean()-Y',
       'tBodyAcc-mean()-Z', 'tBodyAcc-std()-X', 'tBodyAcc-std()-Y',
       'tBodyAcc-std()-Z', 'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y',
       'tBodyAcc-mad()-Z',
       ...
       'fBodyBodyGyroJerkMag-kurtosis()', 'angle(tBodyAccMean,gravity)',
       'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity',
       'label_Activity'],
      dtype='object', length=564)
```

In [19]:

```
hac_new.dtypes
```

Out[19]:

```
Unnamed: 0                int64
tBodyAcc-mean()-X       float64
tBodyAcc-mean()-Y       float64
tBodyAcc-mean()-Z       float64
tBodyAcc-std()-X        float64
                         ...
angle(X,gravityMean)    float64
angle(Y,gravityMean)    float64
angle(Z,gravityMean)    float64
Activity                 object
label_Activity            int64
Length: 564, dtype: object
```

In [20]:

```
hac_new.value_counts()
```

Out[20]:

```
Unnamed: 0  tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBody
Acc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAc
c-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  tBodyAcc-max()-Y  tBodyAcc-
max()-Z  tBodyAcc-min()-X  tBodyAcc-min()-Y  tBodyAcc-min()-Z  tBodyAcc-sm
a()  tBodyAcc-energy()-X  tBodyAcc-energy()-Y  tBodyAcc-energy()-Z  tBodyA
cc-iqr()-X  tBodyAcc-iqr()-Y  tBodyAcc-iqr()-Z  tBodyAcc-entropy()-X  tBod
yAcc-entropy()-Y  tBodyAcc-entropy()-Z  tBodyAcc-arCoeff()-X,1  tBodyAcc-a
rCoeff()-X,2  tBodyAcc-arCoeff()-X,3  tBodyAcc-arCoeff()-X,4  tBodyAcc-arC
oeff()-Y,1  tBodyAcc-arCoeff()-Y,2  tBodyAcc-arCoeff()-Y,3  tBodyAcc-arCoe
ff()-Y,4  tBodyAcc-arCoeff()-Z,1  tBodyAcc-arCoeff()-Z,2  tBodyAcc-arCoeff
()-Z,3  tBodyAcc-arCoeff()-Z,4  tBodyAcc-correlation()-X,Y  tBodyAcc-corre
lation()-X,Z  tBodyAcc-correlation()-Y,Z  tGravityAcc-mean()-X  tGravityAc
c-mean()-Y  tGravityAcc-mean()-Z  tGravityAcc-std()-X  tGravityAcc-std()-Y
tGravityAcc-std()-Z  tGravityAcc-mad()-X  tGravityAcc-mad()-Y  tGravityAcc
-mad()-Z  tGravityAcc-max()-X  tGravityAcc-max()-Y  tGravityAcc-max()-Z  t
GravityAcc-min()-X  tGravityAcc-min()-Y  tGravityAcc-min()-Z  tGravityAcc-
sma()  tGravityAcc-energy()-X  tGravityAcc-energy()-Y  tGravityAcc-energy
()-Z  tGravityAcc-iqr()-X  tGravityAcc-iqr()-Y  tGravityAcc-iqr()-Z  tGrav
```

In [21]:

```
X_=hac_new.drop(['Activity','label_Activity'],axis=1)
y_=hac_new.Activity
X__train,X__test,y__train,y__test = train_test_split(X_,y_,test_size=0.2,random_state=22)
```

Build GradientBoostingClassifier for 3000 samples

In [22]:

```
gbc_model = GradientBoostingClassifier(subsample=0.5,n_estimators=100,learning_rate=1.0,max
gbc_model.fit(X__train,y__train)
y__predict=gbc_model.predict(X__test)
```

In [23]:

```
print(accuracy_score(y__test,y__predict))
print(classification_report(y__test,y__predict))
```

```
0.705
              precision    recall  f1-score   support

           0       0.36      0.11      0.16        94
           1       0.57      0.70      0.63        97
           2       0.54      0.74      0.63       101
           3       0.82      0.92      0.87       105
           4       0.87      0.87      0.87       103
           5       0.88      0.83      0.86       100

    accuracy                           0.70       600
   macro avg       0.68      0.70      0.67       600
weighted avg       0.68      0.70      0.68       600
```

Build AdaBoostClassifier for 3000 samples

In [24]:

```
abc1 = DecisionTreeClassifier(max_features=4)
abc2 = AdaBoostClassifier(base_estimator=abc1,random_state=0)
par_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

In [25]:

```
gscv_model1 = GridSearchCV(abc2,par_grid,cv=10,n_jobs=-1)
gscv_model1.fit(X__train,y__train)
y__predict1=gscv_model1.predict(X__test)
```

In [26]:

```
print(accuracy_score(y__test,y__predict1))
print(classification_report(y__test,y__predict1))
```

```
0.7716666666666666
              precision    recall  f1-score   support

           0       0.76      0.69      0.72        94
           1       0.62      0.65      0.63        97
           2       0.69      0.71      0.70       101
           3       0.82      0.92      0.87       105
           4       0.90      0.81      0.85       103
           5       0.86      0.83      0.85       100

    accuracy                           0.77       600
   macro avg       0.77      0.77      0.77       600
weighted avg       0.78      0.77      0.77       600
```

In [27]:

```
gscv_model1.best_estimator_
```

Out[27]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_features=4),
                   learning_rate=0.01, n_estimators=100, random_state=0)
```

Build LogisticRegressionCV classifier for 3000 samples

In [28]:

```
lrcv_model2 = LogisticRegressionCV(cv=4,Cs=5,penalty='l2')
lrcv_model2.fit(X__train,y__train)
y__predict2=lrcv_model2.predict(X__test)
```

In [29]:

```
print(accuracy_score(y__test,y__predict2))
print(classification_report(y__test,y__predict2))
```

```
0.9766666666666667
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        94
           1       0.96      0.92      0.94        97
           2       0.93      0.96      0.95       101
           3       1.00      0.99      1.00       105
           4       0.98      1.00      0.99       103
           5       1.00      0.99      0.99       100

    accuracy                           0.98       600
   macro avg       0.98      0.98      0.98       600
weighted avg       0.98      0.98      0.98       600
```

Find Best no. of trees and Best Learning Rate using Grid Search and Cross Validation for 3000 samples

In [30]:

```
Par_grid={'n_estimators':[50,100,200, 400],'learning_rate':[0.1,0.01]}
```

In [31]:

```
all_scores_1 = cross_val_score(estimator=gbc_model,X=X__train,y=y__train,cv=5)
print(all_scores_1)
```

```
[0.10416667 0.55416667 0.72708333 0.80416667 0.95      ]
```

In [32]:

```
gs_model3 = GridSearchCV(estimator=gbc_model,param_grid=Par_grid,cv=5,n_jobs=-1)
gs_model3.fit(X__train,y__train)
y__predict3=gs_model3.predict(X__test)
```

In [33]:

```
print(accuracy_score(y__test,y__predict3))
print(classification_report(y__test,y__predict3))
```

```
0.985
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        94
           1       0.97      0.95      0.96        97
           2       0.95      0.97      0.96       101
           3       1.00      0.99      1.00       105
           4       0.99      1.00      1.00       103
           5       1.00      1.00      1.00       100

    accuracy                           0.98       600
   macro avg       0.99      0.98      0.98       600
weighted avg       0.99      0.98      0.98       600
```

In [34]:

```
gs_model3.best_estimator_
```

Out[34]:

```
GradientBoostingClassifier(max_depth=1, max_features=4, n_estimators=400,
                           random_state=0, subsample=0.5)
```

Build VotingClassifier for 3000 samples

In [35]:

```
vc_model4=VotingClassifier(estimators=[('lr',gs_model3),('gbc',abc2)],voting='soft')
vc_model4.fit(X__train,y__train)
y__predict4=vc_model4.predict(X__test)
```

In [36]:

```
print(accuracy_score(y__test,y__predict4))
print(classification_report(y__test,y__predict4))
```

```
0.7716666666666666
              precision    recall  f1-score   support

           0       0.76      0.69      0.72        94
           1       0.62      0.65      0.63        97
           2       0.69      0.71      0.70       101
           3       0.82      0.92      0.87       105
           4       0.90      0.81      0.85       103
           5       0.86      0.83      0.85       100

    accuracy                           0.77       600
   macro avg       0.77      0.77      0.77       600
weighted avg       0.78      0.77      0.77       600
```

From this 3000 smaples you should take 1500 samples for each activites

In [37]:

```
samp = hac_new[hac_new['Activity']=='LAYING'][:500]
samp1 = hac_new[hac_new['Activity']=='SITTING'][:500]
samp2 = hac_new[hac_new['Activity']=='WALKING'][:500]
```

In [38]:

```
hac1 = pd.concat([samp,samp1,samp2])
```

In [39]:

```
hac1.to_csv("human_activity_clipped1500.csv")
```

In [40]:

```
hac1 = pd.read_csv("human_activity_clipped1500.csv")
```

In [41]:

```
hac1.head()
```

Out[41]:

	Unnamed: 0	Unnamed: 0.1	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z
0	0	51	0.403474	-0.015074	-0.118167	-0.914811	-0.895231	-0.891748
1	1	52	0.278373	-0.020561	-0.096825	-0.984883	-0.991118	-0.982112
2	2	53	0.276555	-0.017869	-0.107621	-0.994195	-0.996372	-0.995615
3	3	54	0.279575	-0.017276	-0.109481	-0.996135	-0.995812	-0.998689
4	4	55	0.276527	-0.016819	-0.107983	-0.996775	-0.997256	-0.995422

5 rows × 565 columns

In [42]:

```
hac1.shape
```

Out[42]:

```
(1500, 565)
```

In [43]:

```
hac1.columns
```

Out[43]:

```
Index(['Unnamed: 0', 'Unnamed: 0.1', 'tBodyAcc-mean()-X', 'tBodyAcc-mean()-
Y',
       'tBodyAcc-mean()-Z', 'tBodyAcc-std()-X', 'tBodyAcc-std()-Y',
       'tBodyAcc-std()-Z', 'tBodyAcc-mad()-X', 'tBodyAcc-mad()-Y',
       ...
       'fBodyBodyGyroJerkMag-kurtosis()', 'angle(tBodyAccMean,gravity)',
       'angle(tBodyAccJerkMean),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity',
       'label_Activity'],
      dtype='object', length=565)
```

In [44]:

```
hac1.dtypes
```

Out[44]:

```
Unnamed: 0                int64
Unnamed: 0.1              int64
tBodyAcc-mean()-X       float64
tBodyAcc-mean()-Y       float64
tBodyAcc-mean()-Z       float64
                         ...
angle(X,gravityMean)    float64
angle(Y,gravityMean)    float64
angle(Z,gravityMean)    float64
Activity                 object
label_Activity            int64
Length: 565, dtype: object
```

In [45]:

```
hac1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Columns: 565 entries, Unnamed: 0 to label_Activity
dtypes: float64(561), int64(3), object(1)
memory usage: 6.5+ MB
```

In [46]:

```
hac1.value_counts()
```

Out[46]:

```
Unnamed: 0  Unnamed: 0.1  tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-m
ean()-Z  tBodyAcc-std()-X  tBodyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-ma
d()-X  tBodyAcc-mad()-Y  tBodyAcc-mad()-Z  tBodyAcc-max()-X  tBodyAcc-max
()-Y  tBodyAcc-max()-Z  tBodyAcc-min()-X  tBodyAcc-min()-Y  tBodyAcc-min()
-Z  tBodyAcc-sma()  tBodyAcc-energy()-X  tBodyAcc-energy()-Y  tBodyAcc-ene
rgy()-Z  tBodyAcc-iqr()-X  tBodyAcc-iqr()-Y  tBodyAcc-iqr()-Z  tBodyAcc-en
tropy()-X  tBodyAcc-entropy()-Y  tBodyAcc-entropy()-Z  tBodyAcc-arCoeff()-
X,1  tBodyAcc-arCoeff()-X,2  tBodyAcc-arCoeff()-X,3  tBodyAcc-arCoeff()-X,
4  tBodyAcc-arCoeff()-Y,1  tBodyAcc-arCoeff()-Y,2  tBodyAcc-arCoeff()-Y,3
tBodyAcc-arCoeff()-Y,4  tBodyAcc-arCoeff()-Z,1  tBodyAcc-arCoeff()-Z,2  tB
odyAcc-arCoeff()-Z,3  tBodyAcc-arCoeff()-Z,4  tBodyAcc-correlation()-X,Y
tBodyAcc-correlation()-X,Z  tBodyAcc-correlation()-Y,Z  tGravityAcc-mean()
-X  tGravityAcc-mean()-Y  tGravityAcc-mean()-Z  tGravityAcc-std()-X  tGrav
ityAcc-std()-Y  tGravityAcc-std()-Z  tGravityAcc-mad()-X  tGravityAcc-mad
()-Y  tGravityAcc-mad()-Z  tGravityAcc-max()-X  tGravityAcc-max()-Y  tGrav
ityAcc-max()-Z  tGravityAcc-min()-X  tGravityAcc-min()-Y  tGravityAcc-min
()-Z  tGravityAcc-sma()  tGravityAcc-energy()-X  tGravityAcc-energy()-Y  t
GravityAcc-energy()-Z  tGravityAcc-iqr()-X  tGravityAcc-iqr()-Y  tGravityA
```

Step3. [Build GradientBoostingClassifier]

In [47]:

```
X=hac1.drop(['Activity','label_Activity'],axis=1)
y=hac1.Activity
```

In [48]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=22)
```

Create GradientBoostingClassifier, fit and predict

In [49]:

```
model = GradientBoostingClassifier(subsample=0.5,n_estimators=100,learning_rate=1.0,max_dep
```

In [50]:

```
model.fit(X_train,y_train)
```

Out[50]:

```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, max_features=4,
                           random_state=42, subsample=0.5)
```

In [51]:

```
y_predict=model.predict(X_test)
y_predict
```

Out[51]:

```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

Print accuracy and classification report

In [52]:

```
print(accuracy_score(y_test,y_predict))
```

Out[52]:

```
1.0
```

In [53]:

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

Step4. [Find Best no. of trees and Best Learning Rate using Grid Search and Cross Validation]

Create GridSearchCV model with GradientBoostingClassifier

In [55]:

```
all_scores = cross_val_score(estimator=model,X=X_train,y=y_train,cv=5)
print(all_scores)
```

```
[0.99166667 1.         1.         1.         1.        ]
```

In [56]:

```
model2 = GridSearchCV(estimator=model,param_grid=Param_grid,cv=5,n_jobs=-1)
```

Parameters: param_grid = {'n_estimators': [50, 100, 200, 400], 'learning_rate': [0.1, 0.01]}

In [54]:

```
Param_grid={'n_estimators':[50, 100, 200, 400],'learning_rate':[0.1, 0.01]}
```

Perform fit and predict

In [57]:

```
model2.fit(X_train,y_train)
```

Out[57]:

```
GridSearchCV(cv=5,
             estimator=GradientBoostingClassifier(learning_rate=1.0,
                                                  max_depth=1, max_features=
4,
                                                  random_state=42,
                                                  subsample=0.5),
             n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.01],
                         'n_estimators': [50, 100, 200, 400]})
```

In [58]:

```
y_pred2=model2.predict(X_test)
y_pred2
```

Out[58]:

```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

Print accuracy, classification report

In [59]:

```
accuracy_score(y_test,y_predict1)
```

Out[59]:

```
1.0
```

In [60]:

```
print(classification_report(y_test,y_predict1))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

Print best parameters such as best no. of trees and learning rate. Use the attribute best_estimator_

In [61]:

```
print(model1.best_estimator_)
```

Out[61]:

```
GradientBoostingClassifier(max_depth=1, max_features=4, n_estimators=200,
                           random_state=42, subsample=0.5)
```

Step5. [Build AdaBoostClassifier]

Create AdaBoostClassifier with DecisionTreeClassifier

Parameters: param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}

In [62]:

```
abc = DecisionTreeClassifier()
model2 = AdaBoostClassifier(base_estimator=abc,random_state=0)
param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

Create GridSearchCV with AdaBoostClassifier model that you created as before

In [63]:

```
model3 = GridSearchCV(model2,param_grid,cv=5,n_jobs=-1)
```

Perform fit, predict

In [64]:

```
model3.fit(X_train,y_train)
```

Out[64]:

```
GridSearchCV(cv=10,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassif
ier(max_features=4),
                                          random_state=0),
             n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.001],
                         'n_estimators': [100, 150, 200]})
```

In [65]:

```
y_predict2=model3.predict(X_test)
y_predict2
```

Out[65]:

```
array([0, 1, 1, 1, 3, 0, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 1, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 1, 0, 0, 1, 1, 0, 3, 3, 1, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 1, 3, 1, 3, 0, 0, 3, 0, 3, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       1, 1, 0, 3, 3, 3, 3, 3, 3, 0, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 0, 1, 0, 1, 1, 0, 0, 3, 0, 0, 1, 3, 0, 0,
       1, 1, 1, 3, 0, 0, 1, 1, 3, 1, 1, 0, 3, 1, 1, 3, 0, 3, 1, 1, 3, 0,
       1, 1, 3, 1, 1, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 0, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 0, 1, 3, 0, 0, 1, 1, 0, 1, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 0, 0, 0, 3, 0, 1, 0, 0, 1, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

Print accuracy, classification report

In [66]:

```
print(accuracy_score(y_test,y_predict2))
```

Out[66]:

```
0.9133333333333333
```

In [67]:

```
print(classification_report(y_test,y_predict2))
```

```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       100
           1       0.86      0.88      0.87       100
           3       1.00      1.00      1.00       100

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300
```

Print best parameters such as best no. of trees and learning rate. Use the attribute best_estimator_

In [68]:

```
print(model3.best_estimator_)
```

Out[68]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_features=4),
                   learning_rate=0.01, n_estimators=100, random_state=0)
```

Step6. [Build LogisticRegressionCV classifier]

Create a LogisticRegressionCV model with the parameters Cs=5, cv=4, penalty='l2'.

In [69]:

```
model4 = LogisticRegressionCV(cv=4,Cs=5,penalty='l2')
```

Perform fit and predict

In [70]:

```
model4.fit(X_train,y_train)
```

Out[70]:

```
LogisticRegressionCV(Cs=5, cv=4)
```

In [71]:

```
y_predict3=model4.predict(X_test)
y_predict3
```

Out[71]:

```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

Print classification report

In [72]:

```
accuracy_score(y_test,y_predict3)
```

Out[72]:

```
1.0
```

In [73]:

```
print(classification_report(y_test,y_predict3))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

Step 7 [Build VotingClassifier]

Build VotingClassifier model with GradientBoostingClassifier and LogisticRegressionCV that you created in the previous steps

In [74]:

```
model5=VotingClassifier(estimators=[('lr',model4),('gbc',model1)], voting='hard')
```

Perform fit and predict operations

In [75]:

```
model5.fit(X_train,y_train)
```

Out[75]:

```
VotingClassifier(estimators=[('lr', LogisticRegressionCV(Cs=5, cv=4)),
                             ('gbc',
                              AdaBoostClassifier(base_estimator=DecisionTree
Classifier(max_features=4),
                                                 random_state=0))],
                 voting='soft')
```

In [76]:

```
y_predict4=model5.predict(X_test)
y_predict4
```

Out[76]:

```
array([0, 1, 1, 1, 3, 0, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 1, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 1, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 0, 3, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       1, 1, 0, 3, 3, 3, 3, 3, 3, 0, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 0, 1, 0, 1, 0, 0, 0, 3, 0, 0, 1, 3, 0, 0,
       1, 1, 1, 3, 0, 0, 1, 1, 3, 1, 1, 0, 3, 1, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 0, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 0, 1, 3, 0, 0, 1, 1, 0, 1, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 0, 0, 0, 3, 0, 1, 0, 0, 1, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

Print classification report

In [77]:

```
print(accuracy_score(y_test,y_predict4))
```

Out[77]:

```
0.93
```

In [78]:

```
print(classification_report(y_test,y_predict4))
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.90       100
           1       0.91      0.88      0.89       100
           3       1.00      1.00      1.00       100

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300
```

Step8. [Interpret your results]

GradientBoostingClassifier(n_estimators=50)

GradientBoostingClassifier(n_estimators=50,learning_rate=1.0,max_depth=1,random_state=32)

In [79]:

```
model6 = GradientBoostingClassifier(n_estimators=50,learning_rate=1.0,max_depth=1,random_st
```

In [80]:

```
model6.fit(X_train,y_train)
```

Out[80]:

```
GradientBoostingClassifier(learning_rate=1.0, max_depth=1, n_estimators=50,
                           random_state=32)
```

In [81]:

```
y_predict6=model6.predict(X_test)
y_predict6
```

Out[81]:

```
array([0, 1, 1, 1, 3, 1, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 0, 0, 0, 1, 1, 0, 3, 3, 0, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 0, 3, 1, 3, 0, 0, 3, 1, 3, 1, 0, 1, 0,
       1, 0, 0, 0, 1, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       0, 1, 0, 3, 3, 3, 3, 3, 3, 1, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 3, 0, 0,
       0, 0, 1, 3, 1, 0, 1, 1, 3, 1, 1, 0, 3, 0, 1, 3, 0, 3, 0, 1, 3, 0,
       1, 1, 3, 1, 0, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 1, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 1, 1, 3, 0, 1, 1, 1, 0, 0, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 1, 0, 0, 3, 0, 1, 0, 0, 0, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [82]:

```
print(accuracy_score(y_test,y_predict6))
```

Out[82]:

```
1.0
```

In [83]:

```
print(classification_report(y_test,y_predict6))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100
           3       1.00      1.00      1.00       100

    accuracy                           1.00       300
   macro avg       1.00      1.00      1.00       300
weighted avg       1.00      1.00      1.00       300
```

AdaBoostClassifier

AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.01, n_estimators=75, random_state=0)

In [84]:

```
ADBC = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.01,n_est
```

In [85]:

```
model7 = GridSearchCV(ADBC,param_grid,cv=5,n_jobs=-1)
```

In [86]:

```
model7.fit(X_train,y_train)
```

Out[86]:

```
GridSearchCV(cv=5,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassif
ier(max_features=4),
                                          learning_rate=0.01, n_estimators=7
5,
                                          random_state=0),
             n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.001],
                         'n_estimators': [100, 150, 200]})
```

In [87]:

```
y_predict7=model7.predict(X_test)
y_predict7
```

Out[87]:

```
array([0, 1, 1, 1, 3, 0, 0, 0, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 0, 3, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 3, 0, 3, 1, 1, 3, 0, 0, 3, 1, 1, 1, 0, 0,
       3, 0, 3, 3, 3, 1, 1, 0, 0, 1, 1, 0, 3, 3, 1, 3, 1, 1, 0, 1, 0, 3,
       3, 0, 3, 0, 0, 3, 0, 0, 1, 3, 3, 0, 3, 0, 3, 3, 1, 3, 1, 1, 1, 3,
       0, 3, 1, 3, 1, 1, 1, 3, 1, 1, 3, 1, 3, 0, 0, 3, 0, 3, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 3, 0, 0, 0, 0, 0, 1, 3, 3, 1, 1, 1, 1, 3,
       1, 1, 0, 3, 3, 3, 3, 3, 3, 0, 0, 3, 1, 1, 3, 3, 3, 0, 1, 1, 0, 0,
       0, 3, 0, 1, 3, 3, 1, 1, 0, 1, 0, 1, 1, 0, 0, 3, 0, 0, 1, 3, 0, 0,
       1, 1, 1, 3, 0, 0, 1, 1, 3, 1, 1, 0, 3, 1, 1, 3, 0, 3, 1, 1, 3, 0,
       1, 1, 3, 1, 1, 0, 0, 3, 3, 1, 1, 3, 3, 3, 3, 1, 0, 0, 0, 0, 3, 0,
       3, 3, 0, 0, 3, 1, 3, 0, 3, 0, 1, 3, 0, 0, 1, 1, 0, 1, 3, 1, 0, 3,
       3, 0, 3, 1, 0, 1, 1, 3, 1, 3, 3, 0, 0, 1, 3, 0, 0, 3, 0, 1, 0, 1,
       3, 1, 0, 0, 0, 3, 0, 1, 0, 0, 1, 1, 3, 0, 0, 1, 3, 3, 1, 3, 1, 3,
       3, 3, 1, 3, 3, 1, 1, 1, 0, 0, 1, 1, 0, 3], dtype=int64)
```

In [88]:

```
print(accuracy_score(y_test,y_predict7))
```

Out[88]:

```
0.9133333333333333
```

In [89]:

```
print(classification_report(y_test,y_predict7))
```

```
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       100
           1       0.86      0.88      0.87       100
           3       1.00      1.00      1.00       100

    accuracy                           0.91       300
   macro avg       0.91      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300
```