Lab10. Patients Physical Activities Prediction using Boosting

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score,classification_report
from sklearn.ensemble import GradientBoostingClassifier,AdaBoostClassifier,Randor
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

Step 1 [Understand Data]

In [2]:
```
original = pd.read_csv("Human_Activity_Data.csv")
```

In [3]:
```
original.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	tE
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]:
```
original.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),gravityMean)',
       'angle(tBodyGyroMean,gravityMean)',
       'angle(tBodyGyroJerkMean,gravityMean)', 'angle(X,gravityMean)',
       'angle(Y,gravityMean)', 'angle(Z,gravityMean)', 'Activity'],
      dtype='object', length=562)
```

In [5]:
```
original.shape
```

Out[5]:
```
(10299, 562)
```

In [6]:
```
original.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]:
```
original.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]:
```
original.value_counts()
```

Out[8]:
```
tBodyAcc-mean()-X  tBodyAcc-mean()-Y  tBodyAcc-mean()-Z  tBodyAcc-std()-X  tB
odyAcc-std()-Y  tBodyAcc-std()-Z  tBodyAcc-mad()-X  tBodyAcc-mad()-Y  tBodyAc
c-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-energy()-Z  tBodyAcc-iqr()-X  tBodyAcc-iqr()-Y
tBodyAcc-iqr()-Z  tBodyAcc-entropy()-X  tBodyAcc-entropy()-Y  tBodyAcc-entrop
y()-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  tBody
Acc-arCoeff()-Y,3  tBodyAcc-arCoeff()-Y,4  tBodyAcc-arCoeff()-Z,1  tBodyAcc-a
rCoeff()-Z,2  tBodyAcc-arCoeff()-Z,3  tBodyAcc-arCoeff()-Z,4  tBodyAcc-correl
ation()-X,Y  tBodyAcc-correlation()-X,Z  tBodyAcc-correlation()-Y,Z  tGravity
Acc-mean()-X  tGravityAcc-mean()-Y  tGravityAcc-mean()-Z  tGravityAcc-std()-X
tGravityAcc-std()-Y  tGravityAcc-std()-Z  tGravityAcc-mad()-X  tGravityAcc-ma
d()-Y  tGravityAcc-mad()-Z  tGravityAcc-max()-X  tGravityAcc-max()-Y  tGravit
yAcc-max()-Z  tGravityAcc-min()-X  tGravityAcc-min()-Y  tGravityAcc-min()-Z
tGravityAcc-sma()  tGravityAcc-energy()-X  tGravityAcc-energy()-Y  tGravityAc
c-energy()-Z  tGravityAcc-iqr()-X  tGravityAcc-iqr()-Y  tGravityAcc-iqr()-Z
tGravityAcc-entropy()-X  tGravityAcc-entropy()-Y  tGravityAcc-entropy()-Z  tG
ravityAcc-arCoeff()-X,1  tGravityAcc-arCoeff()-X,2  tGravityAcc-arCoeff()-X,3
```

In [9]:
```
label_encoder = LabelEncoder()
original["label_Activity"] = label_encoder.fit_transform(original["Activity"])
```

Step2. [Build a small dataset]

In [10]:
```
original.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]:
```
original.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]:
```
tem1 = original[original['Activity']=='LAYING'][:500]
tem2 = original[original['Activity']=='SITTING'][:500]
tem3 = original[original['Activity']=='WALKING'][:500]
tem4 = original[original['Activity']=='STANDING'][:500]
tem5 = original[original['Activity']=='WALKING_UPSTAIRS'][:500]
tem6 = original[original['Activity']=='WALKING_DOWNSTAIRS'][:500]
```

In [13]:
```
new_df = pd.concat([tem1,tem2,tem3,tem4,tem5,tem6])
```

In [14]:
```
new_df.to_csv("human_activity_clipped3000.csv")
```

In [15]:
```
new_df = pd.read_csv("human_activity_clipped3000.csv")
```

In [16]:
```
new_df.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	tB
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]:
```
new_df.shape
```

Out[17]:
```
(3000, 564)
```

In [18]:
```
new_df.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]:
```
new_df.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]:
```
new_df.value_counts()
```

Out[20]:
```
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  tBodyAcc-max()-X  tBodyAcc-max()-Y  tBodyAcc-max()-Z
tBodyAcc-min()-X  tBodyAcc-min()-Y  tBodyAcc-min()-Z  tBodyAcc-sma()  tBodyAc
c-energy()-X  tBodyAcc-energy()-Y  tBodyAcc-energy()-Z  tBodyAcc-iqr()-X  tBo
dyAcc-iqr()-Y  tBodyAcc-iqr()-Z  tBodyAcc-entropy()-X  tBodyAcc-entropy()-Y
tBodyAcc-entropy()-Z  tBodyAcc-arCoeff()-X,1  tBodyAcc-arCoeff()-X,2  tBodyAc
c-arCoeff()-X,3  tBodyAcc-arCoeff()-X,4  tBodyAcc-arCoeff()-Y,1  tBodyAcc-arC
oeff()-Y,2  tBodyAcc-arCoeff()-Y,3  tBodyAcc-arCoeff()-Y,4  tBodyAcc-arCoeff
()-Z,1  tBodyAcc-arCoeff()-Z,2  tBodyAcc-arCoeff()-Z,3  tBodyAcc-arCoeff()-Z,
4  tBodyAcc-correlation()-X,Y  tBodyAcc-correlation()-X,Z  tBodyAcc-correlati
on()-Y,Z  tGravityAcc-mean()-X  tGravityAcc-mean()-Y  tGravityAcc-mean()-Z  t
GravityAcc-std()-X  tGravityAcc-std()-Y  tGravityAcc-std()-Z  tGravityAcc-mad
()-X  tGravityAcc-mad()-Y  tGravityAcc-mad()-Z  tGravityAcc-max()-X  tGravity
Acc-max()-Y  tGravityAcc-max()-Z  tGravityAcc-min()-X  tGravityAcc-min()-Y  t
GravityAcc-min()-Z  tGravityAcc-sma()  tGravityAcc-energy()-X  tGravityAcc-en
ergy()-Y  tGravityAcc-energy()-Z  tGravityAcc-iqr()-X  tGravityAcc-iqr()-Y  t
GravityAcc-iqr()-Z  tGravityAcc-entropy()-X  tGravityAcc-entropy()-Y  tGravit
yAcc-entropy()-Z  tGravityAcc-arCoeff()-X,1  tGravityAcc-arCoeff()-X,2  tGrav
```

In [21]:
```
X_=new_df.drop(['Activity','label_Activity'],axis=1)
y_=new_df.label_Activity
X__train,X__test,y__train,y__test = train_test_split(X_,y_,test_size=0.2,random_s
```

Build GradientBoostingClassifier for 3000 samples

In [22]:
```
model_ = GradientBoostingClassifier(subsample=0.5,n_estimators=100,learning_rate=
model_.fit(X__train,y__train)
y__pred=model_.predict(X__test)
```

In [23]:
```
print(accuracy_score(y__test,y__pred))
print(classification_report(y__test,y__pred))
```

```
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]:
```
base = DecisionTreeClassifier(max_features=4)
base2 = AdaBoostClassifier(base_estimator=base,random_state=0)
param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

In [25]:
```
model2_ = GridSearchCV(base2,param_grid,cv=10,n_jobs=-1)
model2_.fit(X__train,y__train)
y__pred2=model2_.predict(X__test)
```

In [26]:
```
print(accuracy_score(y__test,y__pred2))
print(classification_report(y__test,y__pred2))
```

```
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]:
```
model2_.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]:
```
model3_ = LogisticRegressionCV(cv=4,Cs=5,penalty='l2')
model3_.fit(X__train,y__train)
y__pred3=model3_.predict(X__test)
```

In [29]:
```
print(accuracy_score(y__test,y__pred3))
print(classification_report(y__test,y__pred3))
```

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

In [31]:
```
all_scores = cross_val_score(estimator=model_,X=X__train,y=y__train,cv=5)
print(all_scores)
```

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

In [32]:
```
model4_ = GridSearchCV(estimator=model_,param_grid=Param_grid,cv=5,n_jobs=-1)
model4_.fit(X__train,y__train)
y__pred4=model4_.predict(X__test)
```

In [33]:
```
print(accuracy_score(y__test,y__pred4))
print(classification_report(y__test,y__pred4))
```

```
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]:
```
model4_.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]:
```
model5_=VotingClassifier(estimators=[('lr',model3_),('gbc',base2)],voting='soft')
model5_.fit(X__train,y__train)
y__pred5=model5_.predict(X__test)
```

In [36]:
```
print(accuracy_score(y__test,y__pred5))
print(classification_report(y__test,y__pred5))
```

```
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]:
```
temp1 = new_df[new_df['Activity']=='LAYING'][:500]
temp2 = new_df[new_df['Activity']=='SITTING'][:500]
temp3 = new_df[new_df['Activity']=='WALKING'][:500]
```

In [38]:
```
df = pd.concat([temp1,temp2,temp3])
```

In [39]:
```
df.to_csv("human_activity_clipped1500.csv")
```

In [40]:
```
df = pd.read_csv("human_activity_clipped1500.csv")
```

In [41]:
```
df.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	tB
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]:
```
df.shape
```

Out[42]:
```
(1500, 565)
```

In [43]:
```
df.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]:
```
df.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]:
```
df.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]:
```
df.value_counts()
```

Out[46]:
```
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  tBodyAcc-mad()-Z  tBodyAcc-max()-X  tBodyAcc-max()-Y  tBody
Acc-max()-Z  tBodyAcc-min()-X  tBodyAcc-min()-Y  tBodyAcc-min()-Z  tBodyAcc-s
ma()  tBodyAcc-energy()-X  tBodyAcc-energy()-Y  tBodyAcc-energy()-Z  tBodyAcc
-iqr()-X  tBodyAcc-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
tBodyAcc-arCoeff()-Y,2  tBodyAcc-arCoeff()-Y,3  tBodyAcc-arCoeff()-Y,4  tBody
Acc-arCoeff()-Z,1  tBodyAcc-arCoeff()-Z,2  tBodyAcc-arCoeff()-Z,3  tBodyAcc-a
rCoeff()-Z,4  tBodyAcc-correlation()-X,Y  tBodyAcc-correlation()-X,Z  tBodyAc
c-correlation()-Y,Z  tGravityAcc-mean()-X  tGravityAcc-mean()-Y  tGravityAcc-
mean()-Z  tGravityAcc-std()-X  tGravityAcc-std()-Y  tGravityAcc-std()-Z  tGra
vityAcc-mad()-X  tGravityAcc-mad()-Y  tGravityAcc-mad()-Z  tGravityAcc-max()-
X  tGravityAcc-max()-Y  tGravityAcc-max()-Z  tGravityAcc-min()-X  tGravityAcc
-min()-Y  tGravityAcc-min()-Z  tGravityAcc-sma()  tGravityAcc-energy()-X  tGr
avityAcc-energy()-Y  tGravityAcc-energy()-Z  tGravityAcc-iqr()-X  tGravityAcc
-iqr()-Y  tGravityAcc-iqr()-Z  tGravityAcc-entropy()-X  tGravityAcc-entropy()
-Y  tGravityAcc-entropy()-Z  tGravityAcc-arCoeff()-X,1  tGravityAcc-arCoeff()
```

Step3. [Build GradientBoostingClassifier]

In [47]:
```
X=df.drop(['Activity','label_Activity'],axis=1)
y=df.label_Activity
```

In [48]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2
```

In [49]:
```
model = GradientBoostingClassifier(subsample=0.5,n_estimators=100,learning_rate=1
```

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_pred=model.predict(X_test)
y_pred
```

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)
```

In [52]:
```
accuracy_score(y_test,y_pred)
```

Out[52]:
```
1.0
```

In [53]:
```
print(classification_report(y_test,y_pred))
```

```
              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]

In [54]:
```
Param_grid={'n_estimators':[50, 100, 200, 400],'learning_rate':[0.1, 0.01]}
```

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)
```

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)
```

In [59]:
```
accuracy_score(y_test,y_pred2)
```

Out[59]:
```
1.0
```

In [60]:
```
print(classification_report(y_test,y_pred2))
```

```
              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
```

In [61]:
```
model2.best_estimator_
```

Out[61]:
```
GradientBoostingClassifier(max_depth=1, max_features=4, n_estimators=200,
                           random_state=42, subsample=0.5)
```

Step5. [Build AdaBoostClassifier]

In [62]:
```
base = DecisionTreeClassifier(max_features=4)
base2 = AdaBoostClassifier(base_estimator=base,random_state=0)
param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

In [63]:
```
model3 = GridSearchCV(base2,param_grid,cv=10,n_jobs=-1)
```

In [64]:
```
model3.fit(X_train,y_train)
```

Out[64]:
```
GridSearchCV(cv=10,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier
(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_pred3=model3.predict(X_test)
y_pred3
```

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)
```

In [66]:
```
accuracy_score(y_test,y_pred3)
```

Out[66]:
```
0.9133333333333333
```

In [67]:
```
print(classification_report(y_test,y_pred3))
```

```
              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
```

In [68]:
```
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]

In [69]:
```
model4 = LogisticRegressionCV(cv=4,Cs=5,penalty='l2')
```

In [70]:
```
model4.fit(X_train,y_train)
```

Out[70]:
```
LogisticRegressionCV(Cs=5, cv=4)
```

In [71]:
```
y_pred4=model4.predict(X_test)
y_pred4
```

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)
```

In [72]:
```
accuracy_score(y_test,y_pred4)
```

Out[72]:
```
1.0
```

In [73]:
```
print(classification_report(y_test,y_pred4))
```

```
              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]

In [74]:
```
model5=VotingClassifier(estimators=[('lr',model4),('gbc',base2)],voting='soft')
```

In [75]:
```
model5.fit(X_train,y_train)
```

Out[75]:
```
VotingClassifier(estimators=[('lr', LogisticRegressionCV(Cs=5, cv=4)),
                             ('gbc',
                              AdaBoostClassifier(base_estimator=DecisionTreeCla
ssifier(max_features=4),
                                                 random_state=0))],
                 voting='soft')
```

In [76]:
```
y_pred5=model5.predict(X_test)
y_pred5
```

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)
```

In [77]:
```
accuracy_score(y_test,y_pred5)
```

Out[77]:
```
0.93
```

In [78]:
```
print(classification_report(y_test,y_pred5))
```

```
              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=3

In [79]:
```
model6 = GradientBoostingClassifier(n_estimators=50,learning_rate=1.0,max_depth=1
```

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_pred6=model6.predict(X_test)
y_pred6
```

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]:
```
accuracy_score(y_test,y_pred6)
```

Out[82]:
```
1.0
```

In [83]:
```
print(classification_report(y_test,y_pred6))
```

```
              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]:
```
base3 = AdaBoostClassifier(base_estimator=base,learning_rate=0.01,n_estimators=75
```

In [85]:
```
model7 = GridSearchCV(base3,param_grid,cv=5,n_jobs=-1)
```

In [86]:
```
model7.fit(X_train,y_train)
```

Out[86]:
```
GridSearchCV(cv=5,
             estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier
(max_features=4),
                                          learning_rate=0.01, n_estimators=75,
                                          random_state=0),
             n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.001],
                         'n_estimators': [100, 150, 200]})
```

In [87]:
```
y_pred7=model7.predict(X_test)
y_pred7
```

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]:
```
accuracy_score(y_test,y_pred7)
```

Out[88]:
```
0.9133333333333333
```

In [89]:
```
print(classification_report(y_test,y_pred7))
```

```
              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
```