

Lab 10: Patients Physical Activities prediction using Boosting

Name : ARUL KUMAR ARK (225229101)

Step 1: Understand Data

```
In [6]: 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_
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.model_selection import cross_val_score
```

```
In [7]: df = pd.read_csv("Human_Activity_Data.csv")
```

```
In [8]: df.head()
```

```
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
```

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 [9]:
```

```
Out[9]: df.shape
```

```
: (151, 562)
```

```
In [10]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 151 entries, 0 to 150
Columns: 562 entries, tBodyAcc-mean()-X to Activity
dtypes: float64(561), object(1)
memory usage: 663.1+ KB
```

```
In [11]: df.columns
```

```
Out[11]: 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 [12]: type(df)
```

```
Out[12]: pandas.core.frame.DataFrame
```

```
In [13]: df['Activity'].value_counts
```

```
Out[13]: <bound method IndexOpsMixin.value_counts of 0           STANDING
          1           STANDING
          2           STANDING
          3           STANDING
          4           STANDING
          ...
          146      WALKING_DOWNSTAIRS
          147      WALKING_DOWNSTAIRS
          148      WALKING_DOWNSTAIRS
          149      WALKING_DOWNSTAIRS
Name: Activity, Length: 151, dtype: object>
```

Step 2: Build a small dataset

```
In [14]: lay = df.loc[df['Activity'] == "LAYING"][:500]
sit = df.loc[df['Activity'] == "SITTING"][:500]
walk = df.loc[df['Activity'] == "WALKING"][:500]
frames = [lay, sit, walk]
df_new = pd.concat(frames)
```

```
In [15]: df_new.shape
```

```
Out[15]: (98, 562)
```

```
In [16]: df_new.to_csv("Human_Activity_sample.csv")
```

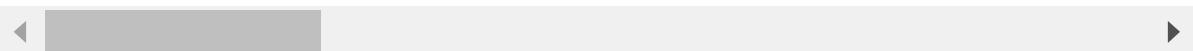
```
In [17]: df1=pd.read_csv('Human_Activity_sample.csv')
```

```
In [18]: df1.head()
```

```
Out[18]:
```

	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 × 563 columns



```
In [19]: df1.shape
```

```
Out[19]: (98, 563)
```

```
In [20]: df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98 entries, 0 to 97
Columns: 563 entries, Unnamed: 0 to Activity
dtypes: float64(561), int64(1), object(1)
memory usage: 431.2+ KB
```

```
In [21]: df1.columns
```

```
Out[21]: 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-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=563)
```

```
In [22]: type(df1)
```

```
Out[22]: pandas.core.frame.DataFrame
```

```
In [23]: df1["Activity"].value_counts()
```

```
Out[23]: WALKING      47  
LAYING        27  
SITTING       24  
Name: Activity, dtype: int64
```

Step 3: Build GradientBoostingClassifier

```
In [24]: X=df1.drop('Activity',axis=1)  
y=df1.Activity
```

```
In [25]: X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_stan
```

```
In [26]: model = GradientBoostingClassifier(n_estimators=100,learning_rate=1.0,max_depth=1,random_state=42)  
model.fit(X_train,y_train)
```

```
Out[26]: GradientBoostingClassifier(learning_rate=1.0, max_depth=1, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with [nbviewer.org](#).

```
In [27]: y_pred=model.predict(X_test)
```

```
In [28]: accuracy_score(y_test,y_pred)
```

```
Out[28]: 0.9666666666666667
```

```
In [29]: print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
LAYING	1.00	0.90	0.95	10
SITTING	0.89	1.00	0.94	8
WALKING	1.00	1.00	1.00	12
accuracy			0.97	30
macro avg	0.96	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

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

```
In [30]: classifier = GradientBoostingClassifier()
```

```
In [31]: all_scores = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=5)
```

```
In [32]: all_scores
```

```
Out[32]: array([1., 1., 1., 1., 1.])
```

To find the average of all the accuracies, simple use the mean() method

```
In [33]: all_scores.mean()
```

```
Out[33]: 1.0
```

```
In [34]: parameter = {'n_estimators': [50, 100, 200, 400], 'learning_rate': [0.1, 0.01]}
```

```
In [35]: model1 = GridSearchCV(estimator=classifier, param_grid=parameter, cv=5, n_jobs=-1)
```

```
In [36]: model1.fit(X_train,y_train)
```

```
Out[36]: GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,  
param_grid={'learning_rate': [0.1, 0.01],  
'n_estimators': [50, 100, 200, 400]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [41]: y_pred2=model1.predict(X_test)
```

```
In [42]: accuracy_score(y_test,y_pred2)
```

```
Out[42]: 1.0
```

```
In [43]: print(classification_report(y_test,y_pred2))
```

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	10
SITTING	1.00	1.00	1.00	8
WALKING	1.00	1.00	1.00	12
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
In [44]: print(model1.best_estimator_)
```

```
GradientBoostingClassifier(n_estimators=50)
```

Step5. [Build AdaBoostClassifier]

```
In [45]: base = DecisionTreeClassifier()
```

```
In [46]: model2 = AdaBoostClassifier(base_estimator=base,random_state=0)
```

```
In [47]: param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

```
In [48]: model3 = GridSearchCV(model2,param_grid,cv=5,n_jobs=-1)
```

```
In [49]: model3.fit(X_train,y_train)
```

```
Out[49]: GridSearchCV(cv=5,
                      estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifi
ier(),
                                         random_state=0),
                     n_jobs=-1,
                     param_grid={'learning_rate': [0.01, 0.001],
                                 'n_estimators': [100, 150, 200]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with [nbviewer.org](#).

```
In [50]: y_pred3=model3.predict(X_test)
```

```
In [51]: accuracy_score(y_test,y_pred3)
```

```
Out[51]: 1.0
```

```
In [52]: print(classification_report(y_test,y_pred3))
```

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	10
SITTING	1.00	1.00	1.00	8
WALKING	1.00	1.00	1.00	12
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
In [53]: : print(model3.best_estimator_)
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.
01,
                   n_estimators=100, random_state=0)
```

Step6. [Build LogisticRegressionCV classifier]

```
In [54]: : model4 = LogisticRegressionCV(cv=4,Cs=5,penalty='l2')
```

```
In [55]: model4.fit(X_train,y_train)
```

Out[55]: LogisticRegressionCV(Cs=5, cv=4)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [56]: y_pred4=model4.predict(X_test)
```

```
In [57]: accuracy_score(y_test,y_pred4)
```

Out[57]: 1.0

```
In [58]: print(classification_report(y_test,y_pred4))
```

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	10
SITTING	1.00	1.00	1.00	8
WALKING	1.00	1.00	1.00	12
accuracy				
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Step 7 [Build VotingClassifier]

```
In [59]: model5=VotingClassifier(estimators=[('lr',model4),('gbc',model1)], voting='hard')
```

```
In [60]: model5.fit(X_train,y_train)
```

```
Out[60]: VotingClassifier(estimators=[('lr', LogisticRegressionCV(Cs=5, cv=4)),
                                         ('gbc',
                                          GridSearchCV(cv=5,
                                                       estimator=GradientBoostingClassif
                                         ier(),
                                                       n_jobs=-1,
                                                       param_grid={'learning_rate': [0.
                                         1,
                                         0.0
                                         1],
                                                       'n_estimators': [50,
                                         100,
                                         200,
                                         40
                                         0]}))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [63]: y_pred5=model5.predict(X_test)
```

```
In [64]: print(classification_report(y_test,y_pred5))
```

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	10
SITTING	1.00	1.00	1.00	8
WALKING	1.00	1.00	1.00	12
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Step8. [Interpret your results]

```
In [65]: print(model1.best_estimator_)
```

```
GradientBoostingClassifier(n_estimators=50)
```

```
In [66]: print(model3.best_estimator_)
```

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

GradientBoostingClassifier

GradientBoostingClassifier(n_estimators=50)

```
In [67]: classifierF = GradientBoostingClassifier(n_estimators=50)  
all_scoresF = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=parameter = {'n_estimators': [50, 100, 200, 400], 'learning_rate': [0.1, 0.01]})
```

```
In [68]: modelGC = GridSearchCV(estimator=classifier, param_grid=parameter, cv=5, n_jobs=-1)
```

```
In [69]: modelGC.fit(X_train, y_train)
```

```
Out[69]: GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,  
param_grid={'learning_rate': [0.1, 0.01],  
'n_estimators': [50, 100, 200, 400]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with [nbviewer.org](#).

```
In [71]: y_predGC = model3.predict(X_test)
```

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

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

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

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	10
SITTING	1.00	1.00	1.00	8
WALKING	1.00	1.00	1.00	12
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
In [74]: ### AdaBoostClassifier
```

```
In [75]: ##### AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.01, n_estimators=100)
```

```
In [77]: modelABC = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), learning_rate=0.01, n_estimators=100)
```

```
In [78]: param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}
```

```
In [79]: modelGSCV = GridSearchCV(modelABC, param_grid, cv=5, n_jobs=-1)
modelGSCV.fit(X_train, y_train)
```

```
Out[79]: GridSearchCV(cv=5,
                     estimator=AdaBoostClassifier(base_estimator=DecisionTreeClassifier(),
                     learning_rate=0.01),
                     n_jobs=-1,
                     param_grid={'learning_rate': [0.01, 0.001],
                     'n_estimators': [100, 150, 200]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [80]: y_predGSCV=model3.predict(X_test)
```

```
In [81]: accuracy_score(y_test,y_predGSCV)
```

```
Out[81]: 1.0
```

```
In [82]: print(classification_report(y_test,y_predGSCV))
```

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	10
SITTING	1.00	1.00	1.00	8
WALKING	1.00	1.00	1.00	12
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

