```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import GradientBoostingClassifier, AdaBoost classifier,
from sklearn.linear_model import LogisticRegressionCV.
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier CV.
from sklearn.preprocessing import LabelEncoder.
```

Step:1

```
Original = pd.read_csv("Human_Activity_Data.csv")
Original.head()
Original.columns
Original.shape
Original.dtypes.
Original.info()
Original.value_counts()
label_encoder = LabelEncoder()
Original["Label_Activity"] = label_encoder.fit_transform
                              (original["Activity"])
```

Step:2

```
original.Activity.value_counts()
Original.label_Activity.value_counts()
```

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

Business Use Case

As you a data scientist, one popular hospital in your city has asked you to design a model based on the mobile phone data of their patients. Your model will help the hospital to understand the physical activity level of their patients. The physical activities are classified into 3 levels - Laying, Sitting and Walking. Based on the prediction, Doctors will recommend health diets, exercise and physical activities to the new patients.

Dataset

You will use a dataset named, ***Human_Activity_Data.csv***.

Step1. [Understand Data]

- Using Pandas, import "***Human_Activity_Data.csv*** " file and print properties such as head, shape, columns, dtype, info and value_counts.

Step2. [Build a small dataset]

- As it is a big dataset, execution time will be long for training and testing. So build a small dataset with only 3 classes, laying, sitting and walking, where each class with 500 samples. So, shape of this new dataframe will be (1500, 563). That is 1500 rows and 563 features.
- Store this new dataframe as a CSV file.

Step3. [Build GradientBoostingClassifier]

- Import your new reduced CSV file
- Print basic properties of the new CSV file
- Split it into training set and test set (30% samples for testing)
- Create GradientBoostingClassifier, fit and predict
- Print accuracy and classification report

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

- Create GridSearchCV model with GradientBoostingClassifier
- Parameters: *param_grid = {'n_estimators': [50, 100, 200, 400], 'learning_rate': [0.1, 0.01]}*
- Perform fit and predict
- Print accuracy, classification report
- Print best parameters such as best no. of trees and learning rate. Use the attribute best_estimator_

Take first 3000 samples:

```
tem1 = original [original ['Activity'] == 'LAYING'] [:500]
tem2 = original [original [original ['Activity'] == 'SITTING'] [:500]
tem3 = original [original ['Activity'] == 'WALKING'] [:500]
tem4 = original [original ['Activity'] == 'STANDING'] [:500]
tem5 = original [original ['Activity'] == 'WALKING_UPSTAIRS'
tem6 = original [original ['Activity'] == 'WALKING_DOWNSTAIRS' [:500] [:500]
new_df = pd.concat ([tem1, tem2, tem3, tem4, tem5 tem6])
new_df.to_csv ("human_activity_clipped 3000.csv")
new_df.head()
new_df.shape
new_df.columns
new_df.dtypes
new_df.value_counts()
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,
```

Build Gradient Boosting:

```
model = GradientBoostingClassifier (subsample=0.5, n_estimators=100,
model.fit (X_train, y_train)
y_pred = model.predict(X_test)
print (accuracy_score (y_test, y_pred))
print (classification_report (y_test, y_pred))
```

Step5. [Build AdaBoostClassifier]

- Create AdaBoostClassifier with DecisionTreeClassifier
- Create GridSearchCV with AdaBoostClassifier model that you created as before
- Parameters: *param_grid = {'n_estimators': [100, 150, 200], 'learning_rate': [0.01, 0.001]}*
- Perform fit, predict
- Print accuracy, classification report
- Print best parameters such as best no. of trees and learning rate. Use the attribute best_estimator_

Step6. [Build LogisticRegressionCV classifier]

- Create a LogisticRegressionCV model with the parameters Cs=5, cv=4, penalty='l2'.
- Perform fit and predict
- Print classification report

Step7. [Build VotingClassifier]

- Build VotingClassifier model with GradientBoostingClassifier and LogisticRegressionCV that you created in the previous steps
- Perform fit and predict operations
- Print classification report

Step8. [Interpret your results]

- Analyze your results
- Change parameters and play with your code

```
base = DecisionTree.classifier(max_features=4)
base2 = AdaBoostClassifier(base_estimator=base, random_state=0)
param_grid = {'n_estimators': [100,150,200], 'learning_rate': [0.01, 0.001]}
model2 = GridSearchCV(base2, param_grid, cv=10, n_jobs=-1)
model2.fit(x_train, y_train)
y_pred2 = model2.predict(x_test)
Print(accuracy_score(y_test, y_pred2))
print(classification_report(y_test, y_pred2))
model2_best_estimator_
Build Logistic Regression
model3 = LogisticRegressionCV(cv=4, Cs=5, penalty='l2')
model3.fit(x_train, y_train)
y_pred3 = model3.predict(x_test)
```

```
Print (accuracy_score(y_test, y_pred3))
print (classification_report (y_test, y_pred3))

Find Best No of trees:

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

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

model_4 = GridSearchCV(estimator = model4, param_grid = param
                       -grid, cv=5, n= jobs=-1),

model4 fit (X_train, y_train)

y_pred4 = model4. predict (x_test)

Point (accuracy_score (y_test, y_pred4))

print (Classification_report (y_test, y_pred4))

model4_best = estimator_

Build Voting classifier for 3000 samples:

model5 = VotingClassifier (estimators = [('lr', model3_),
                                         ('gbc', base 2)], voting = 'soft'
model5 fit (x_train, y_train)

y_pred5 = model5_predict (x_test)

Print (accuracy_score (y_test, y_pred5))

print (classification_report (y_test, y_pred5))

From this 3000 samples you should:

temp1 = new_df [new_df ['Activity'] == 'LAYING'] [:500]
temp2 = new df [new_df ['Activity'] == 'SITTING'] [:500]
```

NOTE

temp3 = raw_df[raw_df['Activity'] == 'WALKING'][:500]

df = pd.concat([temp1, temp2, temp3])

df.to_csv("human_activity_clipped1500.csv")

df = pd.read_csv("human_activity_clipped1500.csv")

df.head()

df.shape

df.columns

df.dtypes

df.info()

df.value_counts()

Step:3 [Build GradientBoostingClassifier]

```
X = df.drop(['Activity', 'label_Activity'], axis=1)
y = df.label_Activity
X_train, X_test, y_train, y_test = train_test_split
                                   (X, y, test_size = 0.2)
model = GradientBoostingClassifier(subsample = 0.5, n_estimators=100,
                                   learning_rate =1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred
```

accuracy_score(y_test, y_pred)

print(classification_report(y_test, y_pred))

step:4

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

all_scores = cross_val_score(estimator=model, X=x_train, y=y_train, cv=5)

print(all_scores)

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

print(all_scores)

model2 = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, n_job=-1)

model2.fit(x_train, y_train)

y_pred2 = model2.predict(x_test)

y_pred2

accuracy_score(y_test, y_pred2)

model2.best_estimator_

step:5

base = DecisionTreeClassifier(max_features=4)

base2 = AdaBoostClassifier(base_estimator=base, random_state=0)

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

model3 = GridSearchCV(base2, param_grid, cv=10, n_jobs=-1)

model3.fit(x_train, y_train)

y_pred3 = model3.predict(x_test)

y_pred3

accuracy_score(y_test, y_pred3)

print(classification_report(y_test, y_pred3))

NOTE

model 3. best_estimator_

Step :6

```
model4 = LogisticRegressionCV (CV=4, CS=5, penalty='l2')
model4.fit (X_train, y_train)
y_pred4 = model4.predict (X_test)
y_pred4
accuracy_score (y_test, y_pred4)
print (classification_report (y_test, y_pred4))
```

Step: 7

```
model5 = VotingClassifier (estimators=[('lr', model4),
                                       ('gbc', base2)], voting='soft')
model5.fit (X_train, y_train)
y_pred5 = model5.predict (X_test)
y_pred5
accuracy: score (y_test, y_pred5)
print (classification_report (y_test, y_pred5))
```

Step:8

GradientBoostingClassifier

```
model6 = GradientBoostingClassifier (n_estimators=50,
                                     learning_rate=1.0,
model6.fit (X_train, y_train)
y_pred6 = model6.predict (X_test)
y_pred6
```

```
accuracy_score(y_test, y_pred6)

print(classification_report(y_test, y_pred6))
```

Ada Boost Classifier:-

```
base3 = AdaBoostClassifier(base_estimator=base,
model7 = GridSearchCV(base3, param_grid, cv=5, n_jobs=-1)
model7.fit(x_train, y_train)
y_pred7 = model7.predict(x_test)
y_pred7.
accuracy_score(y_test, y_pred7)
print(classification_report(y_test, y_pred7))
```