EXERCISE	10. ENSEMBLE LEARNING
DATE	05.11.2024

AIM:
To implement Random Forest and Adaboost using ensemble learning approach.

DESCRIPTION:
Random Forest is a machine learning algorithm that is used for classification, regression, and other tasks. It works by combining multiple decision trees and creating an ensemble of trees.

A simplified explanation of how the algorithm works:

1. Randomly select a subset of the data (sampling with replacement)
2. Build a decision tree based on the selected subset of the data
3. Repeat the process (step 1 and 2) to create multiple decision trees
4. Combine the predictions of all the decision trees to create a final prediction.

Adaptive Boosting (AdaBoost) is a popular ensemble learning algorithm that combines multiple weak classifiers to create a stronger classifier. In AdaBoost, each classifier is trained on the same dataset, but the weights of the data points are adjusted in each iteration to give more weight to the misclassified points.

A simplified explanation of how the algorithm works:

1. Initialize the weights of each data point to 1/N, where N is the total number of data points in the dataset.
2. Train a weak classifier on the dataset.
3. Calculate the error of the classifier by summing the weights of the misclassified points.
4. Calculate the weight of the classifier by using the formula:
weight = ln((1 - error) / error)

5. This formula gives a higher weight to classifiers that have a lower error rate.
6. Update the weights of the data points by multiplying the weights of the misclassified points by e^weight, and the weights of the correctly classified points by e^-weight.
7. Normalize the weights of the data points so that they sum up to 1.
8. Repeat steps 2-6 for a predetermined number of iterations or until the desired accuracy is achieved.
9. Combine the weak classifiers by giving each classifier a weight proportional to its accuracy. The final prediction of the AdaBoost algorithm is obtained by combining the predictions of all the weak classifiers, weighted by their accuracy.

PROGRAM:

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score
from sklearn.preprocessing import StandardScaler

# Load the dataset
data = pd.read_csv("/content/customer_segmentation.csv")  # Replace with the actual path to
your dataset

# Step 1: Data Preprocessing

# 1. Handle missing values (imputation)
# Separate numeric and categorical columns
numeric_columns = data.select_dtypes(include=['float64', 'int64']).columns
categorical_columns = data.select_dtypes(include=['object']).columns

# Impute missing values in numeric columns using the mean strategy
imputer_numeric = SimpleImputer(strategy='mean')
data[numeric_columns] = imputer_numeric.fit_transform(data[numeric_columns])

# Impute missing values in categorical columns using the most_frequent strategy
imputer_categorical = SimpleImputer(strategy='most_frequent')
data[categorical_columns] = imputer_categorical.fit_transform(data[categorical_columns])

# 2. Encoding categorical features to numerical values
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le

# Step 2: Choose independent variables (X) and dependent variable (Y)
X = data.drop(['ID', 'Response'], axis=1)  # Dropping the ID and the target column 'Response'
y = data['Response']  # Target variable 'Response'

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Optional: Standard scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 3: Random Forest Model with different parameters
rf_parameters = [
    {'n_estimators': 50, 'max_depth': 10, 'criterion': 'gini'},
    {'n_estimators': 100, 'max_depth': 15, 'criterion': 'gini'},
    {'n_estimators': 200, 'max_depth': 20, 'criterion': 'entropy'},
    {'n_estimators': 300, 'max_depth': None, 'criterion': 'entropy'}
]

print("Random Forest Results:\n")
for params in rf_parameters:
    rf = RandomForestClassifier(**params, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)

    # Calculate metrics
    cm = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Display results
    print(f"Parameters: {params}")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1 Score: {f1}\n")

# Step 4: Find the best parameters using GridSearchCV for Random Forest
from sklearn.model_selection import GridSearchCV
rf = RandomForestClassifier(random_state=42)
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 15, 20, None],
    'criterion': ['gini', 'entropy']
```

```
}
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=3,
scoring='accuracy')
grid_search_rf.fit(X_train, y_train)

# Best parameters and score
best_rf = grid_search_rf.best_estimator_
y_pred_best_rf = best_rf.predict(X_test)
print("Best Random Forest Parameters:", grid_search_rf.best_params_)
print("Best Random Forest Score:", grid_search_rf.best_score_)

# Step 5: AdaBoost Model with different parameters
ada_parameters = [
    {'n_estimators': 50, 'learning_rate': 0.5},
    {'n_estimators': 100, 'learning_rate': 0.7},
    {'n_estimators': 200, 'learning_rate': 1.0},
    {'n_estimators': 300, 'learning_rate': 1.5}
]

print("\nAdaBoost Results:\n")
for params in ada_parameters:
    ada = AdaBoostClassifier(**params, random_state=42)
    ada.fit(X_train, y_train)
    y_pred = ada.predict(X_test)

    # Calculate metrics
    cm = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Display results
    print(f"Parameters: {params}")
    print(f"Confusion Matrix:\n{cm}")
    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1 Score: {f1}\n")

# Step 6: Find the best parameters using GridSearchCV for AdaBoost
ada = AdaBoostClassifier(random_state=42)
```

```
param_grid_ada = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.5, 0.7, 1.0, 1.5]
}
grid_search_ada = GridSearchCV(estimator=ada, param_grid=param_grid_ada, cv=3,
scoring='accuracy')
grid_search_ada.fit(X_train, y_train)

# Best parameters and score
best_ada = grid_search_ada.best_estimator_
y_pred_best_ada = best_ada.predict(X_test)
print("Best AdaBoost Parameters:", grid_search_ada.best_params_)
print("Best AdaBoost Score:", grid_search_ada.best_score_)
```

```
Random Forest Results:

Parameters: {'n_estimators': 50, 'max_depth': 10, 'criterion': 'gini'}
Confusion Matrix:
[[371   8]
 [ 51  18]]
Accuracy: 0.8683035714285714
Precision: 0.6923076923076923
Recall: 0.2608695652173913
F1 Score: 0.37894736842105264

Parameters: {'n_estimators': 100, 'max_depth': 15, 'criterion': 'gini'}
Confusion Matrix:
[[367  12]
 [ 49  20]]
Accuracy: 0.8638392857142857
Precision: 0.625
Recall: 0.2898550724637681
F1 Score: 0.39603960396039606

Parameters: {'n_estimators': 200, 'max_depth': 20, 'criterion': 'entropy'}
Confusion Matrix:
[[370   9]
 [ 49  20]]
Accuracy: 0.8705357142857143
Precision: 0.6896551724137931
Recall: 0.2898550724637681
F1 Score: 0.40816326530612246
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527:
  warnings.warn(
Parameters: {'n_estimators': 200, 'learning_rate': 1.0}
Confusion Matrix:
[[357  22]
 [ 42  27]]
Accuracy: 0.8571428571428571
Precision: 0.5510204081632653
Recall: 0.391304347826087
F1 Score: 0.4576271186440678

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527:
  warnings.warn(
Parameters: {'n_estimators': 300, 'learning_rate': 1.5}
Confusion Matrix:
[[353  26]
 [ 41  28]]
Accuracy: 0.8504464285714286
Precision: 0.5185185185185185
Recall: 0.4057971014492754
F1 Score: 0.45528455284552843
```

RESULT:

The above code is executed successfully using Ensemble Learning.