Ex. No.8.A

Date:

ENSEMBLING TECHNIQUE

BAGGING

Aim:

To write a Python program that uses Bagging ensembling techniques to analyze the Iris dataset.

Algorithm:

Step 1. Import the required utility modules such as pandas, scikit-learn's train_test_split function and metrics module, xgboost, and the BaggingRegressor class from the ensemble module.
Step 2. Load the Iris dataset using the **load_iris()** function from scikit-learn's datasets module.
Step 3. Extract the target variable from the dataset, which in this case is the "target" attribute.
Step 4. Extract the feature variables from the dataset and store them in a pandas DataFrame. Use the **pd.DataFrame()** constructor to create the DataFrame, passing in the data from the dataset and the column names from the "feature_names" attribute of the dataset.
Step 5. Split the dataset into training and testing sets using the **train_test_split()** function from scikit-learn.
Step 6. Initialize a BaggingRegressor model with XGBoost as the base estimator, by using the **BaggingRegressor()** constructor and passing an instance of **xgb.XGBRegressor()** as the **estimator** parameter.
Step 7. Train the model using the **fit()** method, passing in the training data and target variables.
Step 8. Make predictions on the test set using the **predict()** method.
Step 9. Calculate the mean squared error between the predicted values and the actual target values using the **mean_squared_error()** function from scikit-learn's metrics module.
Step 10. Print out the value of the mean squared error.

Program:

```
# importing utility modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# importing machine learning models for prediction
import xgboost as xgb

# importing bagging module
from sklearn.ensemble import BaggingRegressor

# loading Iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
```

```
# getting target data from the dataset
target = iris.target

# getting train data from the dataset
train = pd.DataFrame(iris.data, columns=iris.feature_names)

# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(
    train, target, test_size=0.20)

# initializing the bagging model using XGBoost as base model with default parameters
model = BaggingRegressor(estimator=xgb.XGBRegressor())

# training model
model.fit(X_train, y_train)

# predicting the output on the test dataset
pred = model.predict(X_test)

# printing the mean squared error between real value and predicted value
print(mean_squared_error(y_test, pred))
```

Viva Questions:

1. What are ensembling techniques, and how do they work?
2. What are some of the benefits of using ensembling techniques, such as Bagging and Boosting?
3. How does Bagging differ from Boosting, and what are the key characteristics of each technique?
4. What is the purpose of the **train_test_split()** function in this code, and why is it important for machine learning?
5. What is the **mean_squared_error()** function, and what does it tell us about the accuracy of a machine learning model?

Result:

Thus the python program to analyze the Iris dataset using Bagging ensembling technique was developed and the output was verified successfully.

Ex. No.8.B **ENSEMBLING TECHNIQUE**

Date: **BOOSTING**

Aim:

To write a Python program that uses Boosting ensembling technique to analyse the Iris dataset.

Algorithm:

Step 1. Import the required utility modules: pandas, load_iris from sklearn.datasets, train_test_split, and accuracy_score from sklearn.metrics

Step 2. Import the required machine learning model: GradientBoostingClassifier from sklearn.ensemble

Step 3. Load the iris dataset using load_iris() function from scikit-learn and assign the data and target to variables **features** and **target**, respectively.

Step 4. Split the dataset into training and validation datasets using train_test_split() function from sklearn.model_selection, and assign the output to variables **X_train**, **X_test**, **y_train**, and **y_test**.

Step 5. Initialize the boosting module with default parameters by creating an instance of GradientBoostingClassifier() and assign it to a variable **model**.

Step 6. Train the model on the training dataset using the **fit()** method of the **model** object.

Step 7. Predict the target variable using the **predict()** method of the **model** object and the test dataset.

Step 8. Evaluate the performance of the model using the **accuracy_score()** function from sklearn.metrics and print the output.

Program:

```
# importing utility modules
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# importing machine learning models for prediction
from sklearn.ensemble import GradientBoostingClassifier

# loading iris dataset
iris = load_iris()

# getting feature data from the iris dataset
features = iris.data

# getting target data from the iris dataset
target = iris.target
```

```
# Splitting between train data into training and validation dataset
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.20)


# initializing the boosting module with default parameters
model = GradientBoostingClassifier()

# training the model on the train dataset
model.fit(X_train, y_train)

# predicting the output on the test dataset
pred_final = model.predict(X_test)

# printing the accuracy score between real value and predicted value
print(accuracy_score(y_test, pred_final))
```

Viva Questions:

1. What is the purpose of the **train_test_split()** function in this code, and how does it work?
2. What is the difference between a regressor and a classifier, and why did we use a classifier in this code?
3. Why did we use the iris dataset in this code, and what are some characteristics of this dataset?
4. What is the purpose of the **accuracy_score()** function, and how does it relate to the performance of the model?
5. What are some advantages and disadvantages of using gradient boosting for machine learning, and how does it compare to other algorithms?

Result:

Thus the python program to analyze the Iris dataset using Boosting ensembling technique was developed and the output was verified successfully.

Ex. No.8.C **ENSEMBLING TECHNIQUE**

Date: **STACKING**

Aim:

To write a Python program that uses Stacking ensembling technique to analyse the Iris dataset.

Algorithm:

Step 1. Import necessary modules including Pandas, scikit-learn's datasets, model_selection, metrics, linear_model, ensemble, and svm.
Step 2. Load iris dataset using the load_iris() method and assign the features and target to variables.
Step 3. Split the dataset into training and testing sets using the train_test_split() method.
Step 4. Initialize the base models with RandomForestClassifier, SVC, and LogisticRegression classes.
Step 5. Create a list of tuples with the initialized base models and pass it along with the final estimator to the StackingClassifier.
Step 6. Fit the StackingClassifier on the training data using the fit() method.
Step 7. Use the predict() method to get the predicted output on the test dataset.
Step 8. Calculate and print the accuracy score using the accuracy_score() method from metrics module.

Program:

```
# importing utility modules
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# importing machine learning models for prediction
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# loading iris dataset
iris = load_iris()

# getting feature data from the iris dataset
features = iris.data

# getting target data from the iris dataset
target = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.20)

# initializing the base models
model1 = RandomForestClassifier(n_estimators=10, random_state=42)
model2 = SVC(kernel='rbf', probability=True, random_state=42)
model3 = LogisticRegression(max_iter=1000, random_state=42)

# initializing the stacking model
estimators = [('rf', model1), ('svc', model2)]
stacking_model = StackingClassifier(estimators=estimators, final_estimator=model3)

# training the stacking model on the train dataset
stacking_model.fit(X_train, y_train)

# predicting the output on the test dataset
pred_final = stacking_model.predict(X_test)

# printing the accuracy score between real value and predicted value
print(accuracy_score(y_test, pred_final))
```

Viva Questions:

1. What is the purpose of importing the pandas and scikit-learn modules in this code?
2. What is the iris dataset, and how is it loaded into the code?
3. How is the data split between training and testing sets in this code?
4. What are the base models used in the stacking classifier, and how are they initialized?
5. How is the final estimator of the stacking classifier determined, and what is its purpose?

Result:

Thus the python program to analyze the Iris dataset using Stacking ensembling technique was developed and the output was verified successfully.