ARTIFICIAL INTELLIGENCE (18CSC305J) LAB

EXPERIMENT 12

Development of ensemble model for an application

Harsh Goel
RA1811003010185
CSE-C1

Aim:

To Develop a model an ensemble model for an application.

Language: Python

Theory:

Ensemble learning helps improve machine learning results bycombining several models. This approach allows the production of betterpredictive performance compared to a single model. Basic idea is to learna set of classifiers (experts) and to allow them to vote.

Problem Formulation and Algorithm:

- Import the required libraries.
- Import the data file in the program.
- Clean the data.
- Find the required features on which the model predicts.
- Split the data into two parts say train_x and train_y
- Train the model using the data train_x and find the mean absolute error with the predicted value and train_y.
- We use random forest and decision trees algorithms to predict our model.
- A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging.

Source Code:

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

melbourne_file_path = './melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
melbourne_data.columns
melbourne_data = melbourne_data.dropna(axis=0)
y = melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude',
'Longtitude']
X = melbourne_data[melbourne_features]
X.describe()
X.head()

# Define model. Specify a number for random_state to ensure same results
 each run
melbourne_model = DecisionTreeRegressor(random_state=1)
# Fit model
melbourne_model.fit(X, y)
print("Making predictions for the following 5 houses:")
print(X.head())
print("The predictions are")
print(melbourne_model.predict(X.head()))
predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)

# split data into training and validation data, for both features and ta
rget
# # The split is based on a random number generator. Supplying a numeric
 value to
# the random_state argument guarantees we get the same split every time
we
# run this script.
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state =0)
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(train_X, train_y)
# get predicted prices on validation data
```

```
val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))

forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

Output Screenshots:

```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

```
1115.7467183128902
```

```
[12] from sklearn.ensemble import RandomForestRegressor
     from sklearn.metrics import mean_absolute_error

     forest_model = RandomForestRegressor(random_state=1)
     forest_model.fit(train_X, train_y)
     melb_preds = forest_model.predict(val_X)
     print(mean_absolute_error(val_y, melb_preds))
```

```
207190.6873773146
```

Verification:

From both the algorithms we can see the difference in the mean absolute error. The predictions made are nearby the data which are input, so we can verify the data is correctly predicted as it lies along the dataset provided.

Result: Hence, successfully implemented the problem and verified the output and document result.