Predicting House Prices Using Machine Learning

phase 4: Model training and Evaluation

Introduction:

In this phase4 we train and test our given dataset US-Housing (https://www.kaggle.com/datasets/vedavyasv/usa-housing)using one machine learning algorithm Random Forest

Random Forest:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Bagging:

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

Model Training:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list files in the input directory

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import explained_variance_score
from sklearn.metrics import confusion_matrix
import os
print(os.listdir("../input"))
import warnings
```

```
warnings.filterwarnings('ignore')

# Any results you write to the current directory are saved as output.
['kc_house_data.csv']

# X(Independent variables) and y(target variables)
X = dataset.iloc[:,1:].values
y = dataset.iloc[:,0].values
```

In [30]:

```
#Splitting the data into train,test data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0
)
```

Multiple Linear Regression:

```
mlr = LinearRegression()
mlr.fit(X_train,y_train)
mlr_score = mlr.score(X_test,y_test)
pred_mlr = mlr.predict(X_test)
expl_mlr = explained_variance_score(pred_mlr,y_test)
```

Decision Tree

```
tr_regressor = DecisionTreeRegressor(random_state=0)
tr_regressor.fit(X_train,y_train)
tr_regressor.score(X_test,y_test)
pred_tr = tr_regressor.predict(X_test)
decision_score=tr_regressor.score(X_test,y_test)
expl_tr = explained_variance_score(pred_tr,y_test)
```

Random Forest Regression Model

```
rf_regressor = RandomForestRegressor(n_estimators=28,random_state=0)
rf_regressor.fit(X_train,y_train)
rf_regressor.score(X_test,y_test)
rf_pred =rf_regressor.predict(X_test)
rf_score=rf_regressor.score(X_test,y_test)
expl_rf = explained_variance_score(rf_pred,y_test)
```

Calculate Model Score

Let's calculate the model score to understand how our model performed along with the explained variance score.

```
print("Multiple Linear Regression Model Score is ",round(mlr.score(X_test,y_test)*100))
print("Decision tree  Regression Model Score is ",round(tr_regressor.score(X_test,y_test)*100))
print("Random Forest Regression Model Score is ",round(rf_regressor.score(X_test,y_test)*100))

#Let's have a tabular pandas data frame, for a clear comparison

models_score =pd.DataFrame({'Model':['Multiple Linear Regression','Decision Tree','Random forest Regression'],'Score':[mlr_score,decision_score,rf_score], 'Explained Variance Score':[expl_mlr,expl_tr,expl_rf]})
models_score.sort_values(by='Score',ascending=False)
```

Output:

Multiple Linear Regression Model Score is 69.0
Decision tree Regression Model Score is 75.0
Random Forest Regression Model Score is 88.0

	Model	Score	Explained Variance Score
2	Random forest Regression	0.88011	0.846248
1	Decision Tree	0.74962	0.730713
0	Multiple Linear Regression	0.68779	0.527528

Evaluation:

- ```
  !pip install -q hvplot
  import pandas as pd
  import numpy as np
  import seaborn as sns
  import matplotlib.pyplot as plt
  import hvplot.pandas
  ```

- ```
  ds=pd.read_csv("/kaggle/input/usa-housing/USA_Housing.csv)
  ds.head()
  ```

Output:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

Dropping of the particular column value*:*

```
df=ds.drop(['Address'],axis=1)
df.head()
```

Output:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05

```
data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
#   Column                          Non-Null Count  Dtype
---  ------                         --------------  -----
0   Avg. Area Income                5000 non-null   float64
1   Avg. Area House Age             5000 non-null   float64
2   Avg. Area Number of Room        5000 non-null   float64
3   Avg. Area Number of Bedrooms    5000 non-null   float64
```

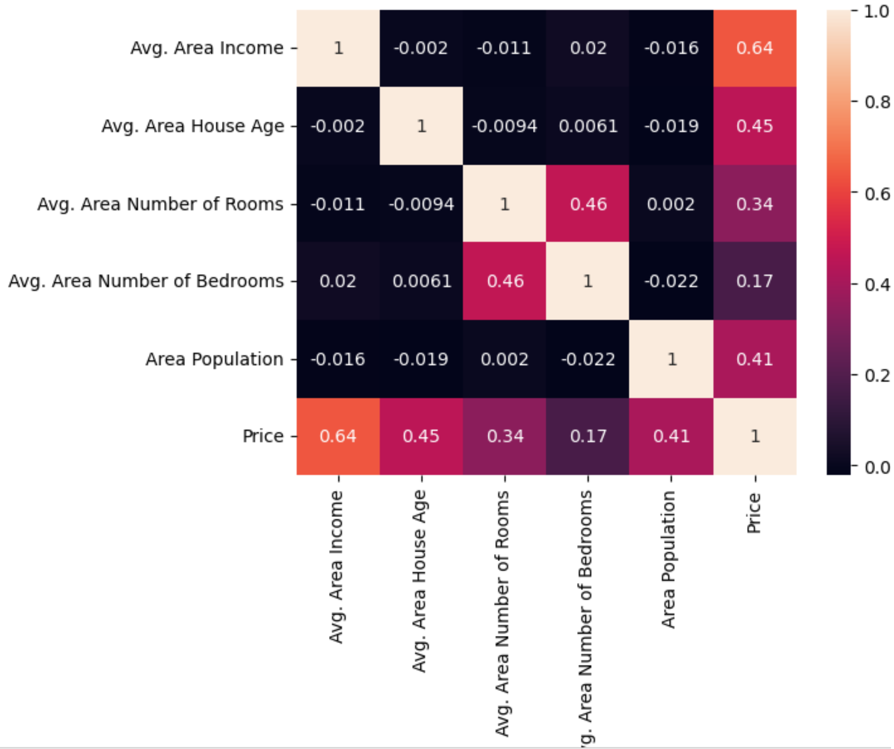
4	Area Population	5000 non-null	float64
5	Price	5000 non-null	float64
6	Address	5000 non-null	object

dtypes: float64(6), object(1)

memory usage: 273.6+ KB

- sns.heatmap(df.corr(),annot=True)

Output:



- x=df.drop(['Price'],axis=1)
  y=df['Price']
  x.column

Output:

Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area Number of Bedrooms', 'Area Population'],dtype='object')

```
➢ from sklearn.model_selection import train_test_split
  x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.3,
random_state=42)
```

```
➢ from sklearn import metrics
  from sklearn.model_selection import cross_val_score
  from sklearn.preprocessing import StandardScaler
  from sklearn.pipeline import Pipeline

  def cross_val(model):
  pred=cross_val_score(model,x,y,cv=10)
  return pred.mean()

  def print_evaluate(true, predicted):
  mae=metrics.mean_absolute_error(true,predicted)
  mse=metrics.mean_squared_error(true,predicted)
  rmse=np.sqrt(metrics.mean_squared_error(true,predicted))
  r2_square=metrics.r2_score(true,predicted)
  print('MAE: ',mae)
  print('MSE: ',mse)
  print('RMSE: ',rmse)
  print('R2 Square',r2_square)
  print('__________________________________________')

  def evaluate(true, predicted):
  mse =metrics.mean_squared_error(true, predicted)
  mae =metrics.mean_absolute_error(true, predicted)
  rmse= np.sqrt(metrics.mean_squared_error(true,predicted))
  r2=square_metrics.r2_score (true, predicted)
  return mae, mse, rmse, r2_square

  pipeline=Pipeline([('std_scalar',StandardScaler())])
  x_train=pipeline.fit_transform(x_train)
  x_test=pipeline.transform(x_test)

#using LinearRegression
```

```
➢ from sklearn.linear_model import LinearRegression
  lin_reg=LinearRegression()
  lin_reg.fit(x_train,y_train)
```

```
test_pred=lin_reg.predict(x_test)
train_pred=lin_reg.predict(x_train)

print('Test set evaluation:\n_____________________________')
print_evaluate(y_test,test_pred)
print('Train set evaluation:\n_____________________________')
print_evaluate(y_train,train_pred)
```

Output:

Test set evaluation:

____

MAE: 81135.56609336878
MSE: 10068422551.40088
RMSE: 100341.52954485436
R2 Square 0.9146818498754016

____

Train set evaluation:

____

MAE: 81480.49973174892
MSE: 10287043161.197224
RMSE: 101425.06180031257
R2 Square 0.9192986579075526

____

#using RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor
rf_reg=RandomForestRegressor(n_estimators=1000)
rf_reg.fit(x_train,y_train)

test_pred=rf_reg.predict(x_test)
train_pred=rf_reg.predict(x_train)

print('Test set evaluation:\n_____________________________')
print_evaluate(y_test,test_pred)
print('Train set evaluation:\n_____________________________')
print_evaluate(y_train,train_pred)
```

Output:

```
Test set evaluation:
_____________________________________
MAE:  94027.38848972939
MSE:  14117343785.892136
RMSE:  118816.42893931856
R2 Square 0.8803719599235804
__________________________________
Train set evaluation:
_____________________________________
MAE:  35362.54528067692
MSE:  1989422953.623089
RMSE:  44602.94781315568
R2 Square 0.9843930758497742
__________________________________
```