Experiment No 5

Title : House price prediction using linear regression model in python.

Theory

In this experiment the Housing dataset which contains information about different houses in Boston is used. We can access this data from the scikit-learn library. There are 506 samples and 13 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.

Python Program

```
# Required Packages:

import numpy as np

import pandas as pd

# Required Dataset:

from sklearn.datasets import load_boston

# Load Dataset:

boston_dataset = load_boston()

print(boston_dataset.keys())

print("Dataset Name and Location : \n")

print(boston_dataset.filename)

# Load Data into Frame:

boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)

print("Dataset Size : ", boston.shape)

print("Initial Instances in boston dataset")

print(boston.head())

print("No of NULL values in each feature")
```

```
print(boston.isnull().sum())

# Set Target Variable 'MEDV':

boston['MEDV'] = boston_dataset.target

correlation_matrix = boston.corr().round(2)

print("Correlation Matrix : ")

print(correlation_matrix)

# Prepare Data to Train Model:

X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])

Y = boston['MEDV']

# Splitting into Training & Testing Set:

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)

print("Train and Test Dataset Size : \n")

print("X_Train Datashape  :",X_train.shape)

print("X_Test Datashape   :",X_test.shape)

print("Y_Train Datashape  :",Y_train.shape)

print("Y_Test Datashape   :",Y_test.shape)

# Training & Testing of Model:

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

lin_model = LinearRegression()

lin_model.fit(X_train, Y_train)

# model evaluation for training set

y_train_predict = lin_model.predict(X_train)
```

```
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))

print("\nThe model performance for training set")

print("--------------------------------------")

print('RMSE is {}'.format(rmse))

print("\n")

# model evaluation for testing set

y_test_predict = lin_model.predict(X_test)

rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

print("The model performance for testing set")

print("--------------------------------------")

print('RMSE is {}'.format(rmse))
```

Output of the program and analysis

Import the required libraries.

```
import numpy as np

import pandas as pd

import seaborn as sns
```

Load the housing data from the scikit-learn library

```
from sklearn.datasets import load_boston

boston_dataset = load_boston()
```

Print the value of the boston_dataset to understand what it contains.

print(boston_dataset.keys()) gives

dict_keys(['data', 'target', 'feature_names', 'DESCR'],' filename')

- data*: contains the information for various houses*
- target*: prices of the house*
- feature_names*: names of the features*
- DESCR*: describes the dataset*
- filename*: gives the filename with location*

Now load the data into a pandas dataframe using pd.DataFrame. We then print the first 5 rows of the data using head()

```
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)

boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98

1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

The target value MEDV is missing from the data. We create a new column of target values and add it to the dataframe.

boston['MEDV'] = boston_dataset.target

Data preprocessing

After loading the data, now we check if there are any missing values in the data. We count the number of missing values for each feature using isnull()

boston.isnull().sum()

CRIM	0
ZN	0
INDUS	0
CHAS	0
NOX	0
RM	0
AGE	0
DIS	0
RAD	0
TAX	0
PTRATIO	0
B	0
LSTAT	0

dtype: int64

there are no missing values in this dataset as shown above.

To understand the relationship of the target variable with other features. we create a correlation matrix that measures the linear relationships between the variables. The correlation matrix can be formed by using the corr function from the pandas dataframe library.

```
correlation_matrix = boston.corr().round(2)

print(correlation_matrix)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	1	-0.2	0.41	-0.06	0.42	-0.22	0.35	-0.38	0.63	0.58	0.29	-0.39	0.46	-0.39
ZN	-0.2	1	-0.53	-0.04	-0.52	0.31	-0.57	0.66	-0.31	-0.31	-0.39	0.18	-0.41	0.36
INDUS	0.41	-0.53	1	0.06	0.76	-0.39	0.64	-0.71	0.6	0.72	0.38	-0.36	0.6	-0.48
CHAS	-0.06	-0.04	0.06	1	0.09	0.09	0.09	-0.1	-0.01	-0.04	-0.12	0.05	-0.05	0.18
NOX	0.42	-0.52	0.76	0.09	1	-0.3	0.73	-0.77	0.61	0.67	0.19	-0.38	0.59	-0.43
RM	-0.22	0.31	-0.39	0.09	-0.3	1	-0.24	0.21	-0.21	-0.29	-0.36	0.13	-0.61	0.7
AGE	0.35	-0.57	0.64	0.09	0.73	-0.24	1	-0.75	0.46	0.51	0.26	-0.27	0.6	-0.38
DIS	-0.38	0.66	-0.71	-0.1	-0.77	0.21	-0.75	1	-0.49	-0.53	-0.23	0.29	-0.5	0.25
RAD	0.63	-0.31	0.6	-0.01	0.61	-0.21	0.46	-0.49	1	0.91	0.46	-0.44	0.49	-0.38
TAX	0.58	-0.31	0.72	-0.04	0.67	-0.29	0.51	-0.53	0.91	1	0.46	-0.44	0.54	-0.47
PTRATIO	0.29	-0.39	0.38	-0.12	0.19	-0.36	0.26	-0.23	0.46	0.46	1	-0.18	0.37	-0.51
B	-0.39	0.18	-0.36	0.05	-0.38	0.13	-0.27	0.29	-0.44	-0.44	-0.18	1	-0.37	0.33
LSTAT	0.46	-0.41	0.6	-0.05	0.59	-0.61	0.6	-0.5	0.49	0.54	0.37	-0.37	1	-0.74
MEDV	-0.39	0.36	-0.48	0.18	-0.43	0.7	-0.38	0.25	-0.38	-0.47	-0.51	0.33	-0.74	1

Observations:

- To fit a linear regression model, we select those features which have a high correlation with our target variable MEDV. By looking at the correlation matrix we can see that RM has a strong positive correlation with MEDV (0.7) where as LSTAT has a high negative correlation with MEDV(-0.74).
- An important point in selecting features for a linear regression model is to check for multi-co-linearity. The features RAD, TAX have a correlation of 0.91. These feature pairs are strongly correlated to each other. We should not select both these features together for training the model. Same goes for the features DIS and AGE which have a correlation of -0.75.

Based on the above observations we select RM and LSTAT as our features.

Now concatenate the LSTAT and RM columns using np.c_ provided by the numpy library.

```
X = pd.DataFrame(np.c_[boston['LSTAT'], boston['RM']], columns = ['LSTAT','RM'])

Y = boston['MEDV']
```

Splitting the data into training and testing sets

Now we split the data into training and testing sets. We train the model with 80% of the samples and test with the remaining 20%. *We do this to assess the model's performance on unseen data*. To split the data we use train_test_split function provided by scikit-learn library. We finally print the sizes of our training and test set to verify if the splitting has occurred properly.

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)

print("X_Train Datashape : ", X_train.shape)

print("X_Test Datashape : ",X_test.shape)

print("Y_Train Datashape: ",Y_train.shape)

print("Y_Test Datashape: ",Y_test.shape)
```

output:

```
X_Train Datashape : (404, 2)

X_Test Datashape : (102, 2)

Y_Train Datashape : (404,)

Y_Test Datashape : (102,)
```

Training and Testing the model

We use scikit-learn's LinearRegression to train our model on both the training and test sets.

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error

lin_model = LinearRegression()

lin_model.fit(X_train, Y_train)
```

Model evaluation :We will evaluate our model using RMSE

```
y_train_predict = lin_model.predict(X_train)

rmse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))

print("The model performance for training set")

print("--------------------------------------")

print('RMSE is {}'.format(rmse))

print("\n")
```

Output:

```
The model performance for training set

--------------------------------------

RMSE is 5.6371293350711955

# model evaluation for testing set

y_test_predict = lin_model.predict(X_test)

rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))

print("The model performance for testing set")

print("--------------------------------------")

print('RMSE is {}'.format(rmse))
```

output:

```
The model performance for testing set

--------------------------------------

RMSE is 5.13740078470291
```