ds-practical-7

October 28, 2024

0.1 DS PRACTICAL 7

0.2 Performing Regression to predict value for target variable using KNN Regression

---

Problem Statement: Predicting House Prices Using K-Nearest Neighbors (KNN) Regression We have to develop a regression model to predict house prices based on several features using the K-Nearest Neighbors (KNN) algorithm. The goal is to create an accurate predictive model that estimates house prices based on key features such as the size of the house, number of bedrooms, distance to the city center, age of the property, and the local crime rate.

0.2.1 Dataset Description:

The dataset contains the following features:

- **Size of the House (sq. ft.):** Continuous feature representing the total living area.
- **Number of Bedrooms:** Integer representing the number of bedrooms.
- **Distance to City Center (miles):** Continuous feature representing the distance from the property to the city center.
- **Age of the Property (years):** Continuous feature representing the age of the house.

---

[20]:
```
import pandas as pd
data = {
    'Rooms': [3, 4, 2, 5, 3, 4, 6, 5, 2, 3, 4, 5, 6, 4, 3],
    'Size_in_SqFt': [1500, 1700, 1200, 2000, 1600, 1750, 2200, 1900, 1100,␣
 ↪1450, 1600, 1800, 2100, 1550, 1300],
    'Distance_to_City': [10, 15, 8, 12, 10, 18, 25, 15, 5, 12, 10, 20, 25, 15,␣
 ↪7],
    'Age_of_House': [10, 5, 20, 15, 8, 12, 4, 7, 30, 12, 6, 10, 5, 15, 20],
    'Bathrooms': [2, 3, 1, 3, 2, 2, 4, 3, 1, 2, 3, 3, 4, 2, 1],
    'Price_in_Thousands': [300, 450, 200, 500, 350, 470, 600, 520, 190, 310,␣
 ↪400, 480, 590, 360, 220]
}
df = pd.DataFrame(data)
df
```

[20]:

	Rooms	Size_in_SqFt	Distance_to_City	Age_of_House	Bathrooms	\
0	3	1500	10	10	2	
1	4	1700	15	5	3	
2	2	1200	8	20	1	
3	5	2000	12	15	3	
4	3	1600	10	8	2	
5	4	1750	18	12	2	
6	6	2200	25	4	4	
7	5	1900	15	7	3	
8	2	1100	5	30	1	
9	3	1450	12	12	2	
10	4	1600	10	6	3	
11	5	1800	20	10	3	
12	6	2100	25	5	4	
13	4	1550	15	15	2	
14	3	1300	7	20	1	

	Price_in_Thousands
0	300
1	450
2	200
3	500
4	350
5	470
6	600
7	520
8	190
9	310
10	400
11	480
12	590
13	360
14	220

0.2.2 Split the dataset into features (X) and target (y)

[21]:
```
X = df.drop('Price_in_Thousands', axis=1)  # Features - Target variable is␣
↪dropped
y = df['Price_in_Thousands']  # Target (Price)
```

[41]:
```
print(X)
```

	Rooms	Size_in_SqFt	Distance_to_City	Age_of_House	Bathrooms
0	3	1500	10	10	2
1	4	1700	15	5	3

```
2       2          1200                8          20          1
3       5          2000               12          15          3
4       3          1600               10           8          2
5       4          1750               18          12          2
6       6          2200               25           4          4
7       5          1900               15           7          3
8       2          1100                5          30          1
9       3          1450               12          12          2
10      4          1600               10           6          3
11      5          1800               20          10          3
12      6          2100               25           5          4
13      4          1550               15          15          2
14      3          1300                7          20          1
```

[42]:
```
print(y)
```

```
0     300
1     450
2     200
3     500
4     350
5     470
6     600
7     520
8     190
9     310
10    400
11    480
12    590
13    360
14    220
Name: Price_in_Thousands, dtype: int64
```

0.2.3 Split the dataset into training and testing sets

[24]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=30)
```

[25]:
```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (11, 5)
```

```
X_test shape: (4, 5)
y_train shape: (11,)
y_test shape: (4,)
```

[26]:
```
print("X_train:\n", X_train)
```

```
X_train:
     Rooms  Size_in_SqFt  Distance_to_City  Age_of_House  Bathrooms
9        3          1450                12            12          2
8        2          1100                 5            30          1
3        5          2000                12            15          3
6        6          2200                25             4          4
1        4          1700                15             5          3
13       4          1550                15            15          2
2        2          1200                 8            20          1
7        5          1900                15             7          3
4        3          1600                10             8          2
14       3          1300                 7            20          1
5        4          1750                18            12          2
```

[27]:
```
print("X_test:\n", X_test)
```

```
X_test:
     Rooms  Size_in_SqFt  Distance_to_City  Age_of_House  Bathrooms
0        3          1500                10            10          2
10       4          1600                10             6          3
11       5          1800                20            10          3
12       6          2100                25             5          4
```

[28]:
```
print("y_train:\n", y_train)
```

```
y_train:
 9     310
8      190
3      500
6      600
1      450
13     360
2      200
7      520
4      350
14     220
5      470
Name: Price_in_Thousands, dtype: int64
```

[29]:
```
print("y_test:\n", y_test)
```

```
y_test:
```

```
 0     300
10    400
11    480
12    590
Name: Price_in_Thousands, dtype: int64
```

0.2.4 Standardize the feature data (since KNN is distance-based)

[30]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[31]:
```
print(X_test_scaled)
```

```
[[-0.59962535 -0.34911298 -0.54198378 -0.46746444 -0.19425717]
 [ 0.22485951 -0.04189356 -0.54198378 -1.00873905  0.87415728]
 [ 1.04934436  0.57254529  1.32108546 -0.46746444  0.87415728]
 [ 1.87382922  1.49420357  2.25262007 -1.14405771  1.94257172]]
```

This line transforms the test data using the mean and standard deviation computed from the training data. It is important to note that the scaler should only be fit on the training data and then applied to both the training and test data to avoid any bias or data leakage. This ensures that the test data is scaled in the same way as the training data, maintaining consistency.

---

[32]:
```
print(X_train_scaled)
```

```
[[-0.59962535 -0.5027227  -0.16936993 -0.19682713 -0.19425717]
 [-1.42411021 -1.57799069 -1.47351839  2.23890863 -1.26267162]
 [ 1.04934436  1.18698414 -0.16936993  0.20912883  0.87415728]
 [ 1.87382922  1.80142299  2.25262007 -1.27937636  1.94257172]
 [ 0.22485951  0.26532587  0.38955084 -1.14405771  0.87415728]
 [ 0.22485951 -0.19550327  0.38955084  0.20912883 -0.19425717]
 [-1.42411021 -1.27077126 -0.91459762  0.8857221  -1.26267162]
 [ 1.04934436  0.87976472  0.38955084 -0.8734204   0.87415728]
 [-0.59962535 -0.04189356 -0.54198378 -0.73810175 -0.19425717]
 [-0.59962535 -0.96355183 -1.10090455  0.8857221  -1.26267162]
 [ 0.22485951  0.41893558  0.94847161 -0.19682713 -0.19425717]]
```

It fits the scaler to the X_train data (calculates the mean and standard deviation for each feature) and then transforms the training data by applying the standardization. After this transformation, each feature will have a mean of 0 and a standard deviation of 1 in the training data.

0.2.5 Apply KNN regression

[33]:
```
from sklearn.neighbors import KNeighborsRegressor

knn_regressor = KNeighborsRegressor(n_neighbors=2,metric='manhattan')  # You␣
 ↪can change the number of neighbors (k)
knn_regressor.fit(X_train_scaled, y_train) # This line trains (fits) the KNN␣
 ↪model using the training data.
```

[33]:
```
KNeighborsRegressor(metric='manhattan', n_neighbors=2)
```

0.2.6 Make predictions

[34]:
```
y_pred = knn_regressor.predict(X_test_scaled)
y_pred
```

[34]:
```
array([330., 400., 495., 560.])
```

0.2.7 Evaluate the model performance using R squared value and MSE

MSE measures the average of the squared differences between actual and predicted values. It's a common metric for regression models, where lower values indicate better model performance.

R-squared value, or coefficient of determination, between the actual and predicted values. R^2 values range from 0 to 1, where: 1 means the model perfectly explains all the variance in the data. 0 means the model explains none of the variance.

[35]:
```
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared value:", r2)
```

```
Mean Squared Error: 506.25
R-squared value: 0.9552733296521259
```

0.2.8 Compare the predicted values to actual values

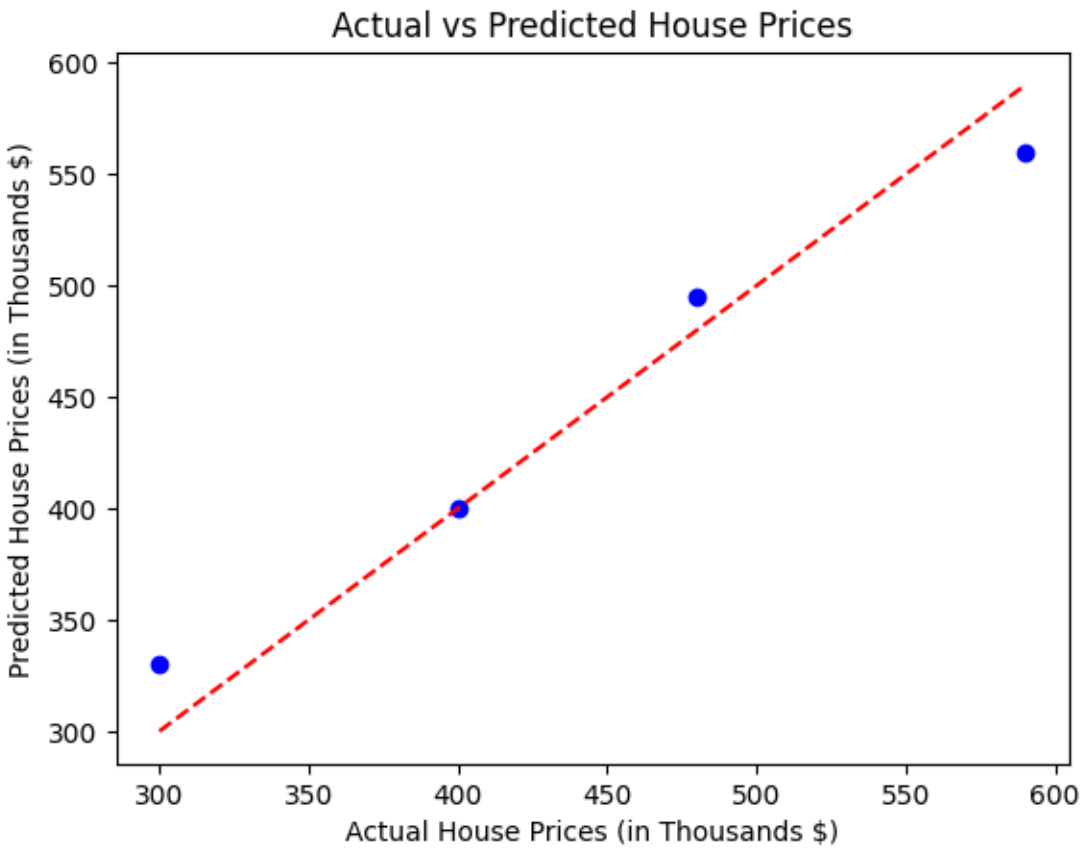
[36]:
```
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(comparison_df)
```

```
    Actual  Predicted
0      300      330.0
10     400      400.0
11     480      495.0
12     590      560.0
```

0.2.9 Visualize the trendline (actual vs predicted prices)

[37]:
```
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',␣
 ↪linestyle='--')
plt.xlabel('Actual House Prices (in Thousands $)')
plt.ylabel('Predicted House Prices (in Thousands $)')
plt.title('Actual vs Predicted House Prices')
plt.show()
```



0.2.10 Predict the price for the given house

[38]:
```
import numpy as np

house = np.array([[3, 1440,6,5,3]])
house_scaled = scaler.fit_transform(house)
price_pred = knn_regressor.predict(house_scaled)
p = price_pred[0].astype(float)
print(f"Predicted Price of the given house is {p} thousand $")
```

```
Predicted Price of the given house is 335.0 thousand $
```