2.One-Hot Encoding Multiple Linear Regression (SK-Learn)

October 28, 2021

One-Hot Encoding for Multiple Linear Regression (SK-Learn)

[40]:
```
# Import necessary package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

0.0.1 Step 1: Load the dataset

[41]:
```
# Load the dataset into pandas dataframe
df = pd.read_csv("E:\\MY LECTURES\\DATA SCIENCE\\3.
↪Programs\\dataset\\house_price.csv")
# Change this location based on the location of dataset in your machine
```

[42]:
```
# Display the first five records
df
```

[42]:

	town	area	price
0	monroe township	2600	550000
1	monroe township	3000	565000
2	monroe township	3200	610000
3	monroe township	3600	680000
4	monroe township	4000	725000
5	west windsor	2600	585000
6	west windsor	2800	615000
7	west windsor	3300	650000
8	west windsor	3600	710000
9	robinsville	2600	575000
10	robinsville	2900	600000
11	robinsville	3100	620000
12	robinsville	3600	695000

[43]:
```
# Dataset shape (number of rows and columns)
df.shape
```

[43]: (13, 3)

0.0.2 Step 2: Apply EDA

You may apply univariate and bivariate analysis

0.0.3 Step 3. Pre-process and extract the features

Unique values in the dataset

[44]:
```
df.nunique()
```

[44]:
```
town      3
area      9
price    13
dtype: int64
```

One hot encoding

[45]:
```
# Do label encoding for feature "town"
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

[46]:
```
dfle = df
le.fit_transform(dfle.town)
```

[46]:
```
array([0, 0, 0, 0, 0, 2, 2, 2, 2, 1, 1, 1, 1])
```

[47]:
```
dfle.town = le.fit_transform(dfle.town)
dfle
```

[47]:

	town	area	price
0	0	2600	550000
1	0	3000	565000
2	0	3200	610000
3	0	3600	680000
4	0	4000	725000
5	2	2600	585000
6	2	2800	615000
7	2	3300	650000
8	2	3600	710000
9	1	2600	575000
10	1	2900	600000
11	1	3100	620000
12	1	3600	695000

[48]:
```
# Load Price feature into Y and remaining features into X
X = dfle.iloc[:,:2].values
Y = dfle.iloc[:,2].values
```

One hot encoder to create dummy variable columns

[49]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
ohe = make_column_transformer((OneHotEncoder(categories='auto'), [0]),␣
 ↪remainder="passthrough")
# [0] indicates based on first column of X dummy variables are created
```

[50]:
```
X = ohe.fit_transform(X)
X
```

[50]:
```
array([[1.0e+00, 0.0e+00, 0.0e+00, 2.6e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 3.0e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 3.2e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 3.6e+03],
       [1.0e+00, 0.0e+00, 0.0e+00, 4.0e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 2.6e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 2.8e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 3.3e+03],
       [0.0e+00, 0.0e+00, 1.0e+00, 3.6e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 2.6e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 2.9e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 3.1e+03],
       [0.0e+00, 1.0e+00, 0.0e+00, 3.6e+03]])
```

[51]:
```
# Drop first column to avoid dummy trap
X = X[:,1:]
X
```

[51]:
```
array([[0.0e+00, 0.0e+00, 2.6e+03],
       [0.0e+00, 0.0e+00, 3.0e+03],
       [0.0e+00, 0.0e+00, 3.2e+03],
       [0.0e+00, 0.0e+00, 3.6e+03],
       [0.0e+00, 0.0e+00, 4.0e+03],
       [0.0e+00, 1.0e+00, 2.6e+03],
       [0.0e+00, 1.0e+00, 2.8e+03],
       [0.0e+00, 1.0e+00, 3.3e+03],
       [0.0e+00, 1.0e+00, 3.6e+03],
       [1.0e+00, 0.0e+00, 2.6e+03],
       [1.0e+00, 0.0e+00, 2.9e+03],
       [1.0e+00, 0.0e+00, 3.1e+03],
       [1.0e+00, 0.0e+00, 3.6e+03]])
```

0.0.4 Step 4. Split the data for training and testing

[52]:
```python
# Splitting dataset into training and 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 = 0)
```

0.0.5 Step 5: Training phase (bulding the model)

[53]:
```python
# Fitting line on two dimension on the training set
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, y_train)
```

[53]:
```
LinearRegression()
```

[54]:
```python
b = model.intercept_
```

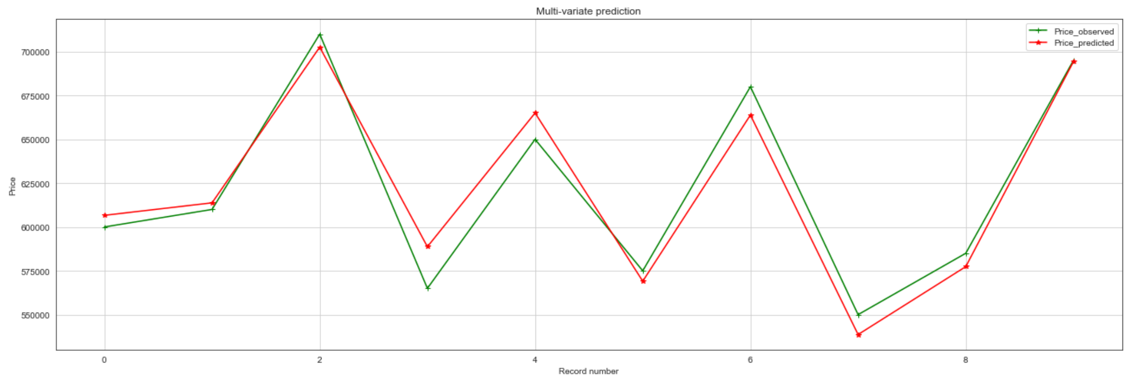
[55]:
```python
coef = model.coef_
```

[56]:
```python
# Predicting the Training set results
y_train_pred = model.predict(x_train)
```

Visualizing the model It involves over 3 dimensions, so imagine yourself.

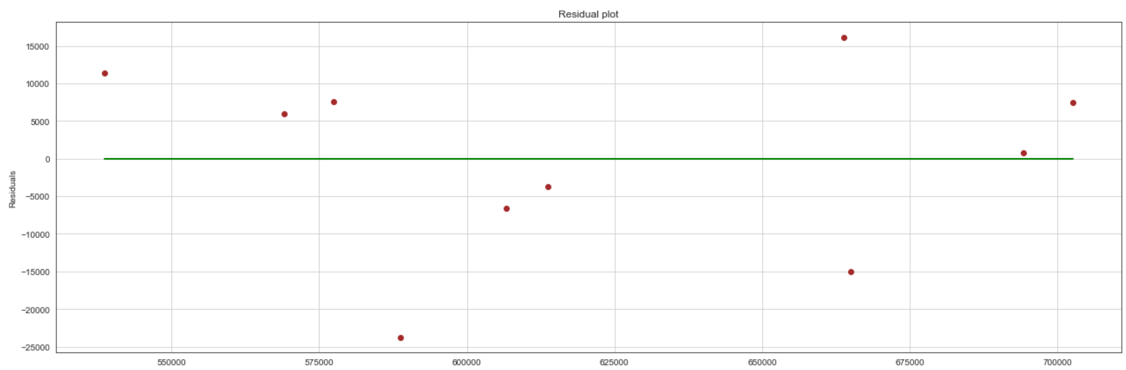
Plotting observed sale (x) and predicted sale (y) for training set

[57]:
```python
# Predicting the Test set results (displaying only for 100 reocrds)
x = np.arange(len(y_train_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_train[:100],"g-+",label="Price_observed")
plt.plot(x,y_train_pred[:100],"r-*",label="Price_predicted")
plt.grid(b=None)
plt.xlabel("Record number")
plt.ylabel("Price")
plt.title("Multi-variate prediction")
plt.legend()
plt.show()
```



Residual (Error) plot If the model has done good predictions, then the datapoints must be near around to horizontal line.

[58]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_train-y_train_pred
zeros = y_train-y_train
plt.scatter(y_train_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_train_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



0.0.6 Different error calculations to asses the model for training set

1. Sum of Squared Error (SSE)

$$SSE(m,b) = \sum_{i=1}^{n}(y_i - \hat{y})^2 = \sum_{i=1}^{n}(y_i - (m * x_i + b))^2 \tag{1}$$

[59]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = y_train[i] - y_train_pred[i]
  squ_diff = diff**2
  sum = sum + squ_diff
Train_SSE = np.round(sum,2)
print("Sum of Squared Error (SSE) :",Train_SSE)
```

```
Sum of Squared Error (SSE) : 1385346045.2
```

2. Mean Squared Error (MSE)

$$MSE(m,b) = \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n} = \frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n} \tag{2}$$

[60]:
```
Train_MSE = np.round(Train_SSE/n,2)
print("Mean Squared Error (MSE) :",Train_MSE)
```

```
Mean Squared Error (MSE) : 138534604.52
```

3. Root Mean Squared Error (RMSE)

$$RMSE(m,b) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n}} \tag{3}$$

[61]:
```
Train_RMSE = np.round(np.sqrt(Train_MSE),2)
print("Root Mean Squared Error (RMSE) :",Train_RMSE)
```

```
Root Mean Squared Error (RMSE) : 11770.07
```

4. Mean Absolute Error (MAE)

$$MAE(m,b) = \frac{\sum_{i=1}^{n}|(y_i - \hat{y})|}{n} \tag{4}$$

[62]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = y_train[i] - y_train_pred[i]
```

```
  sum = sum + np.abs(diff)
Train_MAE = np.round(sum/n,2)
print("Mean Absolute Error (MAE) :",Train_MAE)
```

```
Mean Absolute Error (MAE) : 9833.33
```

5. Mean Absolute Percentage Error (MAPE)

$$MAPE(m,b) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - \hat{y})}{y_i}\right| = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{(y_i - (m * x_i + b))}{y_i}\right| \quad (5)$$

[63]:
```
sum = 0
n = len(x_train)
for i in range (0,n):
  diff = (y_train[i] - y_train_pred[i])/y_train[i]
  sum = sum + np.abs(diff)
Train_MAPE = np.round(sum/n*100,2)
print("Mean Absolute Percentage Error (MAPE) :",Train_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 1.62
```

0.0.7 Calculating R-Squred value (goodness of model) using SSE

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (6)$$

[64]:
```
from sklearn.metrics import r2_score
out = r2_score(y_train,y_train_pred)
Train_RS = np.round(out,2)*100
print("R-Squred value (goodness of model) for training set :",Train_RS,"%")
```

```
R-Squred value (goodness of model) for training set : 95.0 %
```

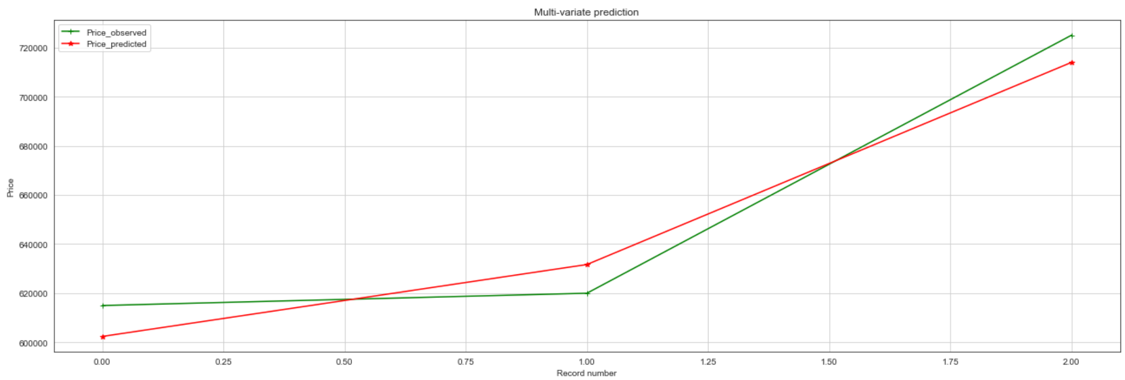
0.0.8 Step 6: Testing phase

[65]:
```
# Predicting values for test input set
y_test_pred = model.predict(x_test)
```

Visualizing the model It involves more than 3 dimensions, so imagine yourself

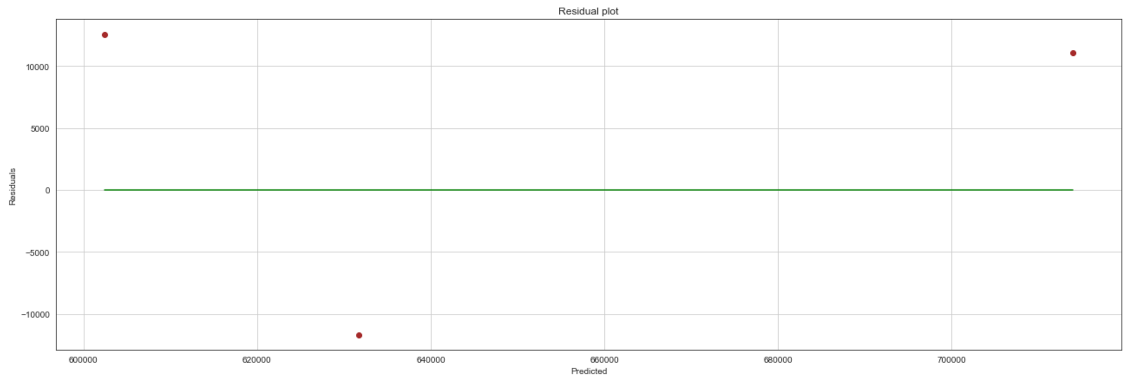
Plotting observed sale (x) and predicted sale (y) for test set

[66]:
```
# Plotting the predicted values
x = np.arange(len(y_test_pred))
fig = plt.figure(figsize=(22,7))
plt.plot(x,y_test,"g-+",label="Price_observed")
plt.plot(x,y_test_pred,"r-*",label="Price_predicted")
plt.grid(b=None)
plt.xlabel("Record number")
plt.ylabel("Price")
plt.title("Multi-variate prediction")
plt.legend()
plt.show()
```



Residual (Error) plot If the model has done good predictions, then the datapoints must be near around to horizontal line.

[67]:
```
sns.set_style(style='white')
fig = plt.figure(figsize=(22,7))
residuals = y_test-y_test_pred
zeros = y_test-y_test
plt.scatter(y_test_pred,residuals,color="brown")
plt.grid(b=None)
plt.plot(y_test_pred,zeros,"g")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual plot")
plt.show()
```



0.0.9 Different error calculations to asses the model for the test set

1. Sum of Squared Error (SSE)

$$SSE(m,b) = \sum_{i=1}^{n}(y_i - \hat{y})^2 = \sum_{i=1}^{n}(y_i - (m * x_i + b))^2 \tag{7}$$

[68]:
```
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = y_test[i] - y_test_pred[i]
  squ_diff = diff**2
  sum = sum + squ_diff
Test_SSE = np.round(sum,2)
print("Sum of Squared Error (SSE) :",Test_SSE)
```

```
Sum of Squared Error (SSE) : 416947606.45
```

2. Mean Squared Error (MSE)

$$MSE(m,b) = \frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n} = \frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n} \tag{8}$$

[69]:
```
Test_MSE = np.round(Train_SSE/n,2)
print("Mean Squared Error (MSE) :",Test_MSE)
```

```
Mean Squared Error (MSE) : 461782015.07
```

3. Root Mean Squared Error (RMSE)

$$RMSE(m,b) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - (m * x_i + b))^2}{n}} \tag{9}$$

[70]:
```python
Test_RMSE = np.round(np.sqrt(Test_MSE),2)
print("Root Mean Squared Error (RMSE) :",Test_RMSE)
```

```
Root Mean Squared Error (RMSE) : 21489.11
```

4. Mean Absolute Error (MAE)

$$MAE(m,b) = \frac{\sum_{i=1}^{n} |(y_i - \hat{y})|}{n} \tag{10}$$

[71]:
```python
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = y_test[i] - y_test_pred[i]
  sum = sum + np.abs(diff)
Test_MAE = np.round(sum/n,2)
print("Mean Absolute Error (MAE) :",Test_MAE)
```

```
Mean Absolute Error (MAE) : 11772.6
```

5. Mean Absolute Percentage Error (MAPE)

$$MAPE(m,b) = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{(y_i - \hat{y})}{y_i} \right| = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{(y_i - (m * x_i + b))}{y_i} \right| \tag{11}$$

[72]:
```python
sum = 0
n = len(x_test)
for i in range (0,n):
  diff = (y_test[i] - y_test_pred[i])/y_test[i]
  sum = sum + np.abs(diff)
Test_MAPE = np.round(sum/n*100,2)
print("Mean Absolute Percentage Error (MAPE) :",Test_MAPE)
```

```
Mean Absolute Percentage Error (MAPE) : 1.82
```

0.0.10 Calculating R-Squred value (goodness of model) using SSE

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{12}$$

[73]:
```python
from sklearn.metrics import r2_score
out = r2_score(y_test,y_test_pred)
Test_RS = np.round(out,2)*100
print("R-Squred value (goodness of model) for testing set :",Test_RS,"%")
```

```
R-Squred value (goodness of model) for testing set : 95.0 %
```

0.0.11 Step 7. Underfitting and overfitting observation

[74]:
```
print("Error \t From training phase       From testing phase ")
print("=========================================================")
print("SSE   \t\t",Train_SSE,"\t\t", Test_SSE)
print("MSE   \t\t",Train_MSE,"\t\t", Test_MSE)
print("RMSE   \t\t",Train_RMSE,"\t\t", Test_RMSE)
print("MAE   \t\t",Train_MAE,"\t\t", Test_MAE)
print("RS   \t\t",Train_RS,"\t\t\t", Test_RS)
```

```
Error    From training phase       From testing phase
===================================================
SSE                 1385346045.2           416947606.45
MSE                 138534604.52           461782015.07
RMSE                11770.07               21489.11
MAE                 9833.33                11772.6
RS                  95.0                   95.0
```

0.0.12 Step 8. Let us predict for future inputs

How much is the house price for 3200 square feet in monroe township? pass input as 3200,0,0 (third feature (monroe township) we dropped is taken as 1).

[75]:
```
model.predict([[3200,0,0]])
```

[75]:
```
array([97591624.29378527])
```

How much is the house price for 2800 square feet in robinsvill? pass input as 2800,1,0 (third feature (monroe township) is taken as 0).

[76]:
```
model.predict([[2800,1,0]])
```

[76]:
```
array([85458043.7853107])
```

How much is the house price for 3400 square feet in west windsor? pass input as 3400,0,1 (third feature (monroe township) we dropped is taken as 0).

[77]:
```
model.predict([[3400,0,1]])
```

[77]:
```
array([1.03677908e+08])
```