Lab Session
Linear Regression

(Implementing Least Square Error Fit and Gradient Descent)

Dr. JASMEET SINGH
ASSISTANT PROFESSOR, CSED
TIET, PATIALA

Multiple Linear Regression- Least Square Error Fit

- A multiple linear regression model with k independent predictor variables $x_1, x_2 \ldots, x_k$ predicts the output variable as:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots \ldots \ldots \ldots .. + \beta_k x_k$$

- The above equation can be represented in matrix form as $y = X\beta$

where y= $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_n \end{bmatrix}$; $\beta = \begin{bmatrix} \beta_0 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix}$ and $X = \begin{bmatrix} 1 & x_{11} & x_{12} \cdots & x_{1k} \\ 1 & x_{21} & x_{22} \cdots & x_{2k} \\ \vdots & \vdots & \vdots \quad \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \cdots & x_{nk} \end{bmatrix}$

- According to least square error method, the mean square error is minimum when

$$\boldsymbol{\hat{\beta}} = (\boldsymbol{X^T X})^{-1} \boldsymbol{X^T y}$$

Least Square Error Fit (LSE)- Step by Step

- Following steps are followed for implementation of least square error fit:

1. Load the dataset.
2. Remove noise, outliers and check for feature selection or extraction (if required).
3. Separate the dataset into X (input/independent variables) and Y (dependent feature). Scale the feature values of X in a fixed range and add a new column (in the beginning) with all values 1.
4. Split the dataset into train and test set.
5. Using the train set to find the optimal value of $\hat{\boldsymbol{\beta}}$ matrix (regression coefficients) for which mean square error is minimum.
6. Predict the values of the output variable on the test set.
7. Perform the performance evaluation of the trained model.

Step 1 (LSE): Load the Dataset

- For implementation of multiple linear regression using least square error fit , we will be using USA housing dataset.
- Code :

```
import pandas as pd
df=pd.read_csv('C:/Users/jasme/Downloads/USA_Housing.csv')
```

You can download the dataset from the following link:

https://drive.google.com/file/d/1O_NwpJT-8xGfU_-3llUl2sgPu0xllOrX/view?usp=sharing

Step 1 (LSE): Load the Dataset Contd.....

- We can check the information regarding dataset using the following code:

df.info()

Output:<class 'pandas.core.frame.DataFrame'>

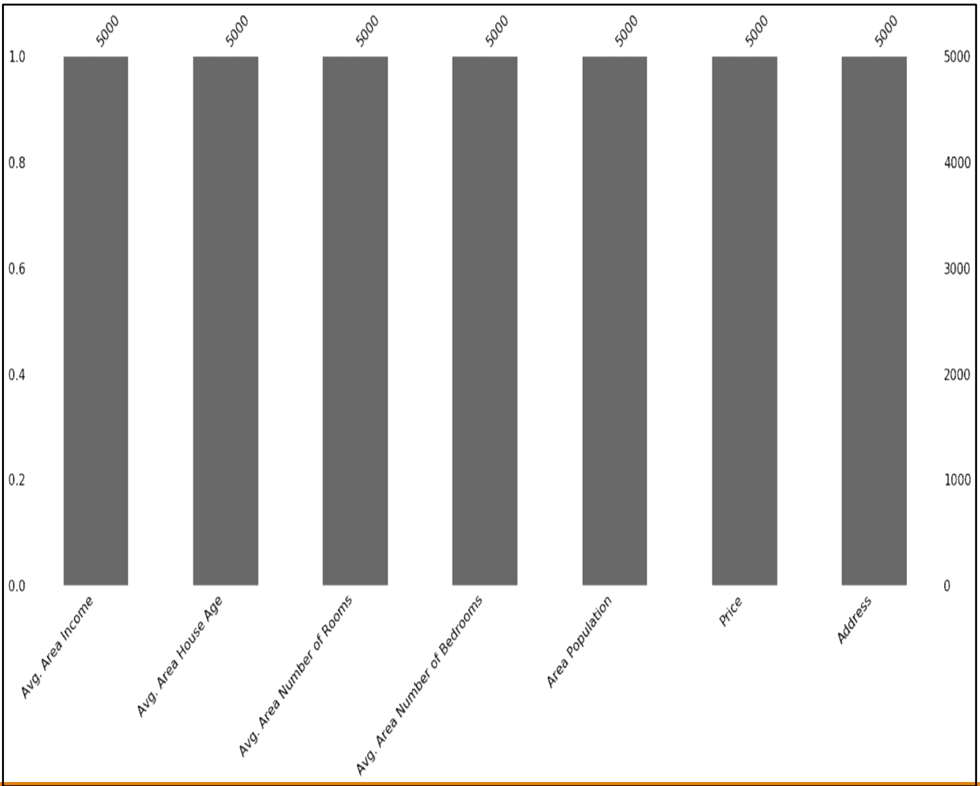
```
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     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
dtypes: float64(6)
memory usage: 234.4 KB
```

Step 2 (LSE): Removing noise

- We can check for null values using missingno library as shown below:
- Code:

```
import missingno as ms
ms.bar(df)
```

- As shown in figure, all columns have 5000 entries. So, there is no missing values.



Step 2 (LSE): Checking Redundancy among features

- We can check redundancy between input features by plotting heatmap of correlation between the input features:

Code:

```
import seaborn as sns

sns.heatmap(df.iloc[:,0:5].corr(),annot=True)
```

- Since correlation between features is less (not greater than 0.7 or 0.8), so feature selection/extraction is not required.

Step 3 (LSE): Split Input & Output Features

- Separate the dataset into X (input/independent variables) and Y (dependent feature).
- Scale the feature values of X in a fixed range
- Add a new column (in the beginning) with all values 1.

Code:

```
import numpy as np
X=df.iloc[:,0:5]
Y=df.iloc[:,5]
Y=np.array(Y)
Y=Y.reshape(-1,1)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
X_scaled= np.insert(X_scaled, 0, values=1, axis=1)
```

Step 4 (LSE): Train/Test Split

- We can split the train and test sets using train/test split of sklearn.model_selection as follows:

Code:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, Y, test_size=0.3, random_state=42)
```

Parameters:

test_size is the percentage of test set from the total dataset; random_state s used for initializing the internal **random** number generator, which will decide the **splitting** of data into **train** and **test** indices

If random_state is None or np.random, then a randomly-initialized RandomState object is returned.

If random_state is an integer, then it is used to seed a new RandomState object.

Step 5 (LSE): Finding Regression Coefficients

- Compute the regression coefficients for which mean least square error is minimum.
- According to least square error method, the mean square error is minimum when $\boldsymbol{\beta}\hat{}=(\boldsymbol{X}^{T}\ \boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{y}$

Code:

```
A=X_train.T.dot(X_train)

B=np.linalg.inv(A)

C=B.dot(X_train.T)

beta=C.dot(y_train)

print(beta)
```

Step 6 (LSE) : Predicting values on test set

- In this step, we predict the values of output variable on the test set.
- It is done by multiplying X_test set with optimal Beta matrix as hown in the code below.

Code:

```
y_predict=X_test.dot(beta)

print(y_predict)
```

Step 7 (LSE): Performance Evaluation

- We check the performance of the trained model by computing following error between the predicted and actual values:
- Mean Square Error
- Root Mean Square Error
- R2_score

Code:

```
error=y_test-y_predict
square_error=np.power(error,2)
sum_square_error=np.sum(square_error)
mean_square_error=sum_square_error/len(y_predict)
print(mean_square_error)
rms_error=np.sqrt(mean_square_error)
print(rms_error)
y_mean=np.mean(y_test)
total_variance=np.sum((y_test-y_mean)**2)
print(1-sum_square_error/total_variance)
```

MLR: Gradient Descent Optimization

- A multiple linear regression model with k independent predictor variables $x_1, x_2 \ldots, x_k$ predicts the output variable as:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots \ldots \ldots \ldots .. + \beta_k x_k$$

- Initialize values of β_0 , β_1, β_2,…………………., β_k to some arbitrary value (usually to 0)
- Repeat until convergence or for fixed number of iterations:

$$\beta_j - \alpha \frac{\partial(J(\beta))}{\partial \beta_j} \; for\; j = 0,1,2, \ldots \ldots .. k \qquad (1)$$

where α is called learning rate; $J(\beta)$ is the mean square error given by:

$$J(\beta) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} - \beta_3 x_{i3} - \cdots \ldots \ldots \ldots .. - \beta_k x_{ik})^2$$

$$\frac{\partial(J(\beta))}{\partial \beta_j} = \frac{1}{n} \sum_{i=1}^{n} (\beta_0 \mp \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \cdots \ldots \ldots \ldots .. + \beta_k x_{ik} - y_i) * x_{ij} \qquad (2)$$

Gradient Descent Optimization: Step- by-Step

- Following steps are followed for implementation of gradient descent optimization:

1. Load the dataset.
2. Remove noise, outliers and check for feature selection or extraction (if required).
3. Separate the dataset into X (input/independent variables) and Y (dependent feature). Scale the feature values of X in a fixed range.
4. Split the dataset into train and test set.
5. Initialize values of regression coefficients to zero and update the coefficients using equations 1 and 2 (in previous slide) for fixed number of iterations and fixed learning rate (simultaneous update).
6. Predict the values of the output variable on the test set.
7. Perform the performance evaluation of the trained model.

*** Steps 1 to 4 are same as Least Square Error fit (except adding column with all 1's at the beginning)**

Step 5 (Gradient Descent): Update Regression Coefficients

```
beta=np.zeros(6)
number_of_iterations=1000
learning_rate=0.01
for i in range(number_of_iterations):
    x0_gradient=0
    x1_gradient=0
    x2_gradient=0
    x3_gradient=0
    x4_gradient=0
    x5_gradient=0
    for j in range(len(X_train)):
        a=X_train[j,0]
        b=X_train[j,1]
        c=X_train[j,2]
        d=X_train[j,3]
        e=X_train[j,4]
        f=Y_train[j]
```

```
        x0_gradient+=(beta[0]+(beta[1]*a)+(beta[2]*b)+(beta[3]*c)+(beta[4]*d)+(beta[5]*e)-f)
        x1_gradient+=((beta[0]+(beta[1]*a)+(beta[2]*b)+(beta[3]*c)+(beta[4]*d)+(beta[5]*e)-f)*a)
        x2_gradient+=((beta[0]+(beta[1]*a)+(beta[2]*b)+(beta[3]*c)+(beta[4]*d)+(beta[5]*e)-f)*b)
        x3_gradient+=((beta[0]+(beta[1]*a)+(beta[2]*b)+(beta[3]*c)+(beta[4]*d)+(beta[5]*e)-f)*c)
        x4_gradient+=((beta[0]+(beta[1]*a)+(beta[2]*b)+(beta[3]*c)+(beta[4]*d)+(beta[5]*e)-f)*d)
        x5_gradient+=((beta[0]+(beta[1]*a)+(beta[2]*b)+(beta[3]*c)+(beta[4]*d)+(beta[5]*e)-f)*e)
    beta[0]=beta[0]-learning_rate/n*x0_gradient
    beta[1]=beta[1]-learning_rate/n*x1_gradient
    beta[2]=beta[2]-learning_rate/n*x2_gradient
    beta[3]=beta[3]-learning_rate/n*x3_gradient
    beta[4]=beta[4]-learning_rate/n*x4_gradient
    beta[5]=beta[5]-learning_rate/n*x5_gradient
print(beta)
```

Step 6: Predict Values on test set

- In this step, we predict the values of output variable on the test set.
- It is done by multiplying X_test set (after adding column with 1 value) with optimal Beta matrix as hown in the code below.

Code:

```
X_test=np.insert(X_test,0,values=1,axis=1)

beta=np.array(beta).reshape(-1,1)

y_predict=X_test.dot(beta)
```

Step 7 (Gradient Descent): Performance Evaluation

- We check the performance of the trained model by computing following error between the predicted and actual values:
- Mean Square Error
- Root Mean Square Error
- R2_score

Code:

```
error=y_test-y_predict
square_error=np.power(error,2)
sum_square_error=np.sum(square_error)
mean_square_error=sum_square_error/len(y_predict)
print(mean_square_error)
rms_error=np.sqrt(mean_square_error)
print(rms_error)
y_mean=np.mean(y_test)
total_variance=np.sum((y_test-y_mean)**2)
print(1-sum_square_error/total_variance)
```

MLR: Inbuilt Functions in Python

- We can also use inbuilt, function of Python.
- For using inbuilt function, steps 1 to 4 are same as for gradient descent optimization.
- In step 5, we fit and transform on the training data using the following code:

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X_train,y_train)
```

- In Step 6, we predict the values of output variable on test set as follows:

```
pred=model.predict(X_test)
```

- In Step 7, we perform performance evaluation using following code:

MLR: Inbuilt Functions in Python Contd….

- In Step 7, we perform performance evaluation using following code:

```
from sklearn import metrics
print(metrics.mean_squared_error(pred,y_test))
print(metrics.mean_absolute_error(pred,y_test))
print(np.sqrt(metrics.mean_squared_error(pred,y_test)))
print(metrics.r2_score(y_test, pred))
```