EEE385L: Machine Learning Laboratory

Lab 3: Implementation of Multiple Linear Regression

Name: Afrida Islam

Student ID: 21221030

Import packages

```
import pandas as pd
#MinMaxScaler is for Normalization & StandardScaler is for Standardization
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import numpy as np
import matplotlib.pyplot as plt
```

Load Dataset from Google Drive

```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Wireless_data.csv")
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Explore the Dataset

```
data.head()
```

	Dataset	Bandwidth	TransmissionPower	ChannelGain	Frequency	UnitCost
0	8.517789	8.221591	6.537314	4.294531	1.631189	27.014253
1	9.246335	6.893047	4.565376	13.763571	1.355074	21.211191
2	3.015895	9.057902	7.591142	4.000898	1.997003	28.592177
3	9.307007	7.664128	6.825546	14.521624	1.224171	23.933344
4	7.058874	6.753636	8.175696	14.266931	1.652451	21.655819

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.iloc[5:7,0:3]
```

	Dataset	Bandwidth	TransmissionPower
5	2.780323	9.695008	8.199327
6	4.227986	9.379714	7.831185

```
data["TransmissionPower"]
```

	TransmissionPower
0	6.537314
1	4.565376
2	7.591142
3	6.825546
4	8.175696
...	...
195	6.971062
196	5.138262
197	6.970035
198	4.885649
199	4.329845

200 rows × 1 columns

```
cols = data.columns
print(cols[1])
```

```
Bandwidth
```

```
data[cols[-1]][6:10]
```

	UnitCost
6	27.580278
7	29.778232
8	10.010448
9	27.308772

Perform Feature Engineering

```
m = 2   # number of features
X = data.iloc[:, 0:m]
print(X)
```

```
      Dataset  Bandwidth
0    8.517789   8.221591
1    9.246335   6.893047
2    3.015895   9.057902
3    9.307007   7.664128
4    7.058874   6.753636
..        ...        ...
195  4.450796   5.112563
196  6.068069   7.126297
197  6.086173   6.563594
198  8.541022   5.807424
199  8.358651   5.893831

[200 rows x 2 columns]
```

```
y = data.iloc[:, -1]
print(y)
```

```
0      27.014253
1      21.211191
2      28.592177
3      23.933344
4      21.655819
         ...
195    11.026638
196    11.457706
197    11.770549
198    25.967017
199    28.860163
Name: UnitCost, Length: 200, dtype: float64
```

```
scaler = StandardScaler()
X_standard = scaler.fit_transform(X)
print(X_standard)
```

Show hidden output

Define the loss function

```
# Modify the cost function to incorporate the bias term (w0)

def mse(w0, w1, w2, w3, w4, w5, X1, X2, X3, X4, X5, y):
    y_hat = w0 + w1 * X1 + w2 * X2 + w3 * X3 + w4 * X4 + w5 * X5
    error_sq = (y - y_hat) ** 2
    mse_value = (1 / len(y)) * np.sum(error_sq)
    return mse_value
```

Implement Gradient Descent to update model parameters

```
def gradient_descent(w0, w1, w2, w3, w4, w5, X1, X2, X3, X4, X5, y, alpha, epoch):
    train_loss = [0] * epoch

    for i in range(epoch):
        # Predictions
        y_hat = w0 + w1 * X1 + w2 * X2 + w3 * X3 + w4 * X4 + w5 * X5
        err = y - y_hat

        # Compute gradients
        w0_gradient = (-2 / len(y)) * np.sum(err)
        w1_gradient = (-2 / len(y)) * np.dot(X1, err)
        w2_gradient = (-2 / len(y)) * np.dot(X2, err)
        w3_gradient = (-2 / len(y)) * np.dot(X3, err)
        w4_gradient = (-2 / len(y)) * np.dot(X4, err)
        w5_gradient = (-2 / len(y)) * np.dot(X5, err)

        # Update weights and bias
        w0 -= alpha * w0_gradient
        w1 -= alpha * w1_gradient
        w2 -= alpha * w2_gradient
        w3 -= alpha * w3_gradient
        w4 -= alpha * w4_gradient
        w5 -= alpha * w5_gradient

        # Calculate training loss
        train_loss[i] = mse(w0, w1, w2, w3, w4, w5, X1, X2, X3, X4, X5, y)
        print(f"Epoch {i+1}: Training Loss - {train_loss[i]}")

    return w0, w1, w2, w3, w4, w5, train_loss
```

Prepare the Dataset for training

```
n = int(0.8 * len(y))  # Number of training samples

X_train = X_standard[:n, :]  # Training features
X_test = X_standard[n:, :]   # Testing features

y_train = y[:n]    # Training targets
y_test = y[n:]     # Testing targets

# Separate the individual features for gradient descent
X1_train = X_train[:, 0]
X2_train = X_train[:, 1]
X3_train = X_train[:, 2]
X4_train = X_train[:, 3]
X5_train = X_train[:, 4]

X1_test = X_test[:, 0]
X2_test = X_test[:, 1]
X3_test = X_test[:, 2]
X4_test = X_test[:, 3]
X5_test = X_test[:, 4]
```

```
y_train = y[0:n]
print(y_train)
y_test = y[n: ]
print(y_test)
```

Show hidden output

Train the model

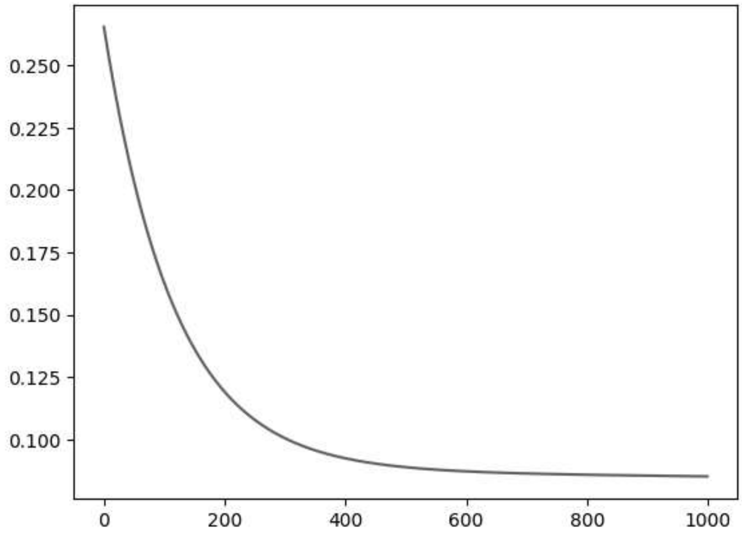
```
w0 = 0  # Bias term
w1, w2, w3, w4, w5 = 0, 0, 0, 0, 0
alpha = 0.001
epoch = 1000

# Run gradient descent to fit the model
updated_w0, updated_w1, updated_w2, updated_w3, updated_w4, updated_w5, loss = gradient_descent(w0, w1, w2, w3, w4, w5, X1_train, X2_trai
```

Show hidden output

Plot the loss in each epoch

```
plt.plot(loss)
```

```
[<matplotlib.lines.Line2D at 0x7daa4657b9d0>]
```




```
# Install the package for Tex and then convert to PDF directly as LaTex
!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic pandoc

# Provide the file path of the notebook file
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/21221030_Lab_3.ipynb"
```

Show hidden output