ECGR_4105_HW5_Problem_2_Source_Code

November 16, 2024

```
[ ]: # Link to Google Colab: https://colab.research.google.com/drive/
      ↪1omnTRKEN9-bOJuBJq-dpHY5SwLbxeL5O?usp=sharing
```

```
[6]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt

     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
```

```
[2]: file_url = 'https://raw.githubusercontent.com/lnguye782/ECGR-4105-Intro-to-ML/
      ↪refs/heads/main/HW2/Housing.csv'
     data = pd.read_csv(file_url)

     data.head(), data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

```
[2]: (      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
      0  13300000  7420         4          2        3      yes        no       no
      1  12250000  8960         4          4        4      yes        no       no
      2  12250000  9960         3          2        2      yes        no      yes
      3  12215000  7500         4          2        2      yes        no      yes
      4  11410000  7420         4          1        2      yes       yes      yes

        hotwaterheating airconditioning  parking prefarea furnishingstatus
      0              no             yes        2      yes        furnished
      1              no             yes        3       no        furnished
      2              no              no        2      yes   semi-furnished
      3              no             yes        3      yes        furnished
      4              no             yes        2       no        furnished  ,
      None)
```

```
[3]: # Select relevant columns
     selected_columns = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking',␣
      ↪'price']
     data_selected = data[selected_columns]

     # Features and target variable
     X = data_selected.drop(columns='price')
     y = data_selected['price']

     # Split the data into training (80%) and validation (20%) sets
     X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
      ↪random_state=42)

     # Standardize features (normalize)
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_val_scaled = scaler.transform(X_val)

     # Convert target to numpy arrays
     y_train = y_train.values
     y_val = y_val.values

     # Check processed shapes
     X_train_scaled.shape, X_val_scaled.shape, y_train.shape, y_val.shape
```

```
[3]: ((436, 5), (109, 5), (436,), (109,))
```

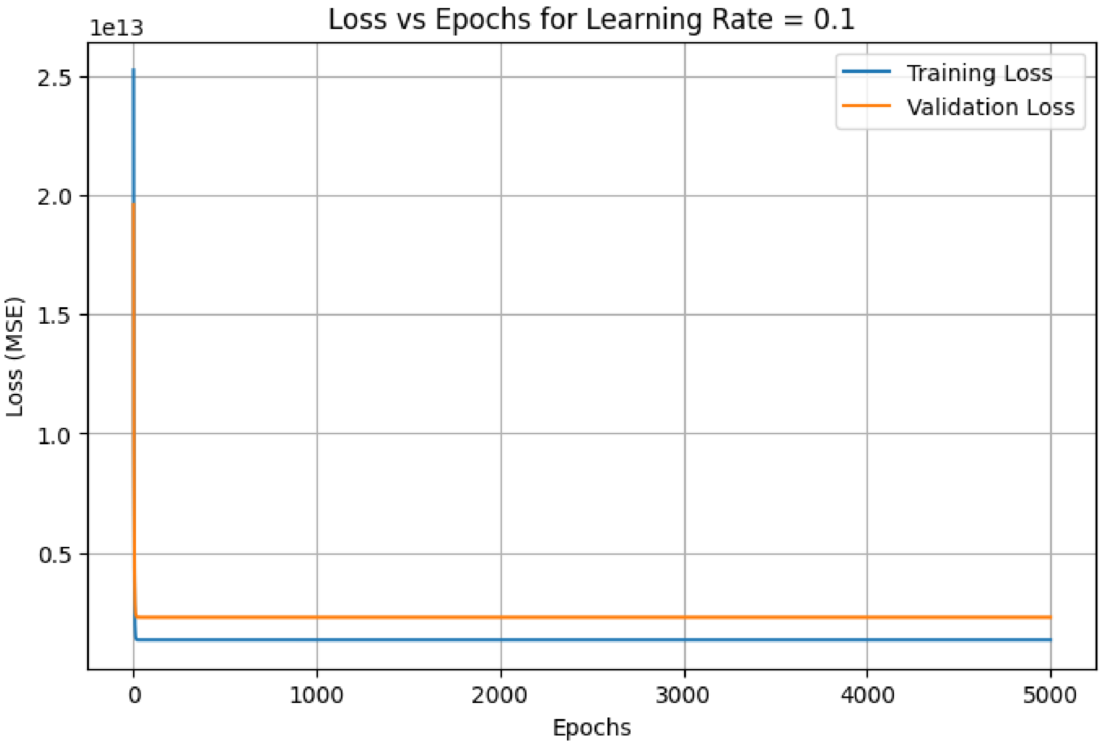
```
[5]: # Linear Regression Model using Gradient Descent
     class LinearRegressionGD:
         def __init__(self, learning_rate=0.01, epochs=5000):
             self.learning_rate = learning_rate
             self.epochs = epochs
```

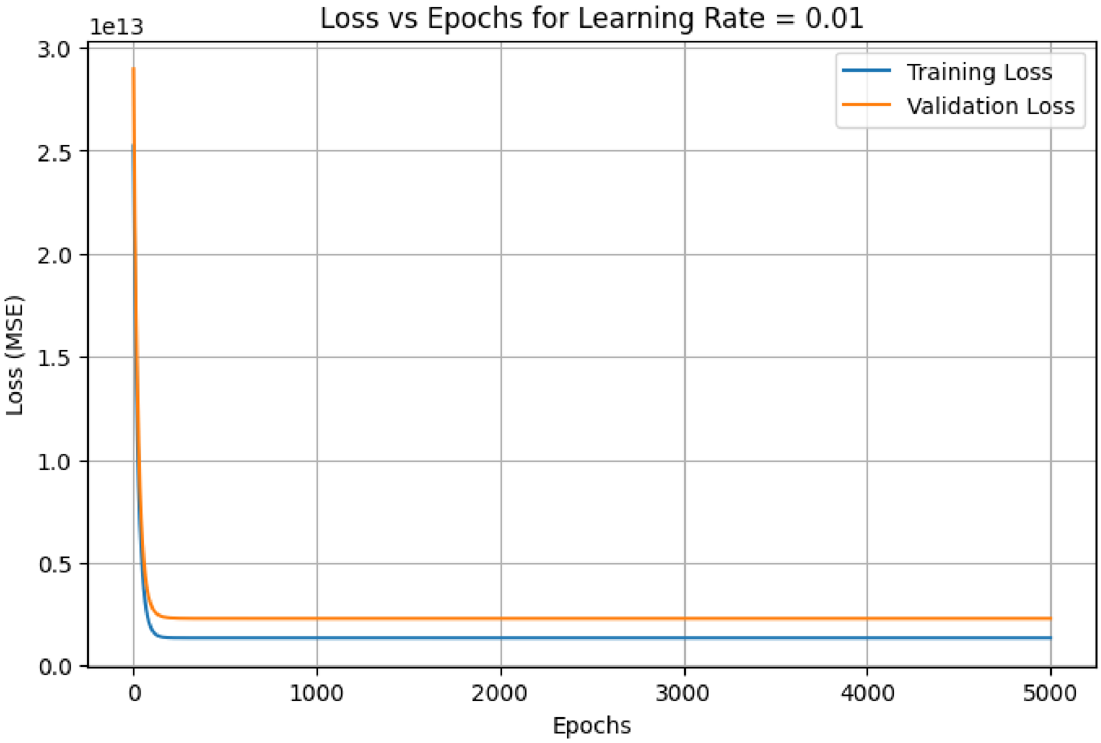
```python
        self.weights = None
        self.bias = None

    def initialize_parameters(self, n_features):
        self.weights = np.zeros(n_features)
        self.bias = 0

    def compute_loss(self, y_true, y_pred):
        # Mean Squared Error
        return np.mean((y_true - y_pred) ** 2)

    def train(self, X, y, X_val, y_val):
        n_samples, n_features = X.shape
        self.initialize_parameters(n_features)

        train_loss = []
        val_loss = []

        # Training loop
        for epoch in range(self.epochs):
            # Predictions
            y_pred = np.dot(X, self.weights) + self.bias

            # Compute gradients
            dw = -(2 / n_samples) * np.dot(X.T, (y - y_pred))
            db = -(2 / n_samples) * np.sum(y - y_pred)

            # Update parameters
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

            # Compute loss
            loss = self.compute_loss(y, y_pred)
            train_loss.append(loss)

            # Validation loss
            y_val_pred = np.dot(X_val, self.weights) + self.bias
            val_loss.append(self.compute_loss(y_val, y_val_pred))

            # Print progress every 500 epochs
            if (epoch + 1) % 500 == 0:
                print(f"Epoch {epoch + 1}/{self.epochs}: Train Loss = {loss:.
↪4f}, Validation Loss = {val_loss[-1]:.4f}")

        return train_loss, val_loss

# Train with different learning rates
```

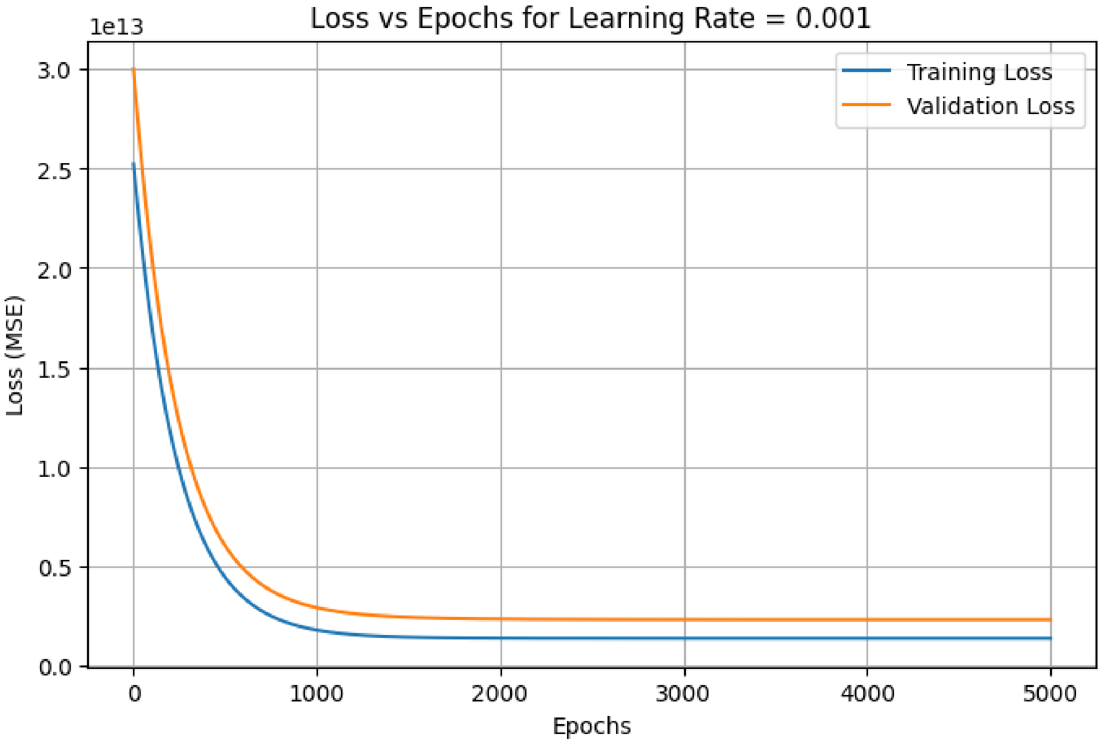
```
learning_rates = [0.1, 0.01, 0.001, 0.0001]
results = {}

for lr in learning_rates:
    print(f"\nTraining with Learning Rate: {lr}")
    model = LinearRegressionGD(learning_rate=lr, epochs=5000)
    train_loss, val_loss = model.train(X_train_scaled, y_train, X_val_scaled,␣
 ↪y_val)
    results[lr] = (train_loss, val_loss, model.weights, model.bias)
```

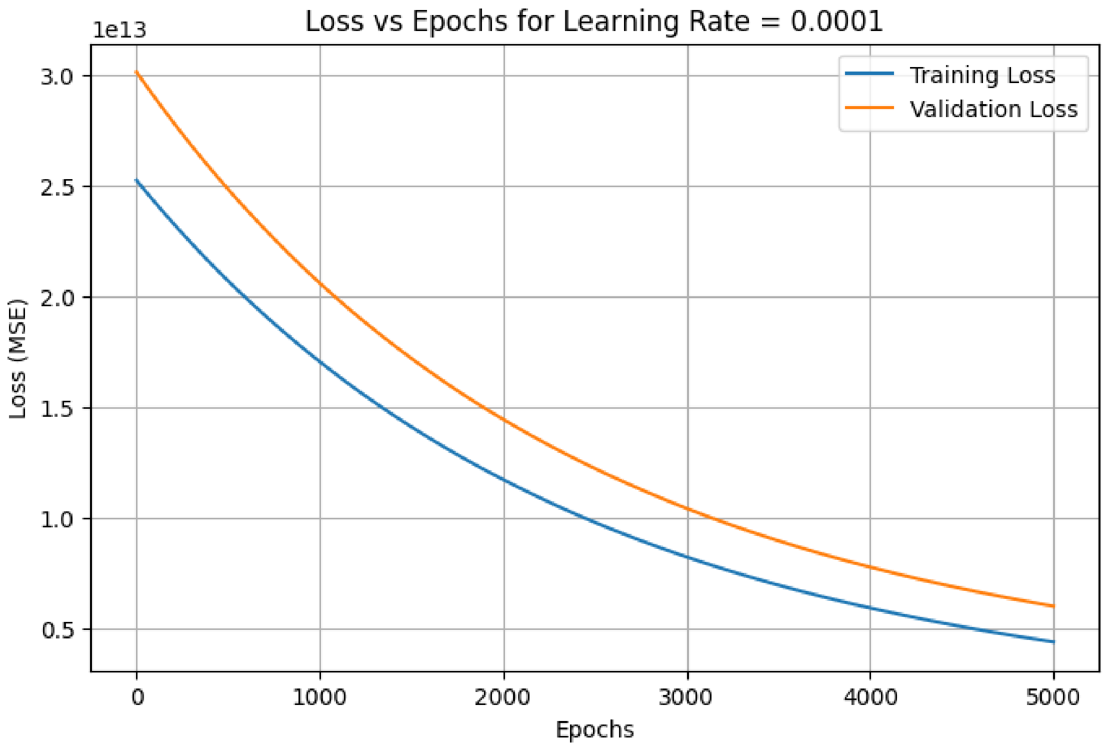
```
Training with Learning Rate: 0.1
Epoch 500/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 1000/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 1500/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 2000/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 2500/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 3000/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 3500/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 4000/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 4500/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662
Epoch 5000/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3662

Training with Learning Rate: 0.01
Epoch 500/5000: Train Loss = 1350009010876.5654, Validation Loss =
2292813178774.1748
Epoch 1000/5000: Train Loss = 1350008211333.5713, Validation Loss =
2292721692029.6787
Epoch 1500/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721546204.0869
Epoch 2000/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545726.9561
Epoch 2500/5000: Train Loss = 1350008211326.5803, Validation Loss =
2292721545725.3770
Epoch 3000/5000: Train Loss = 1350008211326.5801, Validation Loss =
2292721545725.3726
Epoch 3500/5000: Train Loss = 1350008211326.5801, Validation Loss =
2292721545725.3726
Epoch 4000/5000: Train Loss = 1350008211326.5801, Validation Loss =
2292721545725.3726
Epoch 4500/5000: Train Loss = 1350008211326.5801, Validation Loss =
2292721545725.3726
Epoch 5000/5000: Train Loss = 1350008211326.5801, Validation Loss =
2292721545725.3726

Training with Learning Rate: 0.001
Epoch 500/5000: Train Loss = 4428910450828.4600, Validation Loss =
6026504766422.3066
Epoch 1000/5000: Train Loss = 1767420758090.9146, Validation Loss =
2894953483806.9399
Epoch 1500/5000: Train Loss = 1407960624252.7695, Validation Loss =
2413169011139.9165
Epoch 2000/5000: Train Loss = 1358342820260.3408, Validation Loss =
2323576806063.2661
Epoch 2500/5000: Train Loss = 1351290148936.9075, Validation Loss =
2302396127424.1353
Epoch 3000/5000: Train Loss = 1350229453600.0400, Validation Loss =
2296156481221.6343
Epoch 3500/5000: Train Loss = 1350052956926.5608, Validation Loss =
2294032466154.8330
Epoch 4000/5000: Train Loss = 1350018866646.1482, Validation Loss =
2293247896566.0298
Epoch 4500/5000: Train Loss = 1350011086718.7402, Validation Loss =
2292942769433.6924
Epoch 5000/5000: Train Loss = 1350009048453.4460, Validation Loss =
2292818935060.5659

Training with Learning Rate: 0.0001
Epoch 500/5000: Train Loss = 20705542685265.9922, Validation Loss =
24839773808034.9570
Epoch 1000/5000: Train Loss = 17056916524493.8281, Validation Loss =
20604240673950.2227
Epoch 1500/5000: Train Loss = 14115695294628.7383, Validation Loss =
17201089604829.7383
Epoch 2000/5000: Train Loss = 11738794306343.0312, Validation Loss =
14458005933892.8828
Epoch 2500/5000: Train Loss = 9813833385830.5566, Validation Loss =
12240607727733.4473
Epoch 3000/5000: Train Loss = 8252043228973.9297, Validation Loss =
10443501933882.5371
Epoch 3500/5000: Train Loss = 6982947086155.4678, Validation Loss =
8983600956273.5117
Epoch 4000/5000: Train Loss = 5950339148635.6992, Validation Loss =
7795091394560.7373
Epoch 4500/5000: Train Loss = 5109218014923.4170, Validation Loss =
6825618923305.2949
Epoch 5000/5000: Train Loss = 4423429686043.9248, Validation Loss =
6033374860840.7676
```

[8]:
```
# Plot Training and Validation Loss for each learning rate
for lr, (train_loss, val_loss, _, _) in results.items():
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, len(train_loss) + 1), train_loss, label="Training Loss")
    plt.plot(range(1, len(val_loss) + 1), val_loss, label="Validation Loss")
    plt.title(f"Loss vs Epochs for Learning Rate = {lr}")
    plt.xlabel("Epochs")
    plt.ylabel("Loss (MSE)")
    plt.legend()
    plt.grid()
    plt.show()
```




Loss vs Epochs for Learning Rate = 0.01
1e13
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss (MSE)
0
1000
2000
3000
4000
5000
Epochs
Training Loss
Validation Loss


Loss vs Epochs for Learning Rate = 0.001
1e13
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss (MSE)
0
1000
2000
3000
4000
5000
Epochs
Training Loss
Validation Loss


Loss vs Epochs for Learning Rate = 0.0001
1e13
3.0
2.5
2.0
1.5
1.0
0.5
Loss (MSE)
0
1000
2000
3000
4000
5000
Epochs
Training Loss
Validation Loss