EN3150 Assignment 01: Learning from Data and Related Challenges

220481U PITIGALA P.K.N.W.

Department of Electronic and Telecommunication Engineering
University of Moratuwa, Sri Lanka

Full Code: https://github.com/Nishitha0730

1 Linear Regression and Impact of Outliers

1.1 Task 2: Linear Regression Model

Using the dataset in Table 1, a linear regression model was developed using the least squares method. The model is:

Linear Regression Model: $y = -3.55x + 3.91$

The scatter plot of the data points and the fitted model is shown below:

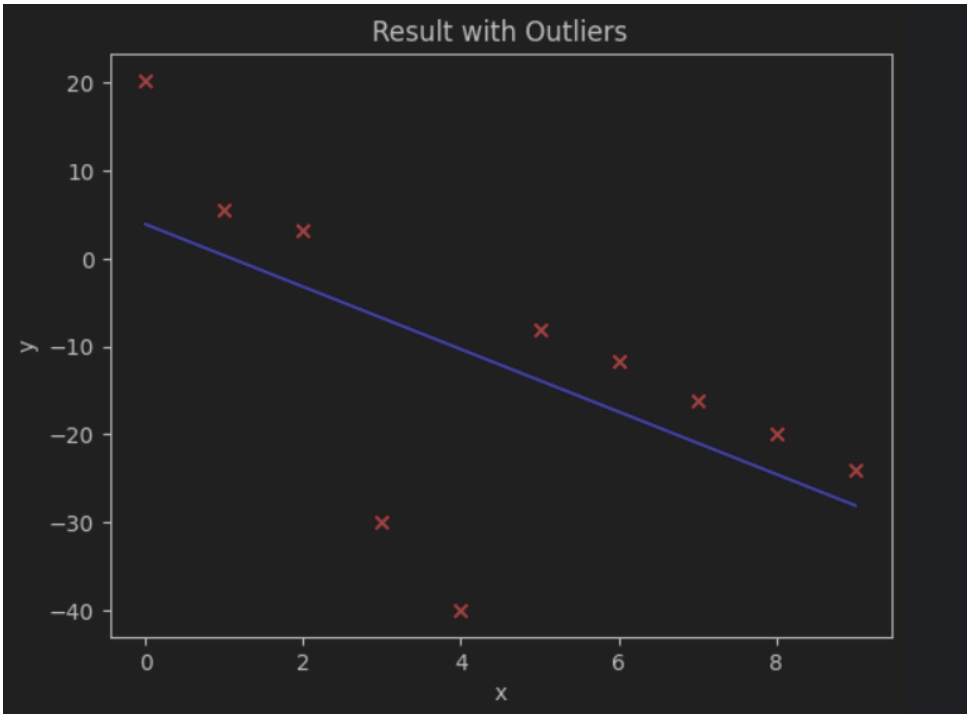


Figure 1: Scatter plot of data points with linear regression model $y = -3.55x + 3.91$.

```python
import numpy as np
import copy

def compute_cost(x, y, w, b):
    m = len(x)
    cost = 0
    for i in range(m):
        f_wb = np.dot(w, x[i]) + b
        cost += (f_wb - y[i]) ** 2
    cost /= (2 * m)
    return cost

def compute_gradient(x, y, w, b):
```

```
    m = len(x)
    dj_dw = 0
    dj_db = 0
    for i in range(m):
        f_wb = np.dot(w, x[i]) + b
        error = f_wb - y[i]
        dj_dw += error * x[i]
        dj_db += error
    dj_dw /= m
    dj_db /= m
    return dj_dw, dj_db

def compute_gradient_descent(x, y, w_in, b_in, cost_function,
   gradient_function, alpha, num_iteration):
    m = len(x)
    cost_history = np.zeros(num_iteration)
    w_history = np.zeros(num_iteration)
    b_history = np.zeros(num_iteration)
    w = copy.deepcopy(w_in)
    b = b_in
    for i in range(num_iteration):
        dj_dw, dj_db = gradient_function(x, y, w, b)
        w -= alpha * dj_dw
        b -= alpha * dj_db
        cost = cost_function(x, y, w, b)
        cost_history[i] = cost
        w_history[i] = w
        b_history[i] = b
        if i % 100 == 0:
            print(f"Iteration {i}: Cost {cost}, w {w}, b {b}")
    return w, b, cost_history, w_history, b_history

initial_w = 0.
initial_b = 0.
iterations = 15000
alpha = 0.0684

w, b, J_history, w_history, b_history = compute_gradient_descent(
    x_train, y_train, initial_w, initial_b, compute_cost,
       compute_gradient, alpha, iterations
)
print("w, b found by gradient descent:", w, b)
```

Listing 1: Gradient Descent for Linear Regression

1.2 Task 4: Loss Function Calculation

The robust estimator loss function is:

$$L(\theta, \beta) = \frac{1}{N} \sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{(y_i - \hat{y}_i)^2 + \beta^2},$$

where $N = 10$. Loss values were calculated for Model 1 ($y = -4x + 12$) and Model 2 ($y = -3.55x + 3.91$) for $\beta = 1$, $\beta = 10^{-6}$, and $\beta = 10^3$. The residuals $(y_i - \hat{y}_i)^2$ were computed for each model using the dataset in Table 1, and the loss function was evaluated.

x	y
0	20.26
1	5.61
2	3.14
3	-30.00
4	-40.00
5	-8.13
6	-11.73
7	-16.08
8	-19.95
9	-24.03

Table 1: Dataset for linear regression.

Model	$\beta = 1$	$\beta = 10^{-6}$	$\beta = 10^3$
Model 1	0.4354	0.9999	0.00022
Model 2	0.97324	0.9999	0.000188

Table 2: Loss function values for Models 1 and 2.

```
import numpy as np

# Given data
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y_true = np.array([20.26, 5.61, 3.14, -30.00, -40.00, -8.13,
   -11.73, -16.08, -19.95, -24.03])
```

```
6
7  # Models
8  def model1(x): return -4*x + 12
9  def model2(x): return -3.55*x + 3.91
10
11 y_pred1 = model1(x)
12 y_pred2 = model2(x)
13
14 # Robust loss function
15 def robust_loss_per_point(y_true, y_pred, beta):
16     errors = y_true - y_pred
17     losses = (errors**2) / (errors**2 + beta**2)
18     return losses, np.mean(losses)
19
20 # Beta values
21 betas = [1, 1e-6, 1e3]
22
23 # Compute losses
24 for beta in betas:
25     losses1, avg1 = robust_loss_per_point(y_true, y_pred1, beta)
26     losses2, avg2 = robust_loss_per_point(y_true, y_pred2, beta)
27     print(f"Beta = {beta}: Model 1 Loss = {avg1:.6f}, Model 2 Loss
       = {avg2:.6f}")
```

Listing 2: Robust Loss Function

1.3 Task 5: Suitable β Value to Mitigate Outliers

The suitable β value to mitigate the impact of outliers is $\beta = 1$.

Justification: The dataset in Table 1 contains outliers at $x = 3$ ($y = -30.00$) and $x = 4$ ($y = -40.00$), which deviate significantly from the general trend. For $\beta = 1$, the loss function $L(\theta, \beta) = \frac{1}{N}\sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{(y_i - \hat{y}_i)^2 + \beta^2}$ normalizes residuals by $(y_i - \hat{y}_i)^2 + 1$, reducing the influence of large residuals (outliers) while maintaining sensitivity to smaller residuals. For $\beta = 10^{-6}$, the loss approaches 0.9999 for both models (Table 2), indicating insensitivity to differences in fit, as all residuals contribute nearly equally. For $\beta = 10^3$, the loss is very small (0.00022 for Model 1, 0.000188 for Model 2), overly suppressing the impact of all residuals, including non-outliers, which may obscure meaningful variations in the data. The loss values for $\beta = 1$ (0.4354 for Model 1, 0.97324 for Model 2) provide a balanced measure, effectively mitigating outliers while allowing the models to be compared based on their fit to the data.

1.4 Task 6: Most Suitable Model

Using the robust estimator with $\beta = 1$, Model 1 ($y = -4x + 12$) is the most suitable.

Justification: From Table 2, Model 1 has a significantly lower loss (0.4354) compared to Model 2 (0.97324) for $\beta = 1$. This indicates that Model 1 provides a better fit to the data while effectively mitigating the impact of outliers at $x = 3$ ($y = -30.00$) and $x = 4$ ($y = -40.00$). Although Model 2 ($y = -3.55x + 3.91$) is derived from the least squares method and may align closely with non-outlier points, its higher robust loss suggests it is more affected by outliers. Model 1's lower loss demonstrates its robustness, making it the preferred choice for this dataset under the robust estimator with $\beta = 1$.

1.5 Task 7: How Robust Estimator Reduces Outlier Impact

The robust estimator reduces the impact of outliers by using a loss function that weights residuals based on their magnitude relative to β. For large residuals (outliers), the denominator $(y_i - \hat{y}_i)^2 + \beta^2$ reduces their contribution to the total loss, unlike mean squared error, which amplifies large residuals quadratically. This ensures that outliers do not disproportionately influence the model parameters, leading to a more robust fit.

1.6 Task 8: Alternative Loss Function

Another loss function for the robust estimator is the Huber loss.

Description: The Huber loss combines squared error for small residuals and linear error for large residuals, defined as:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta, \\ \delta|a| - \frac{1}{2}\delta^2 & \text{if } |a| > \delta, \end{cases}$$

where $a = y_i - \hat{y}_i$. It is suitable for robust estimation as it reduces the influence of outliers (linear term) while maintaining sensitivity to smaller residuals (quadratic term).

2 Loss Function Analysis

2.1 Task 1: MSE and BCE Loss Values

The Mean Squared Error (MSE) and Binary Cross Entropy (BCE) loss values for $y = 1$ and various predictions $\hat{y}$ are calculated using:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)].$$

For $n = 1$ and $y = 1$:

Prediction $\hat{y}$	MSE	BCE
0.005	0.9900	5.2983
0.01	0.9801	4.6052
0.05	0.9025	2.9957
0.1	0.8100	2.3026
0.2	0.6400	1.6094
0.3	0.4900	1.2040
0.4	0.3600	0.9163
0.5	0.2500	0.6931
0.6	0.1600	0.5108
0.7	0.0900	0.3567
0.8	0.0400	0.2231
0.9	0.0100	0.1054
1.0	0.0000	0.0000

Table 3: MSE and BCE loss values for $y = 1$.

```python
import pandas as pd
import numpy as np

data = pd.read_excel('q2.xlsx')
y = data['y']
yhat = data['yhat']

# Calculate per-sample MSE values
data['mse'] = (y - yhat) ** 2

# Calculate per-sample BCE values
epsilon = 1e-15
yhat_clipped = np.clip(yhat, epsilon, 1 - epsilon)
```

```
data['bce'] = -(y * np.log(yhat_clipped) + (1 - y) * np.log(1 -
    yhat_clipped))

overall_mse = np.mean((y - yhat) ** 2)
overall_bce = -np.mean(y * np.log(yhat_clipped) + (1 - y) *
    np.log(1 - yhat_clipped))

print(f"Overall MSE: {overall_mse}")
print(f"Overall BCE: {overall_bce}")

# Save back to Excel
data.to_excel('q2.xlsx', index=False)
```

Listing 3: MSE and BCE Calculation

2.2 Task 2: Loss Function Selection

For Application 1 (continuous dependent variable), the suitable loss function is MSE.

Justification: MSE measures the average squared difference between predicted and actual values, making it ideal for continuous variables where the goal is to minimize prediction errors across a range of values.

For Application 2 (binary dependent variable), the suitable loss function is BCE.

Justification: BCE is designed for binary classification, as it penalizes predictions based on their logarithmic probability, ensuring that the model outputs probabilities between 0 and 1 that align with binary outcomes.

3 Data Pre-processing

3.1 Task 1: Feature Scaling

The features were generated using the provided code, producing Feature 1 (a sparse signal with 10 non-zero values) and Feature 2 (Gaussian noise with mean 0 and standard deviation 15). Three scaling methods were tested: standard scaling, min-max scaling, and max-abs scaling. The selected scaling method for both features is standard scaling.

Justification: - **Standard Scaling**: For Feature 1, standard scaling preserves sparsity, as zero

values remain zero after subtracting the mean (which is small due to the sparse nature of the data) and dividing by the standard deviation. This maintains the distinction between zero and non-zero values, crucial for the sparse structure. For Feature 2, standard scaling preserves the Gaussian distribution by centering the data at mean 0 and scaling to unit variance, maintaining its statistical properties. - **Min-Max Scaling**: This method scales both features to [0, 1]. For Feature 1, it compresses the sparse signal into a narrow range, reducing the distinction between zero and non-zero values, which may obscure the sparse structure. For Feature 2, it forces the Gaussian noise into [0, 1], altering its distribution and clipping negative values, which is less desirable for maintaining statistical properties. - **Max-Abs Scaling**: This method scales both features to [-1, 1] based on the maximum absolute value. For Feature 1, it preserves sparsity better than min-max scaling, as zeros remain zero, but it may overemphasize the largest value (e.g., at index 10, influenced by the index number). For Feature 2, it does not preserve the Gaussian distribution as effectively as standard scaling, as it scales based on the maximum absolute value, which varies with random noise. - Standard scaling is chosen for both features because it consistently preserves the key properties of sparsity for Feature 1 and the Gaussian distribution for Feature 2, making it suitable for downstream machine learning tasks.

The following code snippets were used to apply the scaling methods:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# Reshape for scikit-learn (2D array expected)
sparse_signal_reshaped = sparse_signal.reshape(-1, 1)
epsilon_reshaped = epsilon.reshape(-1, 1)
# Apply standard scaling
scaled_feature1 =
    scaler.fit_transform(sparse_signal_reshaped).flatten()
scaled_feature2 = scaler.fit_transform(epsilon_reshaped).flatten()
```

Listing 4: Standard Scaling

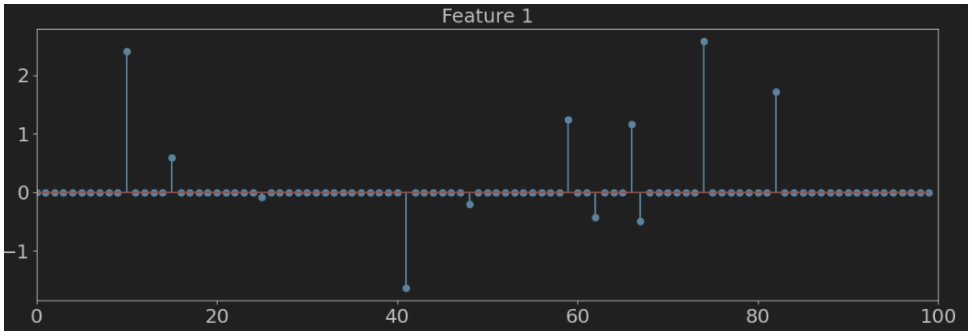
```
from sklearn.preprocessing import MinMaxScaler

minmax_scaler = MinMaxScaler()
# Reshape for scikit-learn (2D array expected)
sparse_signal_reshaped = sparse_signal.reshape(-1, 1)
epsilon_reshaped = epsilon.reshape(-1, 1)
# Apply min-max scaling
minmax_scaled_feature1 =
    minmax_scaler.fit_transform(sparse_signal_reshaped).flatten()
minmax_scaled_feature2 =
    minmax_scaler.fit_transform(epsilon_reshaped).flatten()
```

Listing 5: Min-Max Scaling

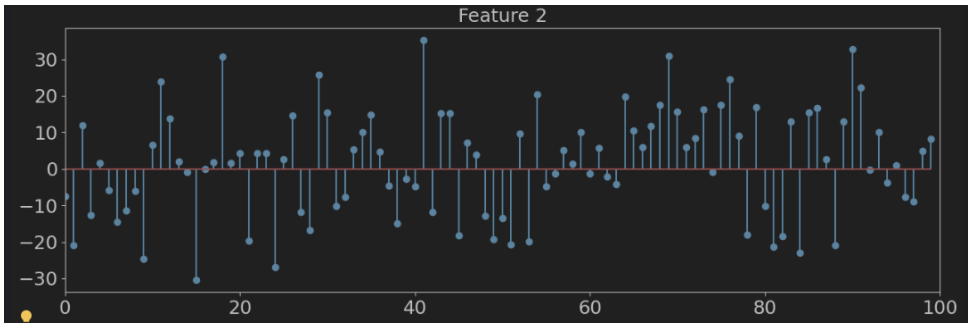
```
from sklearn.preprocessing import MaxAbsScaler

maxabs_scaler = MaxAbsScaler()
# Reshape for scikit-learn (2D array expected)
sparse_signal_reshaped = sparse_signal.reshape(-1, 1)
epsilon_reshaped = epsilon.reshape(-1, 1)
# Apply max-abs scaling
maxabs_scaled_feature1 =
    maxabs_scaler.fit_transform(sparse_signal_reshaped).flatten()
maxabs_scaled_feature2 =
    maxabs_scaler.fit_transform(epsilon_reshaped).flatten()
```

Listing 6: Max-Abs Scaling

The scaled features for each method are shown below:

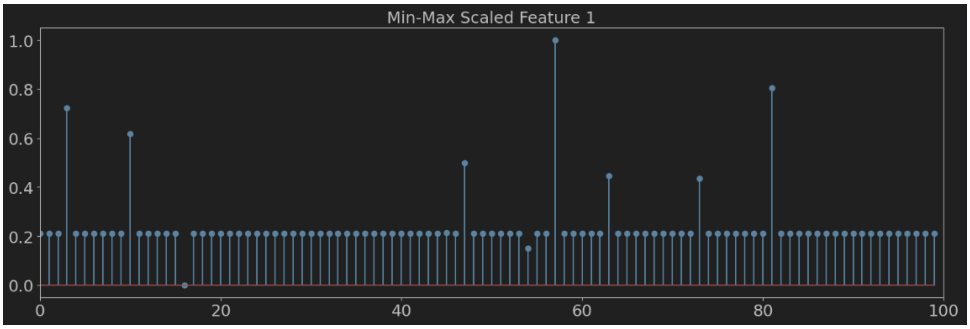


(a) Scaled Feature 1 (Standard Scaling)

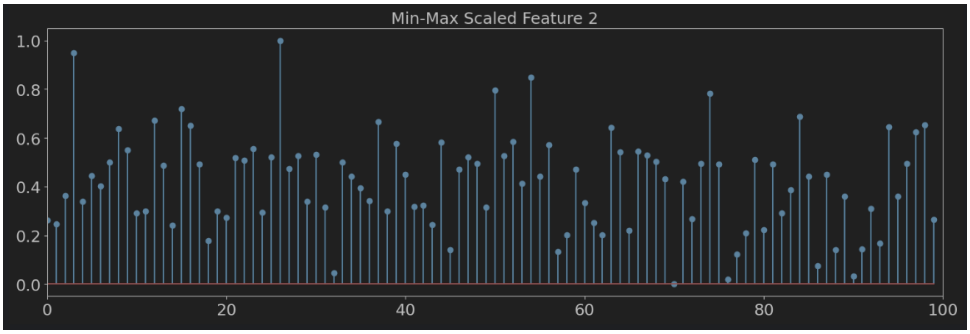


(b) Scaled Feature 2 (Standard Scaling)

Figure 2: Features scaled using standard scaling.



(a) Min-Max Scaled Feature 1



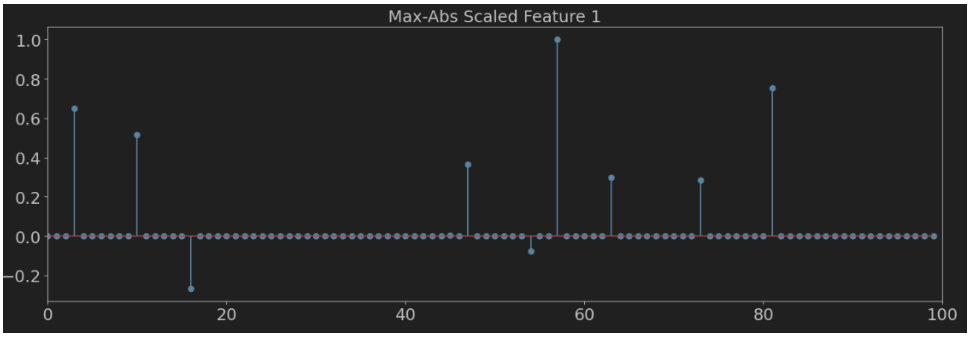
(b) Min-Max Scaled Feature 2

Figure 3: Features scaled using min-max scaling.

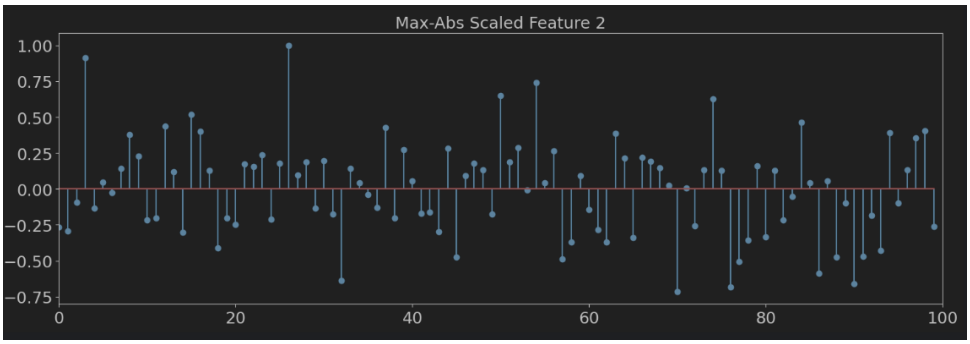
A Source Code

A.1 Python Code for Feature Generation and Scaling

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, MinMaxScaler,
    MaxAbsScaler

# Function to generate sparse signal
def generate_signal(signal_length, num_nonzero):
    signal = np.zeros(signal_length)
    nonzero_indices = np.random.choice(signal_length, num_nonzero,
        replace=False)
    nonzero_values = 10 * np.random.randn(num_nonzero)
    signal[nonzero_indices] = nonzero_values
    return signal

# Parameters
signal_length = 100
num_nonzero = 10
```



(a) Max-Abs Scaled Feature 1



(b) Max-Abs Scaled Feature 2

Figure 4: Features scaled using max-abs scaling.

```
your_index_no = 220481  # Index number

# Generate Feature 1
sparse_signal = generate_signal(signal_length, num_nonzero)
sparse_signal[10] = (your_index_no % 10) * 2 + 10
sparse_signal = sparse_signal / 5

# Generate Feature 2
epsilon = np.random.normal(0, 15, signal_length)

# --- Standard Scaling ---
scaler = StandardScaler()
sparse_signal_reshaped = sparse_signal.reshape(-1, 1)
epsilon_reshaped = epsilon.reshape(-1, 1)
scaled_feature1 =
    scaler.fit_transform(sparse_signal_reshaped).flatten()
scaled_feature2 = scaler.fit_transform(epsilon_reshaped).flatten()

# Plot standard scaled features
plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)
```

```
plt.xlim(0, signal_length)
plt.title("Scaled Feature 1 (Standard Scaling)", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.stem(scaled_feature1)
plt.subplot(2, 1, 2)
plt.xlim(0, signal_length)
plt.title("Scaled Feature 2 (Standard Scaling)", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.stem(scaled_feature2)
plt.tight_layout()
plt.savefig('feature1.png')
plt.savefig('feature2.png')
plt.show()

# --- Min-Max Scaling ---
minmax_scaler = MinMaxScaler()
minmax_scaled_feature1 =
    minmax_scaler.fit_transform(sparse_signal_reshaped).flatten()
minmax_scaled_feature2 =
    minmax_scaler.fit_transform(epsilon_reshaped).flatten()

# Plot min-max scaled features
plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)
plt.xlim(0, signal_length)
plt.title("Min-Max Scaled Feature 1", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.stem(minmax_scaled_feature1)
plt.subplot(2, 1, 2)
plt.xlim(0, signal_length)
plt.title("Min-Max Scaled Feature 2", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.stem(minmax_scaled_feature2)
plt.tight_layout()
plt.savefig('minmax_feature1.png')
plt.savefig('minmax_feature2.png')
plt.show()

# --- Max-Abs Scaling ---
```

```
maxabs_scaler = MaxAbsScaler()
maxabs_scaled_feature1 =
    maxabs_scaler.fit_transform(sparse_signal_reshaped).flatten()
maxabs_scaled_feature2 =
    maxabs_scaler.fit_transform(epsilon_reshaped).flatten()

# Plot max-abs scaled features
plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)
plt.xlim(0, signal_length)
plt.title("Max-Abs Scaled Feature 1", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.stem(maxabs_scaled_feature1)
plt.subplot(2, 1, 2)
plt.xlim(0, signal_length)
plt.title("Max-Abs Scaled Feature 2", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.stem(maxabs_scaled_feature2)
plt.tight_layout()
plt.savefig('maxabs_feature1.png')
plt.savefig('maxabs_feature2.png')
plt.show()
```

Listing 7: Python code for generating and scaling features

A.2 Python Code for Other Tasks

```
import numpy as np
import pandas as pd
import copy

# Linear Regression (Task 2)
def compute_cost(x, y, w, b):
    m = len(x)
    cost = 0
    for i in range(m):
        f_wb = np.dot(w, x[i]) + b
        cost += (f_wb - y[i]) ** 2
    cost /= (2 * m)
    return cost
```

```
14
15 def compute_gradient(x, y, w, b):
16     m = len(x)
17     dj_dw = 0
18     dj_db = 0
19     for i in range(m):
20         f_wb = np.dot(w, x[i]) + b
21         error = f_wb - y[i]
22         dj_dw += error * x[i]
23         dj_db += error
24     dj_dw /= m
25     dj_db /= m
26     return dj_dw, dj_db
27
28 def compute_gradient_descent(x, y, w_in, b_in, cost_function,
       gradient_function, alpha, num_iteration):
29     m = len(x)
30     cost_history = np.zeros(num_iteration)
31     w_history = np.zeros(num_iteration)
32     b_history = np.zeros(num_iteration)
33     w = copy.deepcopy(w_in)
34     b = b_in
35     for i in range(num_iteration):
36         dj_dw, dj_db = gradient_function(x, y, w, b)
37         w -= alpha * dj_dw
38         b -= alpha * dj_db
39         cost = cost_function(x, y, w, b)
40         cost_history[i] = cost
41         w_history[i] = w
42         b_history[i] = b
43         if i % 100 == 0:
44             print(f"Iteration {i}: Cost {cost}, w {w}, b {b}")
45     return w, b, cost_history, w_history, b_history
46
47 # Robust Loss Function (Task 4)
48 def new_loss_function(x, y, w, b, beta=1):
49     m = len(x)
50     L_theta_beta = 0
51     for i in range(m):
52         f_wb = np.dot(w, x[i]) + b
53         residual = (f_wb - y[i]) ** 2
54         L_theta_beta += residual / (residual + beta ** 2)
55     L_theta_beta /= m
```

```
    return L_theta_beta

# MSE and BCE Calculation (Task 2.1)
data = pd.read_excel('q2.xlsx')
y = data['y']
yhat = data['yhat']
# Calculate per-sample MSE values
data['mse'] = (y - yhat) ** 2
# Calculate per-sample BCE values
epsilon = 1e-15
yhat_clipped = np.clip(yhat, epsilon, 1 - epsilon)
data['bce'] = -(y * np.log(yhat_clipped) + (1 - y) * np.log(1 -
   yhat_clipped))
overall_mse = np.mean((y - yhat) ** 2)
overall_bce = -np.mean(y * np.log(yhat_clipped) + (1 - y) *
   np.log(1 - yhat_clipped))
```

Listing 8: Python code for linear regression and loss calculations

B References

1. Scikit-learn Developers, "Preprocessing data," Scikit-learn Documentation, [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html.

2. M. Elad, *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. New York, NY, USA: Springer, 2010.

3. Scikit-learn Developers, "Linear Regression," Scikit-learn Documentation, [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html#linear-regression.