Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Learning from data and related challenges and linear models for regression

EN3150 Assignment 01

Name: Sadaruwan I.J.M.J
Index: 220542J

Github Link : Learning from data and related challenges and linear models for regression

EN3150 - Pattern Recognition

1 Linear Regression Impact on Outliers

1.1 Definition of Outliers

Outliers are data points that differ significantly from other observations in a dataset. They may be unusually high or low values and can result from variability in data, measurement errors, or experimental anomalies. Outliers can affect the result of data analysis, especially in regression, by skewing the model or influencing statistical conclusions.

1.2 Dataset

You are given a set of data points related to the independent variable (x) and dependent variable (y) in Table 1.

i	x_i	y_i
1	0	20.26
2	1	5.61
3	2	3.14
4	3	-30.00
5	4	-40.00
6	5	-8.13
7	6	-11.73
8	7	-16.08
9	8	-19.95
10	9	-24.03

Table 1: Data set for regression analysis

1.3 Linear Regression Model

Use all data given in Table 1 to find a linear regression model. Plot x, y as a scatter plot and plot your linear regression model in the same scatter plot.

```
import numpy as np
from matplotlib import pyplot as plt

# Data points of the table 1
x = np.array([0,1,2,3,4,5,6,7,8,9])
y = np.array
    ([20.26,5.61,3.14,-30.00,-40.00,-8.13,-11.73,-16.08,-19.95,-24.03],
    dtype=float)

# Fit linear regression model using numpy
a,b = np.polyfit(x, y, 1)      # np.polyfit gives the slope and intercept
    and fits the line
                               # 1 = degree of the fitting polynomial
print(f"Regression line: y = {a:.2f}x + {b:.2f}")

# Predicted values
```

```
y_pred = a * x + b

# Plot scatter and regression line
plt.figure(figsize=(8,6))
plt.scatter(x, y, color="blue", label="Data points")
plt.plot(x, y_pred, color="red", label=f"Regression line: y = {a:.2f}x + {b:.2f}")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Regression on Given Data")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 1: Plot the linear regression model and same scatter plot

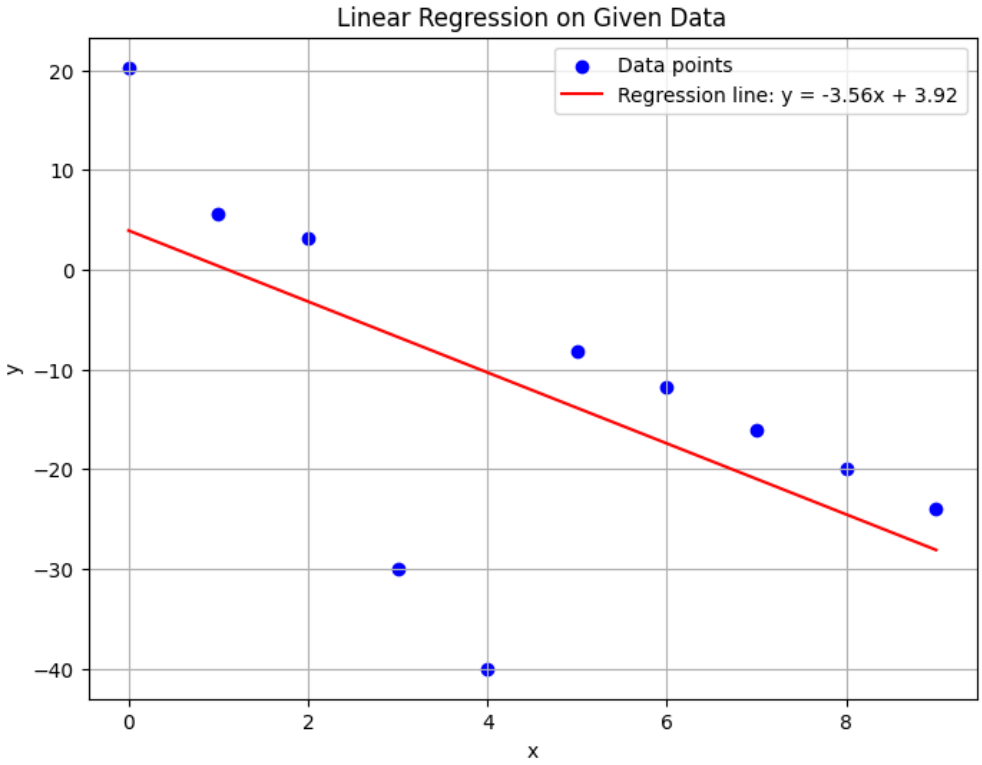

Figure 1: Linear Regression on Given Data

1.4 Loss Functions and Estimators

A **loss function** measures how well a model's predictions match the actual data. It calculates the error between predicted and true values; lower loss means better fit. In regression, common loss functions include mean squared error (MSE).

An **estimator** is an algorithm or method used to find the best model parameters (like slope and intercept in linear regression) by minimizing the loss function.

1.5 Robust Estimator

A robust estimator is introduced to reduce the impact of outliers. The robust estimator finds model parameters which minimize the following loss function:

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

where:

- y_i is the actual value
- $\hat{y}_i$ is the predicted value
- β is a constant that controls the robustness
- N is the number of data points

1.6 Model Comparison

You are given two linear models as follows:

- **Model 1:** $y = -4x + 12$
- **Model 2:** $y = -3.55x + 3.91$

For the two models, calculate the loss function $L(\theta, \beta)$ for all data samples for the following values of β:

- $\beta = 1$
- $\beta = 10^{-6}$
- $\beta = 10^{3}$

Interpretation of Results:

```

def loss_value(y_true, y_pred, beta):
    if (len(y_true) != len(y_pred)):
        raise ValueError("y_true and y_pred must have the same length")
    else:
        term1 = (y_true - y_pred) ** 2
        loss = np.mean(term1 / (term1 + beta**2))
        return loss

# models
model1 = -4*x + 12
model2 = -3.56*x + 3.92

# try different beta values
beta_array = [1.0, 1e-6, 1e3]
results = []
```

```
for beta in beta_array:
    loss1 = loss_value(y, model1, beta)
    loss2 = loss_value(y, model2, beta)
    results.append((beta, loss1, loss2))
    print(f"Beta: {beta:g}, Loss1: {loss1:.6f}, Loss2: {loss2:.6f}")
```

Listing 2: Find loss values for each model using both loss

```
Beta: 1, Loss1: 0.435416, Loss2: 0.973387
Beta: 1e-06, Loss1: 1.000000, Loss2: 1.000000
Beta: 1000, Loss1: 0.000227, Loss2: 0.000188
```

Listing 3: Output values

- For very small β ($\beta = 10^{-6}$), the loss function behaves like mean squared error (MSE). All errors, including those from outliers, are counted fully.
- For $\beta = 1$, the loss is lower, especially for Model 1. This shows that the robust loss function starts to reduce the impact of large errors (outliers).
- For very large β ($\beta = 10^{3}$), the loss values are very small for both models. The denominator dominates, so all errors are down-weighted.

1.7 Suitable Beta Value

To effectively reduce the impact of outliers, β should not be too small or too large. A suitable β is typically chosen based on the expected noise or variability in the data, such as the standard deviation of the residuals.

```
# Range of beta values
beta_values = np.logspace(-6, 4, 200)
loss1 = [loss_value(y, model1, beta) for beta in beta_values]
loss2 = [loss_value(y, model2, beta) for beta in beta_values]

# Plot
plt.figure(figsize=(8,6))
plt.plot(beta_values, loss1, label='Model 1: y = -4x + 12')
plt.plot(beta_values, loss2, label='Model 2: y = -3.56x + 3.92')
plt.xscale('log')
plt.xlabel('Beta (log scale)')
plt.ylabel('Loss')
plt.title('Loss vs Beta for Both Models')
plt.legend()
plt.grid(True, which='both', ls='--')
plt.show()
```

Listing 4: Plot loss vs Beta for both models

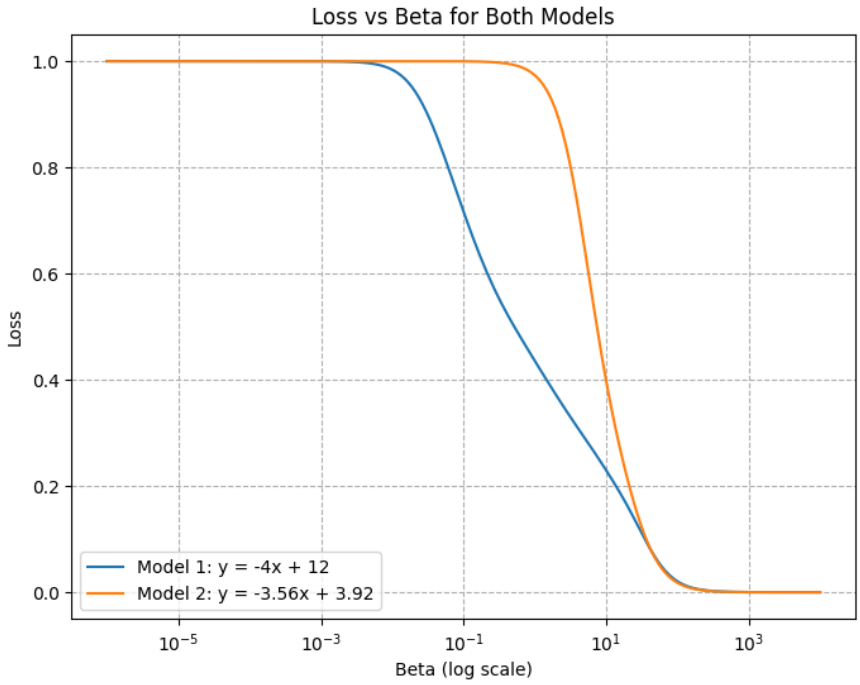

Figure 2: Loss vs Beta for Both Models

Therefore, beta approximately equal to 1 balances sensitivity to normal errors while reducing the effect of extreme outliers.

1.8 Model Selection

Using the robust estimator with $\beta \approx 1$, Model 1 ($y = -4x + 12$) is the most suitable for the provided dataset because its lower error value shows it fits the data better while minimizing the impact of outliers.

```
plt.figure(figsize=(8,6))
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, model1, color='green', linestyle='--', label='Model 1: y = -4x
    + 12')
plt.plot(x, model2, color='red', linestyle='-', label='Model 2: y = -3.56x
    + 3.92')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Regression Lines and Data Points')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 5: plot regrssion line and data points

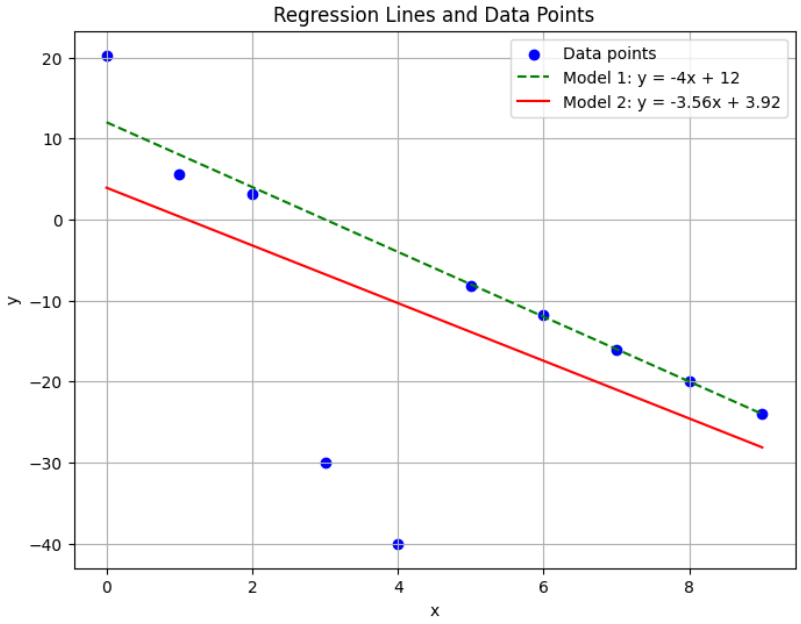

Figure 3: Regression Lines and Data Points

1.9 How Robust Estimator Reduces Outlier Impact

The robust estimator reduces the impact of outliers by bounding the contribution of very large residuals. In ordinary least squares, the squared error grows without limit, so outliers dominate the loss and pull the regression line toward them. In the robust loss, when the residual $(y_i - \hat{y}_i)$ is large compared to β, the fraction

$$\frac{(y_i - \hat{y}_i)^2}{(y_i - \hat{y}_i)^2 + \beta^2} \tag{2}$$

approaches 1, meaning the penalty is capped. Thus, outliers cannot disproportionately affect the parameter estimation.

1.10 Alternative Loss Function: Huber Loss

Another loss function that can be used for robust estimation is the **Huber loss**. It behaves like squared error for small residuals but switches to absolute error for large residuals, maintaining sensitivity for normal data while reducing the effect of outliers.
Mathematically, if we denote the residual as $r = y_i - \hat{y}_i$, then the Huber loss is defined as:

$$L_\delta(r) = \begin{cases} \frac{1}{2}r^2 & \text{if } |r| \le \delta \\ \delta(|r| - \frac{1}{2}\delta) & \text{if } |r| > \delta \end{cases} \tag{3}$$

For small errors ($|r| \le \delta$): the loss is quadratic ($\frac{1}{2}r^2$), same as MSE. For large errors ($|r| > \delta$): the loss grows linearly ($\delta(|r| - \frac{1}{2}\delta)$), reducing the weight of outliers.

2 Loss Functions

Suppose you have two applications:

- **Application 1:** The dependent variable is continuous.
- **Application 2:** The dependent variable is discrete and binary (only takes values 0 or 1, i.e., $y \in \{0, 1\}$).

You plan to train:

- A **Linear Regression** model for Application 1.
- A **Logistic Regression** model for Application 2.

Two common loss functions are:

- **Mean Squared Error (MSE):**

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4}$$

- **Binary Cross Entropy (BCE):**

$$\text{BCE} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{5}$$

where:

- y_i is the true value
- $\hat{y}_i$ is the predicted value
- n is the number of samples

MSE is used for regression tasks with continuous targets, while BCE is used for classification tasks with binary targets.

2.1 Fill the Table and Plot Loss Functions

Table 2: MSE and BCE loss values for different predictions when $y = 1$

True y	Prediction $\hat{y}$	MSE	BCE
1	0.005	0.990025	5.298317
1	0.01	0.9801	4.605170
1	0.05	0.9025	2.995732
1	0.1	0.81	2.302585
1	0.2	0.64	1.609438
1	0.3	0.49	1.203973
1	0.4	0.36	0.916291
1	0.5	0.25	0.693147
1	0.6	0.16	0.510826
1	0.7	0.09	0.356675
1	0.8	0.04	0.223144
1	0.9	0.01	0.105361
1	1.0	0.0	0.0

Table 2: MSE and BCE loss values for different predictions when $y = 1$

Fill in the MSE and BCE values for each prediction. Then, plot both loss functions (MSE and BCE) as a function of the predicted value $\hat{y}$ for $y = 1$.

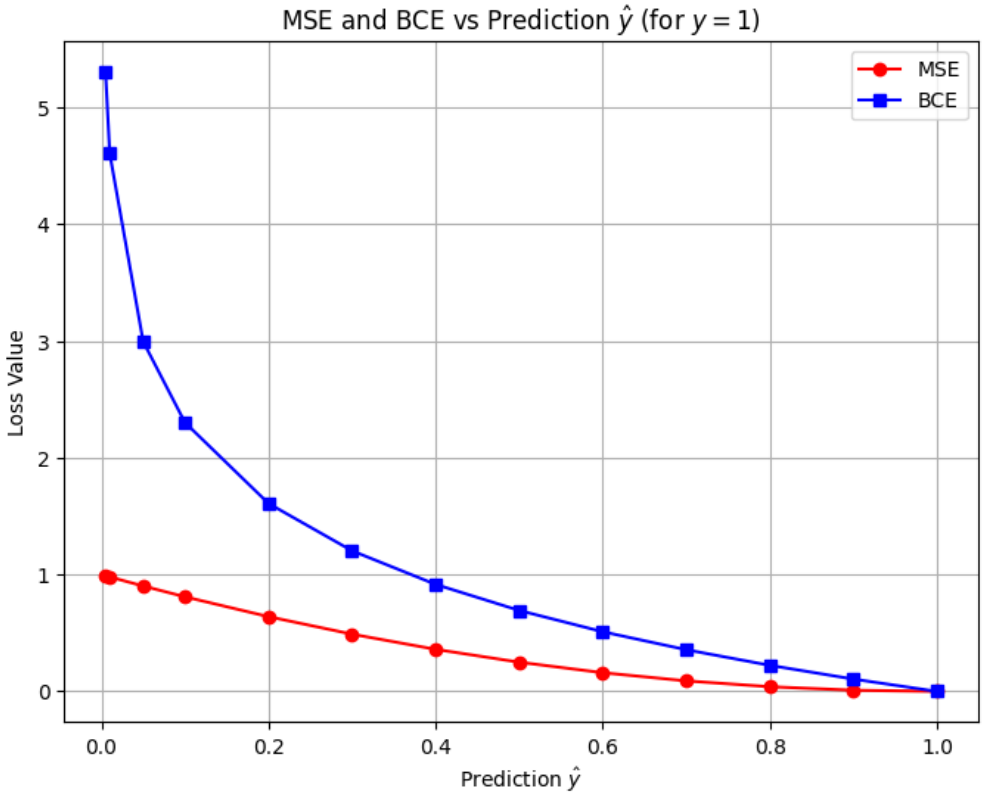

Figure 4: MSE and BCE vs Prediction $\hat{y}$ (for $y = 1$)

2.2 Loss Function Selection

Which loss function (MSE or BCE) would you select for each of the applications (Application 1 and 2)? Justify your answer.

- For Application 1, the dependent variable is continuous, so MSE is the suitable loss function since it directly measures the average squared difference between predicted and actual continuous values.
- For Application 2, the dependent variable is binary, so BCE is the suitable loss function because it measures how well the predicted probabilities match the true binary labels, and it penalizes incorrect confident predictions more effectively than MSE.

3 Data Pre-processing

3.1 Feature Generation

Generate feature values for two features using the code provided in Listing 1. Consider the following scaling methods:

- (a) Standard scaling
- (b) Min-max scaling
- (c) Max-abs scaling

Select one scaling method for each feature, ensuring that the chosen method preserves the structure and properties of the features. Justify your answer.

3.2 Feature Generation Code

```
import numpy as np
import matplotlib.pyplot as plt

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

signal_length = 100 # Total length of the signal
num_nonzero = 10 # Number of non-zero elements in the signal
your_index_no = 220542

sparse_signal = generate_signal(signal_length, num_nonzero)
sparse_signal[10] = (your_index_no % 10) * 2 + 10
if your_index_no % 10 == 0:
    sparse_signal[10] = np.random.randn(1) + 30
sparse_signal = sparse_signal / 5
epsilon = np.random.normal(0, 15, signal_length)
#epsilon = epsilon[:, np.newaxis]

plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)
plt.xlim(0, signal_length)
plt.title("Feature 1", fontsize=18)
plt.xticks(fontsize=18) # Adjust x-axis tick label font size
plt.yticks(fontsize=18)
plt.stem(sparse_signal)
plt.subplot(2, 1, 2)
plt.xlim(0, signal_length)
plt.title("Feature 2", fontsize=18)
plt.stem(epsilon)
```

```
plt.xticks(fontsize=18) # Adjust x-axis tick label font size
plt.yticks(fontsize=18)
plt.show()
```

Listing 6: Feature Generation Code

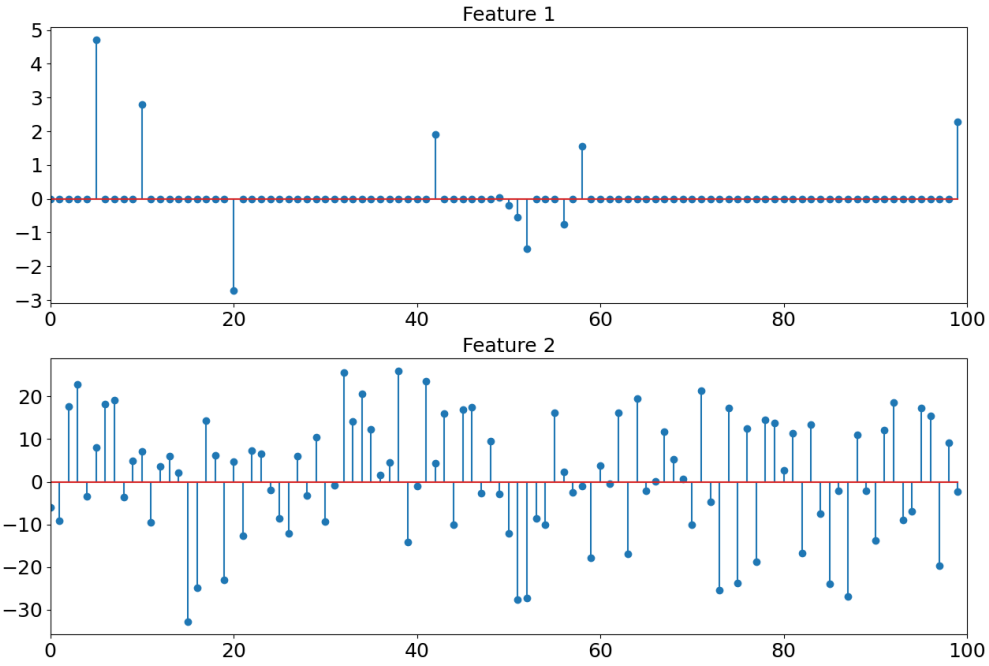

Figure 5: Generated Features - Feature 1 (sparse signal) and Feature 2 (Gaussian noise)

3.3 Scaling Method Selection and Justification

Feature 1 (sparse_signal): Use Max-Abs Scaling

- The signal is sparse (mostly zeros with a few spikes).
- Max-abs scaling preserves zero values and rescales all non-zero values into the range $[-1, 1]$.
- This maintains the sparse structure while normalizing the scale of the spikes.

Feature 2 (epsilon): Use Standard Scaling

- This feature is Gaussian noise centered around zero.
- Standard scaling transforms the data to have mean $= 0$ and standard deviation $= 1$.
- This matches the natural distribution of the noise and ensures balanced feature contribution during training.

3.4 Max-Abs Scaling for Feature 1

```
from sklearn.preprocessing import MaxAbsScaler

# Apply Max-abs scaling to Feature 1 (sparse_signal)
scaler = MaxAbsScaler()
sparse_scaled = scaler.fit_transform(sparse_signal.reshape(-1, 1)).flatten
    ()

plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.title("Sparse Signal (Original)")
plt.stem(sparse_signal)
plt.xlabel("Index")
plt.ylabel("Value")

plt.subplot(1, 2, 2)
plt.title("Sparse Signal (Max-Abs Scaled)")
plt.stem(sparse_scaled)
plt.xlabel("Index")
plt.ylabel("Scaled Value")

plt.tight_layout()
plt.show()
```

Listing 7: Max-Abs Scaling for Feature 1

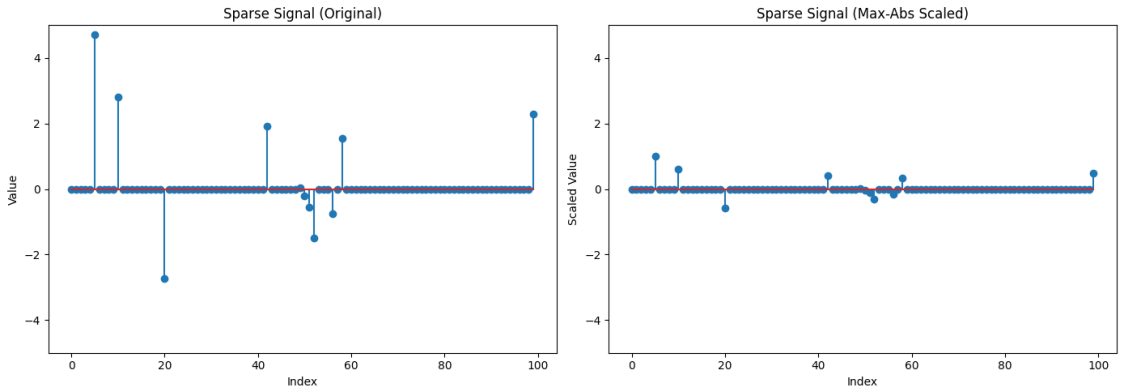

Figure 6: Feature 1: Original vs Max-Abs Scaled Sparse Signal

3.5 Standard Scaling for Feature 2

```
from sklearn.preprocessing import StandardScaler

# Apply Standard scaling to Feature 2 (epsilon)
scaler2 = StandardScaler()
epsilon_scaled = scaler2.fit_transform(epsilon.reshape(-1, 1)).flatten()

# plot original vs scaled epsilon side by side
```

```
plt.figure(figsize=(14, 5))

# Original epsilon
plt.subplot(1, 2, 1)
plt.title("Feature 2 (Original Gaussian Noise)", fontsize=14)
plt.stem(epsilon)
plt.ylim(-35, 30)
plt.xlabel("Index")
plt.ylabel("Value")

# Scaled epsilon
plt.subplot(1, 2, 2)
plt.title("Feature 2 (Standard Scaled)", fontsize=14)
plt.stem(epsilon_scaled)
plt.ylim(-5, 5)
plt.xlabel("Index")
plt.ylabel("Scaled Value")

plt.tight_layout()
plt.show()
```

Listing 8: Standard Scaling for Feature 2

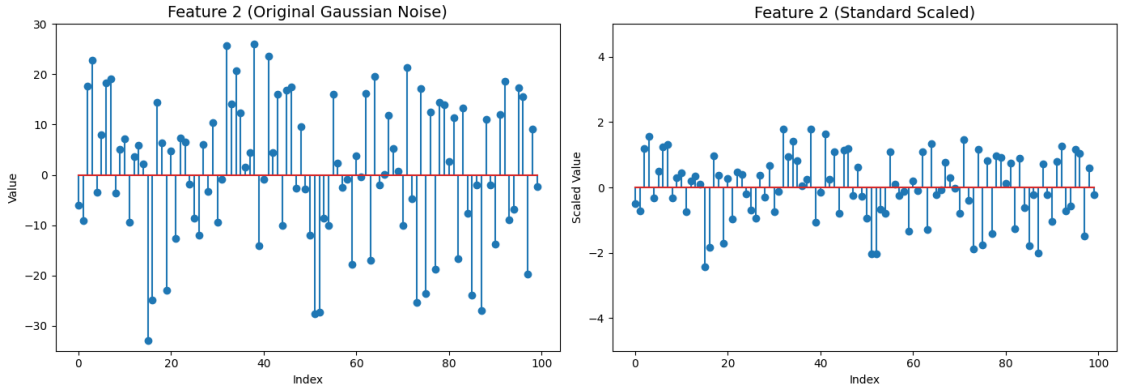

Figure 7: Feature 2: Original vs Standard Scaled Gaussian Noise

3.6 Summary

The choice of scaling methods is crucial for preserving the inherent properties of each feature:

- Max-abs scaling for the sparse signal maintains its sparsity while normalizing the range.
- Standard scaling for the Gaussian noise preserves its statistical properties while ensuring unit variance.