University of Moratuwa

Faculty of Engineering

Department of Electronic & Telecommunication Engineering

Assignment 01 - Learning from Data and Linear Models for Regression

B.Sc. Engineering, Semester 05

Module	EN3150 Pattern Recognition
Student	Ahmed Munavvar M. A
Index No.	220405T
Date	August 13, 2025

Contents

1 Linear Regression Impact on Outliers

Task 2: Linear Regression Model (Code / Scatter plot)

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt

        # Data from Table 1
        x = np.array([0,1,2,3,4,5,6,7,8,9], dtype=float)
        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)

        # Ordinary least squares line: y = m*x + c
        A = np.vstack([x, np.ones_like(x)]).T
        m, c = np.linalg.lstsq(A, y, rcond=None)[0]
        print(f"Learned model (Task 2): y = {m:.2f}x + {c:.2f}")

        # Plot
        plt.scatter(x, y, label="data")
        xx = np.linspace(x.min(), x.max(), 200)
        plt.plot(xx, m*xx + c, label=f"OLS fit: y={m:.2f}x+{c:.2f}")
        plt.xlabel("x"); plt.ylabel("y"); plt.legend(); plt.title("Linear regression on given data")
        plt.show()

        Learned model (Task 2): y = -3.56x + 3.92
```

Figure 1: The code [5] [3]

Learned model: $\hat{y} = -3.56x + 3.92$

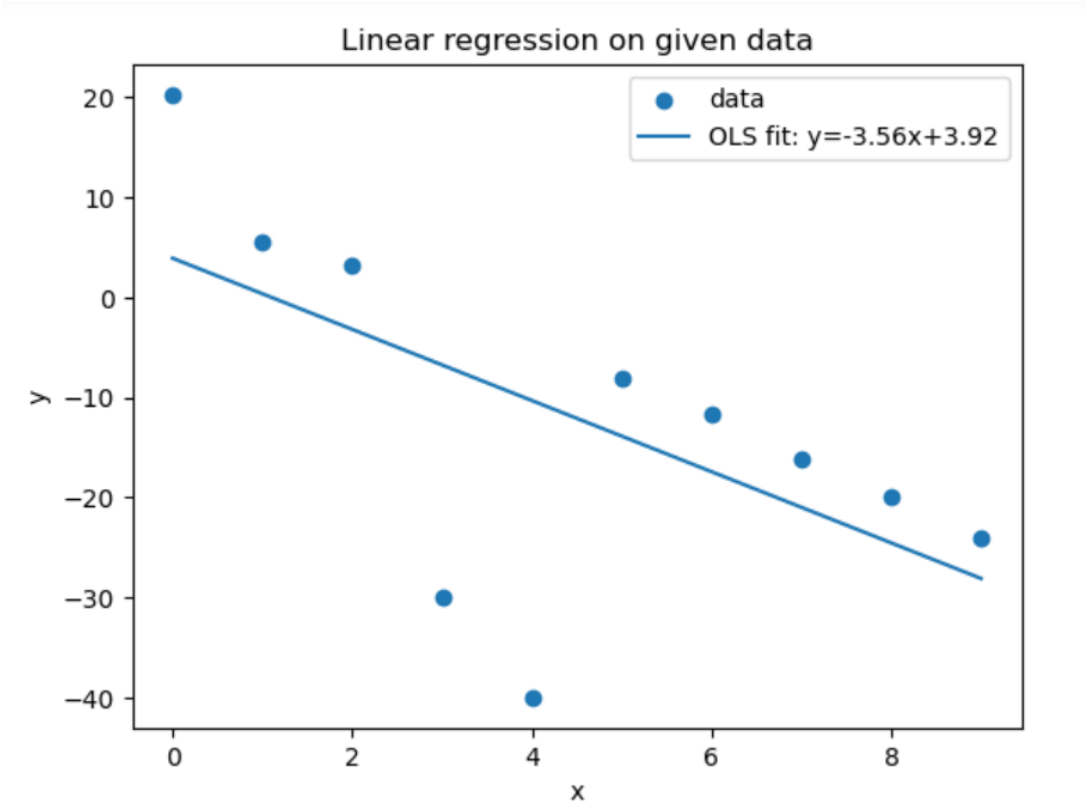

Figure 2: Scatter of (x, y) with learned OLS line (cf. least-squares fit [5]).

Task 3: Given Models

Model 1: $y = -4x + 12$; Model 2: $y = -3.55x + 3.91$ ($\sim y = -3.56x + 3.92$, from the code)

Task 4: Robust Estimator Results

Robust loss (cf. standard texts [5]):

$$L(\theta, \beta) = \frac{1}{N} \sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{(y_i - \hat{y}_i)^2 + \beta^2}, \quad N = 10.$$

Computed for $\beta \in \{1,\ 10^{-6},\ 10^{3}\}$ using a code and filled the below Table 1.

```
In [2]: def robust_loss(y_true, y_pred, beta):
            e2 = (y_true - y_pred)**2
            return np.mean(e2 / (e2 + beta**2))

        # Models
        yhat_m1 = -4*x + 12
        yhat_m2 = m*x + c  # m, c from Task 2

        # Beta values to test
        betas = [1.0, 1e-6, 1e3]
        results = []

        for beta in betas:
            L1 = robust_loss(y, yhat_m1, beta)
            L2 = robust_loss(y, yhat_m2, beta)
            results.append((beta, L1, L2))
            print(f"beta={beta:g}  L(Model1)={L1:.6f}  L(Model2)={L2:.6f}")

        beta=1  L(Model1)=0.435416  L(Model2)=0.973247
        beta=1e-06  L(Model1)=1.000000  L(Model2)=1.000000
        beta=1000  L(Model1)=0.000227  L(Model2)=0.000188
```

Figure 3: The code used to compute $L(\theta, \beta)$ [5]

Table 1: Robust loss $L(\theta, \beta)$ for the two models.

Model	$\beta = 1$	$\beta = 10^{-6}$	$\beta = 10^{3}$
Model 1	0.435416	1.000000	0.000227
Model 2	0.973247	1.000000	0.000188

Note: When β is very small (10^{-6}), almost every nonzero error is counted as the maximum value (≈ 1), so both models give the same loss. When β is very large (10^{3}), the loss behaves almost like the mean squared error (MSE) but divided by a big number, so the values are very small. When $\beta = 1$, large errors are capped near 1 and small errors still contribute proportionally, which reduces the effect of outliers while keeping normal errors important [5].

Task 5: Suitable β to Mitigate Outliers (Justification)

The choice of β controls how the robust loss treats different error sizes:

1. **Purpose of β:**
   - Acts as a threshold between "small" and "large" errors.
   - Determines the point where the loss starts to cap the influence of errors [5].
2. **Reason for choosing $\beta \approx 1$:**
   - It is close to the size of the typical residuals for inlier points in this dataset.
   - Ensures that most normal data points contribute proportionally to the loss.

- Very large residuals from outliers are capped near 1 and cannot dominate [5].

3. **Effect of extreme β values:**
   - If β is too small ($\beta \to 0$): almost all points are treated as outliers, giving each a contribution ≈ 1.
   - If β is too large: loss behaves like MSE, and large errors still have a strong influence [5].

Therefore, $\beta \approx 1$ balances sensitivity to normal errors while reducing the effect of extreme outliers.

Task 6: Most Suitable Model Under the Robust Estimator

The robust loss values from Table 1 with $\beta = 1$ are:

- Model 1: $L_{\text{Model 1}} \approx 0.435$
- Model 2: $L_{\text{Model 2}} \approx 0.973$

1. **Model 1:**
   - Fits the majority of points with small residuals.
   - Only two points have very large residuals, which are capped at ≈ 1 in the robust loss.
2. **Model 2:**
   - Produces moderate residuals for many points.
   - Since many residuals are close to or above β, their contributions to the loss are higher.

Conclusion: With $\beta = 1$, Model 1 yields a much lower robust loss and is therefore the more suitable model for this dataset under the robust estimator [5].

Task 7: How the Robust Estimator Reduces Outlier Impact

The robust loss function for each data point is:

$$\frac{e^2}{e^2 + \beta^2}, \quad \text{where} \quad e = y_i - \hat{y}_i.$$

Its behaviour can be understood in two main cases:

1. **Small errors ($|e| \ll \beta$):**
   - The denominator is dominated by β^2.
   - The term becomes approximately e^2/β^2.
   - Small residuals contribute very little to the overall loss.
2. **Large errors ($|e| \gg \beta$):**
   - The denominator is dominated by e^2.
   - The fraction approaches 1, no matter how large $|e|$ becomes.

- Each large outlier contributes at most 1 to the loss.

This "saturation" effect means that a few extreme outliers cannot dominate the total loss value, while normal data points with small residuals still influence the evaluation meaningfully [5].

Task 8: Another Robust Loss Function

One common alternative robust loss is the **Huber Loss** [4]. Its behaviour is:

1. **Definition:**

$$L_\delta(e) = \begin{cases} \frac{1}{2}e^2, & \text{if } |e| \le \delta, \\ \delta\left(|e| - \frac{1}{2}\delta\right), & \text{if } |e| > \delta \end{cases}$$

   where e is the residual $(y_i - \hat{y}_i)$ and δ is a threshold parameter.

2. **Behaviour:**
   - For small residuals ($|e| \le \delta$), it is *quadratic* like MSE—small errors are penalised to improve accuracy.
   - For large residuals ($|e| > \delta$), it becomes *linear*, reducing the influence of outliers compared to MSE.

3. **Advantages:**
   - Smoothly transitions between sensitivity to normal errors and robustness against outliers.
   - Widely used in regression and computer vision due to its balance between accuracy and robustness [5].

2 Loss Function

2.1 Answer for *"Fill the following table and plot both loss functions."*

Given: $y = 1$, predictions $\hat{y} \in \{0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$.

Equations (standard):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\, y_i - \hat{y}_i\,)^2 \quad \Longrightarrow \quad \text{(here, with one value) } \text{MSE} = (1-\hat{y})^2$$

$$\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right] \quad \Longrightarrow \quad \text{(for } y=1\text{) } \text{BCE} = -\log(\hat{y})$$

[5, 6]

Results

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	0.010	0.980100	4.605170
1	0.050	0.902500	2.995732
1	0.100	0.810000	2.302585
1	0.200	0.640000	1.609438
1	0.300	0.490000	1.203973
1	0.400	0.360000	0.916291
1	0.500	0.250000	0.693147
1	0.600	0.160000	0.510826
1	0.700	0.090000	0.356675
1	0.800	0.040000	0.223144
1	0.900	0.010000	0.105361
1	1.000	0.000000	0.000000

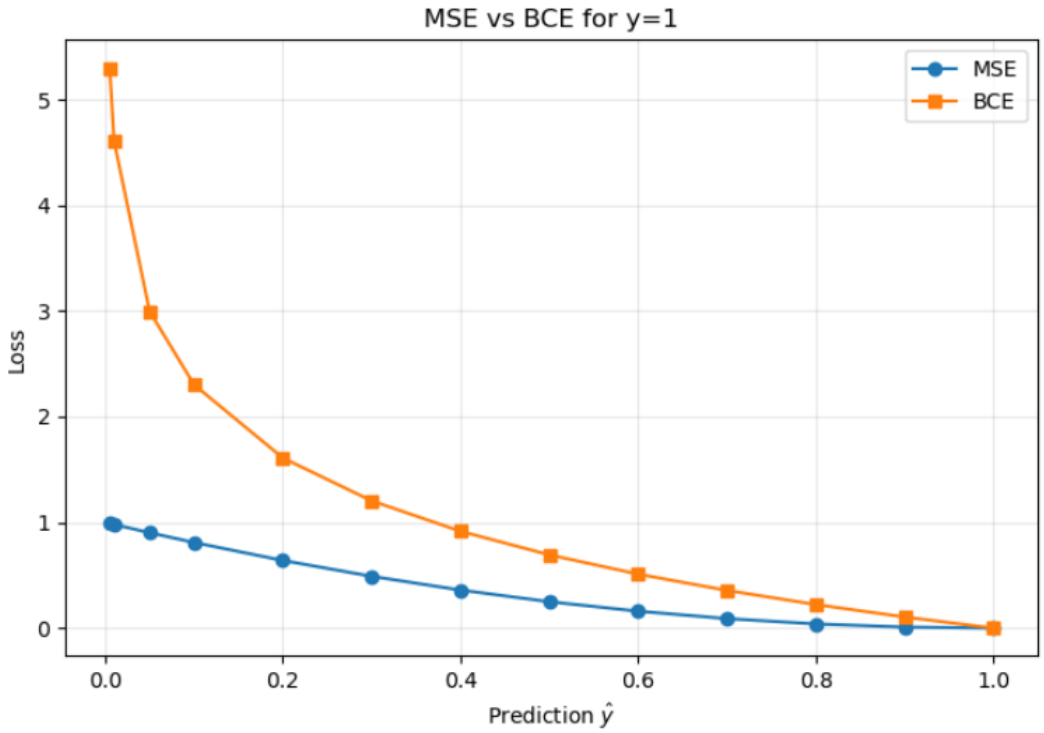

Figure 4: Plot of MSE and BCE versus $\hat{y}$ for $y = 1$ [6].

In [4]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# True y
y_true = 1.0

# Predictions from the assignment table
preds = np.array([0.005, 0.01, 0.05, 0.1, 0.2,
                  0.3, 0.4, 0.5, 0.6, 0.7,
                  0.8, 0.9, 1.0])

# Mean Squared Error
MSE = (y_true - preds)**2

# Binary Cross Entropy for y=1
BCE = -np.log(np.clip(preds, 1e-12, 1.0))  # clip to avoid log(0)

# Create table
df = pd.DataFrame({
    'True y': y_true,
    'Prediction y_hat': preds,
    'MSE': MSE,
    'BCE': BCE
})

print(df.to_string(index=False))

# Save table as CSV if needed
df.to_csv("mse_bce_values.csv", index=False)

# Plot
plt.figure(figsize=(7,5))
plt.plot(preds, MSE, marker='o', label='MSE')
plt.plot(preds, BCE, marker='s', label='BCE')
plt.xlabel("Prediction $\hat{y}$")
plt.ylabel("Loss")
plt.title("MSE vs BCE for y=1")
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("mse_bce_plot.png", dpi=200, bbox_inches="tight")
plt.show()
```

```
 True y  Prediction y_hat      MSE       BCE
    1.0             0.005 0.990025  5.298317
    1.0             0.010 0.980100  4.605170
    1.0             0.050 0.902500  2.995732
    1.0             0.100 0.810000  2.302585
    1.0             0.200 0.640000  1.609438
    1.0             0.300 0.490000  1.203973
    1.0             0.400 0.360000  0.916291
    1.0             0.500 0.250000  0.693147
    1.0             0.600 0.160000  0.510826
    1.0             0.700 0.090000  0.356675
    1.0             0.800 0.040000  0.223144
    1.0             0.900 0.010000  0.105361
    1.0             1.000 0.000000 -0.000000
```

Figure 5: Code used to compute the values in Table 2.

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

- **Application 1 (continuous output): MSE**. Standard for real-valued regression targets [5].
- **Application 2 (binary output / probabilities): BCE**. Properly penalises wrong, confident probabilities (e.g., $\hat{y} \to 0$ when $y = 1$) [5, 6].

3 Data Pre-processing

3.1 Feature Generation

Feature 1 is a sparse signal of length 100 with a few non-zero spikes; the entry at index 10 is set based on my index number 220405T and then divided by 5 (see general background on sparse signals [2]).
Feature 2 is dense Gaussian noise $\epsilon \sim \mathcal{N}(0, 15^2)$ of length 100.

These features were generated in Jupyter Notebook using the code in Figure 6, with the resulting plots shown in Figure 7.

```
In [6]: 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

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

        your_index_no = 220405  # <-- Replace with your index number (no letters, no leading zeros)

        # Generate Feature 1 (sparse signal)
        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

        # Generate Feature 2 (Gaussian noise)
        epsilon = np.random.normal(0, 15, signal_length)
        # epsilon = epsilon[:, np.newaxis]  # Optional reshape if needed

        # Plot
        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)
        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)
        plt.yticks(fontsize=18)

        plt.show()
```

Figure 6: Jupyter Notebook cell used to generate Feature 1 (sparse) and Feature 2 (Gaussian).

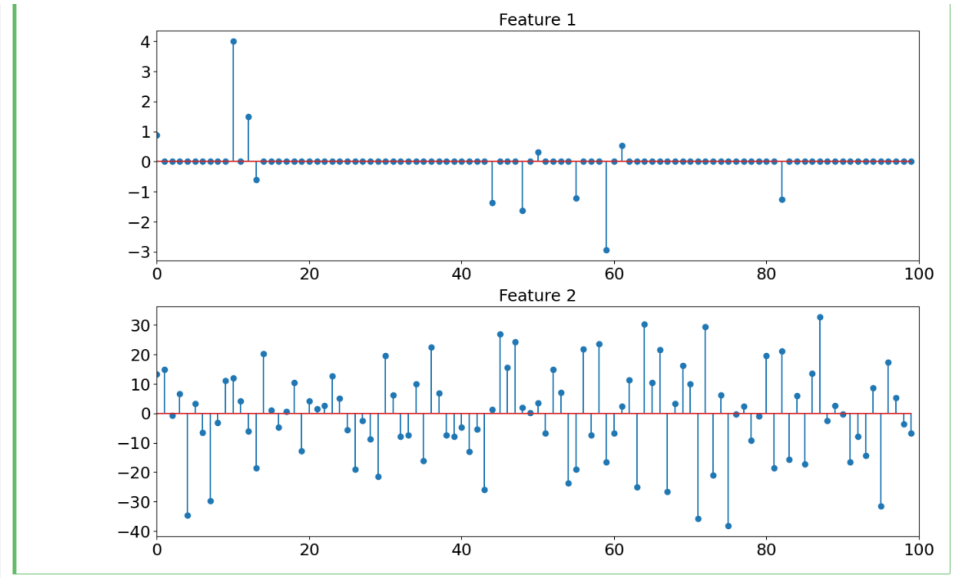

Figure 7: Generated features: top - Feature 1 (sparse), bottom - Feature 2 (Gaussian).

3.2 Scaling and Justification

Feature 1 (sparse signal): Max-Abs scaling

- Preserves zeros exactly, maintaining sparsity.
- Applies a single multiplicative factor without mean-centering.
- Retains both positive and negative spikes proportionally.

Feature 2 (Gaussian-like noise): Standard (z-score) scaling

- Centers data to mean 0 and scales to unit variance, suitable for Gaussian-like dense features.
- Improves conditioning for algorithms assuming zero-mean/unit-variance inputs.

These choices are consistent with common practice and scikit-learn's guidance [1, 6].

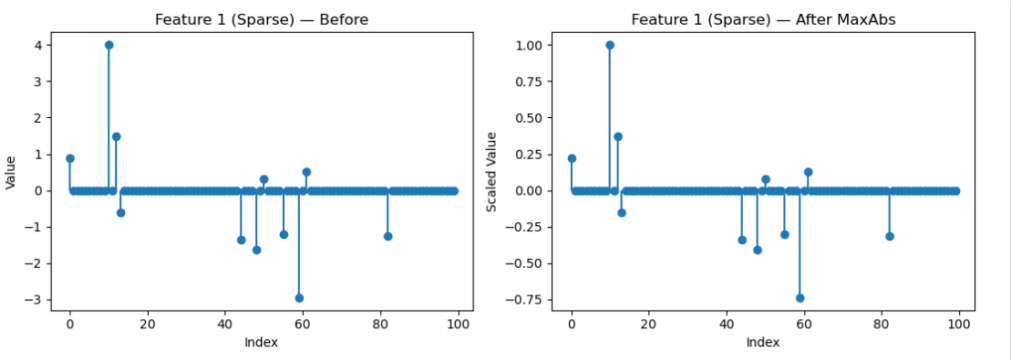

Figure 8: Feature 1: before (left) and after (right) Max-Abs scaling.

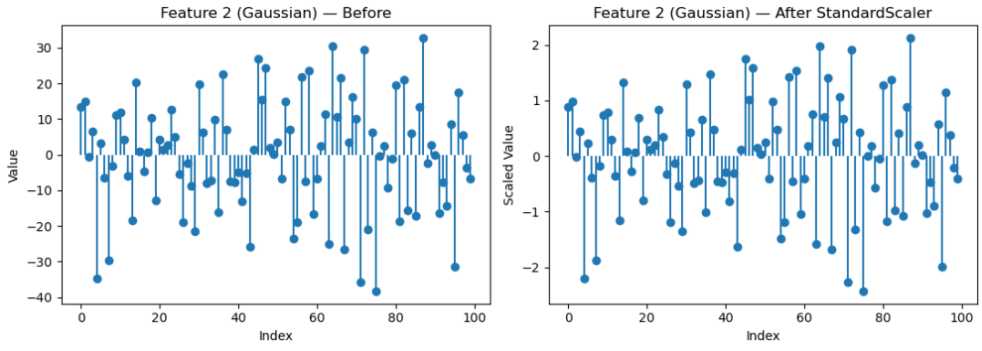

Figure 9: Feature 2: before (left) and after (right) Standard scaling.

References

[1] Scikit-learn Developers. (2025). *Preprocessing data*. Scikit-learn User Guide. Retrieved from https://scikit-learn.org/stable/modules/preprocessing.html

[2] Seiselesni, I. (n.d.). *Introduction to sparsity in signal processing*. New York University. Retrieved from https://eeweb.engineering.nyu.edu/iselesni/lecture_notes/

sparsity_intro/sparse_SP_intro.pdf

[3] Scikit-learn Developers. (2025). *LinearRegression*. Scikit-learn API Reference. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[4] Huber, P. J. (1964). Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1), 73–101. https://doi.org/10.1214/aoms/1177703732

[5] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[6] Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (3rd ed.). O'Reilly Media.