

**Department of Electrical Engineering**

**University of Moratuwa**

**EN 3150 – Pattern Recognition**

**Learning from data and related challenges and linear models for regression**

Kularathna A. K. D. D. D. – 220332P

This is submitted as a partial fulfilment for the module

EN 3150 Pattern Recognition

Department of Electronics and Telecommunication Engineering

University of Moratuwa

1. **Linear regression impact on outliers**
2. A table with numbers and numbers

   AI-generated content may be incorrect.
3. **Finding the Linear regression model and Plotting x, y as scatter and the linear regression model.**

* Regression Line Equation: -

A graph with a red line and blue dots

AI-generated content may be incorrect.

*Figure 1.1*

*Code for Getting regression line and Plotting*

import numpy as np

import matplotlib.pyplot as plt

# Data points (Table 1: Data Set)

x\_i = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

y\_i = [20.26, 5.61, 3.14, -30.00, -40.00, -8.13, -11.73, -16.08, -19.95, -24.03]

# Create design matrix X (with a column of 1s for intercept) and target vector Y

X = np.array([[1, x] for x in x\_i])

Y = np.array([[y] for y in y\_i])

# print("X =", X)

# print("Y =", Y)

# Apply OLS formula: w = (X^T X)^(-1) X^T Y

X\_T = X.T

w\_OLS = np.linalg.inv(X\_T @ X) @ X\_T @ Y

# Extract coefficients

intercept = w\_OLS[0, 0]

slope = w\_OLS[1, 0]

# Predict y values using the regression model

x\_range = np.linspace(min(x\_i), max(x\_i), 100)

y\_predicted = intercept + slope \* x\_range

# Step 4: Plot original data and regression line

plt.figure(figsize=(10, 6))

plt.scatter(x\_i, y\_i, color='blue', label='Data points')

plt.plot(x\_range, y\_predicted, color='red', label=f'Regression line: y = {intercept:.3f} + {slope:.3f}x')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Linear Regression Fit')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

print(f"Regression Line Equation: y = {intercept:.3f} + {slope:.3f}x")

1. A mathematical equation with numbers and symbols

   AI-generated content may be incorrect.
2. **Calculation Loss Function Values for given β values**

**A graph with a red line and blue dots

AI-generated content may be incorrect.**

Figure 1.2

* **β = 1**

Model 1 loss = 0.4354

Model 2 loss = 0.9728

* **β = 10-6**

Model 1 loss = 1.0000

Model 2 loss = 1.0000

* **β = 103**

Model 1 loss = 0.0002

Model 2 loss = 0.0002

A graph with blue and orange dots

AI-generated content may be incorrect.

Figure 1.3

*Code for Calculation of β values and plotting*

# Compare with the model y = -4x + 12

y\_model = lambda x: -4 \* x + 12

# Plot the model line

x\_model = np.linspace(min(x\_i), max(x\_i), 100)

y\_model\_values = y\_model(x\_model)

# Plot the model line

plt.figure(figsize=(10, 6))

plt.plot(x\_model, y\_model\_values, color='green', label='Model: y = -4x + 12', linestyle='--')

# Add the model line to the existing plot

plt.scatter(x\_i, y\_i, color='blue', label='Data points')

plt.plot(x\_range, y\_predicted, color='red', label=f'Regression line: y = {intercept:.3f} + {slope:.3f}x')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Linear Regression Fit with Model Comparison')

plt.legend()

plt.grid(True)

plt.tight\_layout()

def robust\_loss(y\_true, y\_pred, beta):

    N = len(y\_true)

    squared\_errors = (y\_true - y\_pred) \*\* 2

    loss = np.mean(squared\_errors / (squared\_errors + beta\*\*2))

    return loss

# Generate predictions for both models

y\_model1 = -4 \* np.array(x\_i) + 12  # Model 1

y\_model2 = -3.55 \* np.array(x\_i) + 3.91  # Model 2

# Test different beta values

beta\_values = [1, 10\*\*(-6), 10\*\*(3)]

results = []

for beta in beta\_values:

    loss\_model1 = robust\_loss(y\_i, y\_model1, beta)

    loss\_model2 = robust\_loss(y\_i, y\_model2, beta)

    results.append({

        'beta': beta,

        'loss\_model1': loss\_model1,

        'loss\_model2': loss\_model2

    })

# Create a table of results

for result in results:

    print(f"β = {result['beta']:.6f}:")

    print(f"  Model 1 (y = -4x + 12) loss: {result['loss\_model1']:.4f}")

    print(f"  Model 2 (y = -3.55x + 3.91) loss: {result['loss\_model2']:.4f}")

    print()

# Visualize how the loss changes with beta

plt.figure(figsize=(10, 6))

plt.scatter(beta\_values, [r['loss\_model1'] for r in results], marker='o', s=100, label='Model 1 (y = -4x + 12)')

plt.scatter(beta\_values, [r['loss\_model2'] for r in results], marker='s', s=100, label='Model 2 (y = -3.55x + 3.91)')

plt.xlabel('β (log scale)')

plt.ylabel('Robust Loss')

plt.title('Robust Loss vs β for Both Models')

plt.legend()

plt.grid(True)

plt.xscale('log')  # Ensure x-axis is in log scale

plt.tight\_layout()

1. **Suitable β value to mitigate the impact of the outliers**

* When β = 10-6
  + In this case for both models the loss function value becomes really close to 1 since the denominator becomes nearly equal (since β2 term is nearly zero for small values). This makes the loss function **more sensitive to the outliers** in the dataset.
* When β = 1
  + In this case for both models the loss function gives balanced values. This time loss function is little less sensitive to the outliers but not as sensitive as earlier. This allows us to make the more **Robust to outliers** while being sensitive enough to differentiate between the performance of different models on the given data.
* When β = 103
  + In this case for both models the loss function value becomes really close to 0 since the denominator becomes nearly equal to β2. This makes the model **less sensitive to the outliers** in the data set. The low loss values make it **harder to differentiate between the models**, as they both appear to perform well.

**Due to these reasons** β **= 1 is the best value to choose from**

1. **Determining most suitable model from the models**

* Using β = 1 Model 1 (y = -4x + 12) is the better choice because the loss = 0.4354 and the loss = 0.9728 for Model 2 (y = -3.55x + 3.91)
  + The robust loss at β=1 directly measures how well a model explains the data while reducing the effect of outliers. A lower robust loss indicates a better fit under this robust criterion.
  + Numerically Model 1’s loss is substantially lower than Model 2’s at β = 1. Therefore Model 1 better explains the bulk of the data while being less influenced by extreme points, which is exactly the goal when using a robust estimator.

**Therefore, Choose Model 1**

1. **How Robust estimator reduce the impact of the outliers**

A mathematical equations and formulas

AI-generated content may be incorrect.The loss function is:

* **When the error is small compared to β2.**

The fraction is small, so these points contribute only a little to the loss – like normal squared error.

* **When the error is large (from an outlier).**

The denominator grows almost as much as the numerator, and the fraction approaches 1. This means no matter how huge the error is, its contribution is capped at 1.

By doing this, the loss function prevents large errors from dominating the total loss, making the model less affected by outliers. The value of β controls when this “capping” effect starts.

1. **Another loss function that can be used for this robust estimator**

A common alternative is Huber **loss**. It is defined (for residual  ) as:

* Huber is **quadratic for small residuals** (like MSE) and **linear for large residuals** (like absolute error), so it reduces the influence of large errors while staying differentiable and convex.
* The parameter plays a role like β: it sets the threshold between “squared” and “linear” regimes, enabling robustness tuning

1. **Loss Function**