University of Moratuwa

Department of Electronic and Telecommunication Engineering

EN3150

Pattern Recognition

Learning from data and related challenges and linear models for regression

August 21, 2025

220257N Jayasekara S.P.R

1 Question 1

2. 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.

Importing the required libraries

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

Performing linear regression and plotting

```
X = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).reshape(-1, 1)
y = np.array([20.26,5.61,3.14,-30,-40,-8.13,-11.73,-16.08,-19.95,-24.03])
N = 10
# creating and training the model
model = LinearRegression()
model.fit(X, y)

# predicting values
y_pred = model.predict(X)

print(f"Slope: {model.coef_[0]:.2f}")
print(f"Intercept: {model.intercept_:.2f}")

# plotting
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, y_pred, color='red', label='Regression line')
plt.xlabel("X")
plt.ylabel("y")
plt.title("Linear Regression Example")
plt.legend()
plt.show()
```

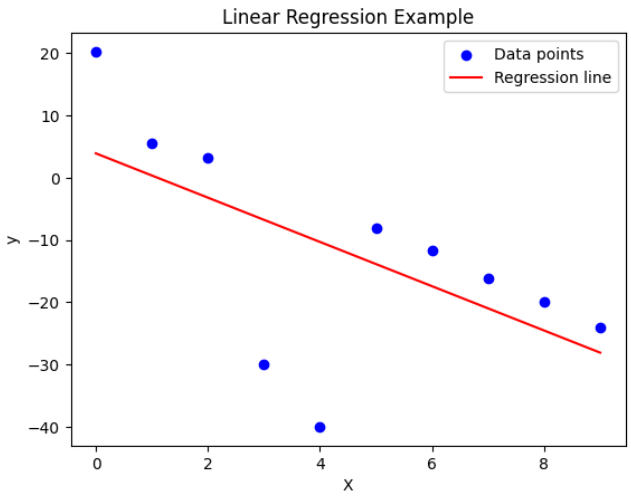

Figure 1: Graph with data and regression model

4.For the given two models in task 3, calculate the loss function $L(\theta, \beta)$ values for all data samples using eq. (1) for $\beta = 1$, $\beta = 10^{-6}$ and $\beta = 10^{3}$

```
beta = [1,1e-6,1e3]
X = X.flatten()

# function to compute loss
def compute_loss(intercept, slope, beta):
    predictions = slope * X + intercept
    residuals = y - predictions
    loss = np.mean(residuals**2 / (residuals**2 + beta**2))
    return loss

#getting the loss for y = -4x + 12
for b in beta:
    loss_calc = compute_loss(12,-4,b)
    print(f'for beta = {b} loss is = {loss_calc}')
print('          ')


#getting the loss for y = -3.55 x + 3.91

for b in beta:
    loss_calc = compute_loss(3.91,-3.55,b)
    print(f'for beta = {b} loss is = {loss_calc}')
```

Results:

Model 1

for $\beta = 1$ loss = 0.4354162624903386
for $\beta = 10^{-6}$ loss = 0.9999999998258296
for $\beta = 1000.0$ loss = 0.0002268287498440988

Model 2

for $\beta = 1$ loss = 0.9728470518681676
for $\beta = 10^{-6}$ loss = 0.9999999999999718
for $\beta = 1000.0$ loss = 0.0001882468465464654

5. What is the suitable β value to mitigate the impact of the outliers. Justify your answer.

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

The purpose of this loss function is to pull back bigger losses and keep all losses to be between 0 and 1. When the beta value is too large the denominator gets dominated by the β^2 term. This causes the overall value to be extremely low as demonstrated by the above results. Therefore it is harder to differentiate between inliers and outliers. When beta value is too small, the overall term approaches zero. This is also clear from above results. Therefore, the most appropriate beta value is 1. It properly weighs down outliers.

6. Utilizing this robust estimator with selected β value, determine the most suitable model from the models specified in task 3 for the provided dataset. Justify your selection.

For $\beta = 1$, the average robust loss values are:

- Model 1: $y = -4x + 12 \rightarrow$ average robust loss ≈ 0.435
- Model 2: $y = -3.55x + 3.91 \rightarrow$ average robust loss ≈ 0.973

Model Choice: From above results we can see that the model 1 ($y = -4x + 12$) is more suitable for the dataset when using the robust estimator with $\beta = 1$. It achieves a significantly lower loss value compared to the model 2 (0.435416 vs 0.972847).

7. How does this robust estimator reduce the impact of the outliers?

According to the equation of the loss function, the overall value always falls between 0 and 1 regardless of the error. In the traditional MSE function, the outliers have high effect due to their high loss being squared. In this function, the outliers cannot dominate the cost.

The gradient for outliers is as follows,

$$\frac{dl}{dr} = \frac{2r\beta^2}{(r^2 + \beta^2)^2} \tag{2}$$

It can be seen that as the r term increases, the gradient value decreases due to r^4 term in the denominator. This helps mitigate the effect of outliers.

8. Identify another loss function that can be used for this robust estimator.

Cauchy Loss Function

Definition

For a prediction error $r = y - \hat{y}$, the Cauchy loss is defined as:

$$L(r) = \frac{c^2}{2} \log\left(1 + \frac{r^2}{c^2}\right)$$

where c is a scaling parameter that determines how quickly the loss saturates.

Behavior for Small and Large Errors

Small errors ($|r| \ll c$):

$$\log\left(1 + \frac{r^2}{c^2}\right) \approx \frac{r^2}{c^2} \quad \Rightarrow \quad L(r) \sim \frac{r^2}{2}$$

This behaves like Mean Squared Error (MSE), allowing accurate gradient updates for most data points.

Large errors ($|r| \gg c$):

$$\log\left(1 + \frac{r^2}{c^2}\right) \sim \log(r^2)$$

which grows slowly with r. This means that very large errors (outliers) contribute much less to the total loss compared to MSE, which grows quadratically.

Gradient Effect

The derivative of the Cauchy loss with respect to r is:

$$\frac{\partial L}{\partial r} = \frac{r}{c^2 + r^2}$$

For small r:

$$\frac{\partial L}{\partial r} \approx \frac{r}{c^2} \quad \text{(standard gradient like MSE)}$$

For large r:

$$\frac{\partial L}{\partial r} \approx \frac{1}{r} \quad \text{(gradient diminishes, reducing outlier influence)}$$

From above equations, we can see that the cauchy loss functions handles outliers better than the MSE loss function.

2 Question 2

1. Fill the following table and plot the both loss functions.

Code for calculating the 2 loss values.

```
predictions = 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])

y_true = 1

# equation for mse
mse_values = (y_true - predictions) ** 2

# small value to avoid log(0) errors
e = 1e-15

# equation for bce
bce_values = -(y_true * np.log(predictions + e ) + (1 - y_true) * np.
   log(1 - predictions + e))
```

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

True $y = 1$	**Prediction $\hat{y}$**	**MSE**	**BCE**
1	0.005	0.990	5.298
1	0.01	0.980	4.605
1	0.05	0.902	2.996
1	0.1	0.810	2.303
1	0.2	0.640	1.609
1	0.3	0.490	1.204
1	0.4	0.360	0.916
1	0.5	0.250	0.693
1	0.6	0.160	0.511
1	0.7	0.090	0.357
1	0.8	0.040	0.223
1	0.9	0.010	0.105
1	1.0	0.000	0.000

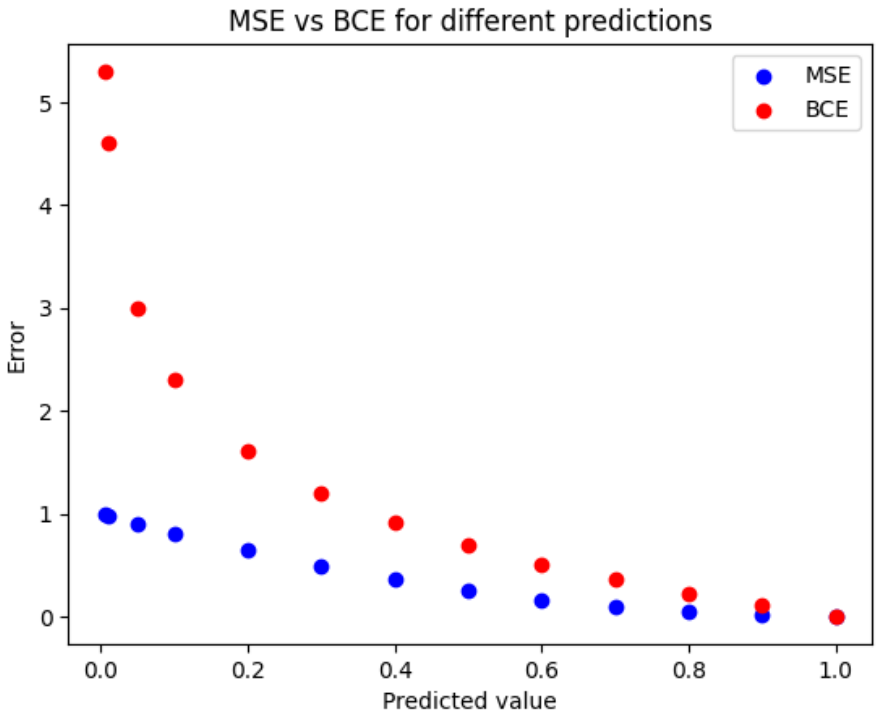

Figure 2: The 2 losses for the given predictions

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

Application 1: Mean Squared Error Loss Function

MSE is ideal for continuous variables because it measures the average of the squared differences between the predicted and actual values. It's a fundamental loss function for regression tasks. By squaring the errors, MSE penalizes larger errors more heavily. This helps the model to make more accurate predictions.

Application 2: Binary Cross Entropy Loss Function

BCE is the appropriate choice for binary classification tasks. This is because it is specifically designed to handle probabilistic outputs (values between 0 and 1). It measures the performance of a classification model whose output is a probability. It is particularly effective at penalizing predictions that are confident and wrong. This can be seen in the results in the table. When the predicted outcome is 0.005, the loss function heavily penalizes this prediction with a loss of 5.298. We can also see this from the graph by how rapidly the loss for BCE climbs when prediction approaches 0.

3 Question 3

Code for scaling using MaxAbsScalar, MinMaxScalar, StandardScalar.

```
# feture 1
sparse_maxabs = MaxAbsScaler().fit_transform(sparse_signal.reshape
    (-1, 1)).flatten()
sparse_minmax = MinMaxScaler().fit_transform(sparse_signal.reshape
    (-1, 1)).flatten()
sparse_standard = StandardScaler().fit_transform(sparse_signal.
    reshape(-1, 1)).flatten()

# feature 2
epsilon_maxabs = MaxAbsScaler().fit_transform(epsilon.reshape(-1, 1))
    .flatten()
epsilon_minmax = MinMaxScaler().fit_transform(epsilon.reshape(-1, 1))
    .flatten()
epsilon_standard = StandardScaler().fit_transform(epsilon.reshape(-1,
     1)).flatten()
```

The equations corresponding to each of the scaling methods is given below

- Standard Scaling: $x_{\text{scaled}} = \dfrac{x - \mu}{\sigma}$
- Min-Max Scaling: $x_{\text{scaled}} = \dfrac{x - x_{\min}}{x_{\max} - x_{\min}}$
- Max-Abs Scaling: $x_{\text{scaled}} = \dfrac{x}{|x_{\max}|}$

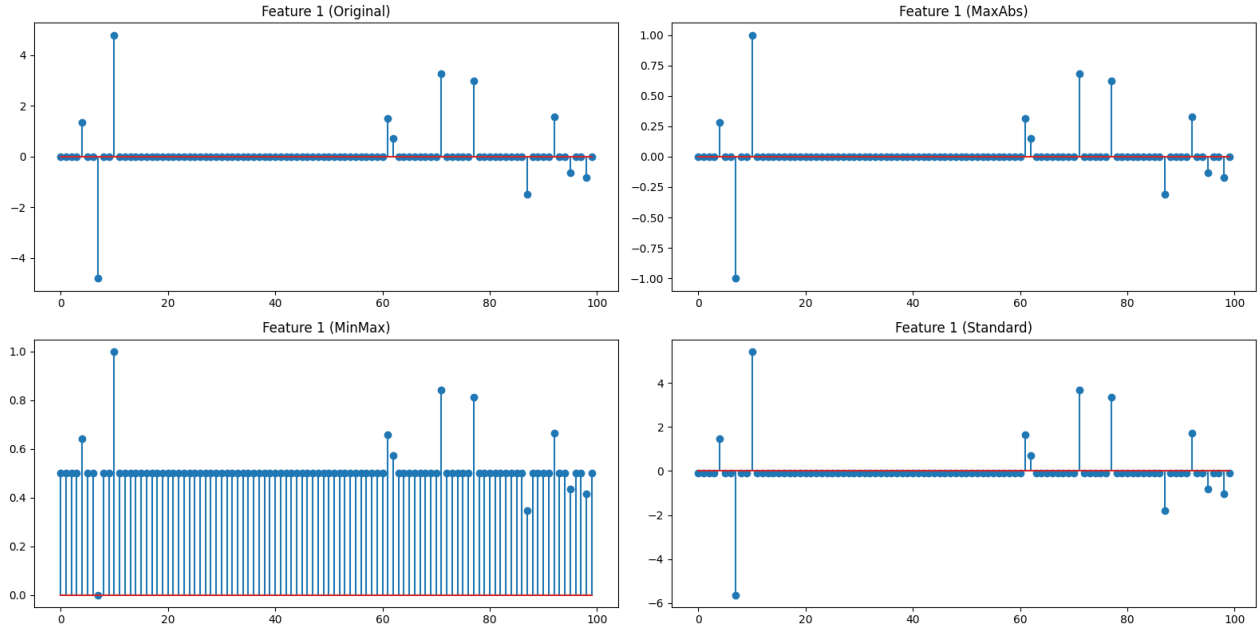

Figure 3: The plots corresponding to feature 1

Scaling for feature 1: **MaxAbs Scaling**

Justification:

- This scaling preserves the sparsity property of the feature. The MinMax Scaling has clearly distorted the signal while a closer look at Standard scaled plot shows that the 0 values have been turned to small negative values.
- MaxAbs scaling has preserved the structure of the signal. The zeroes and non-zero elements has retained their positions and the releative magnitudes.

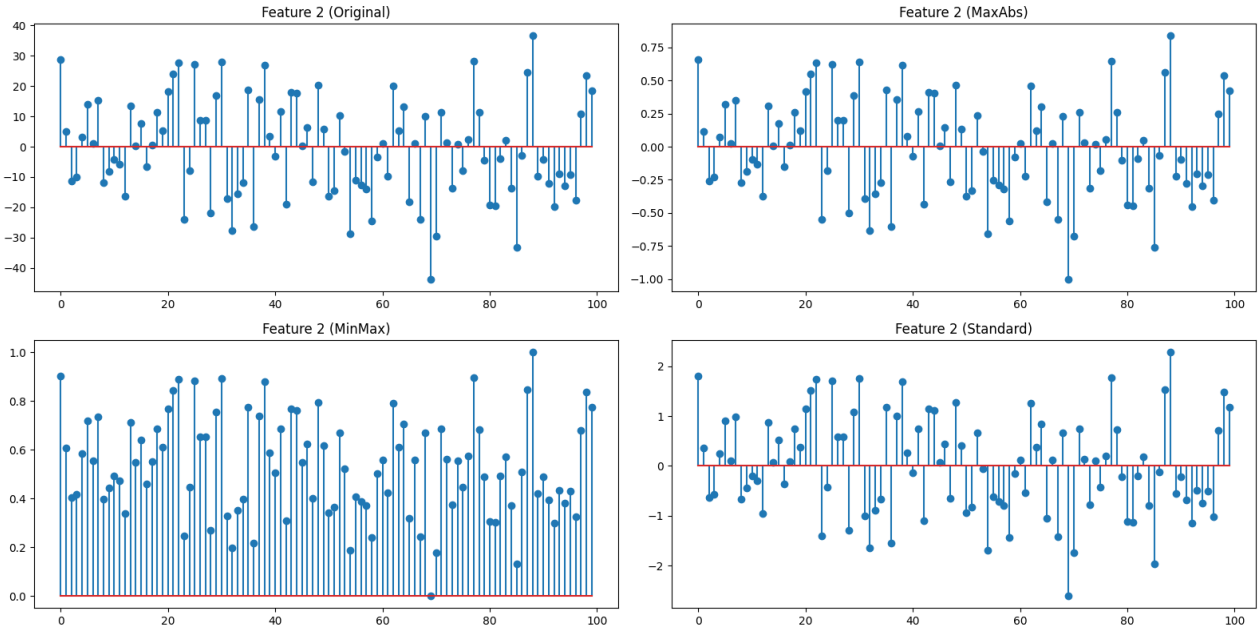

Figure 4: The plots corresponding to feature 2

Scaling for feature 2: **Standard Scaling**

Justification:

- This feature is a noise signal which is approximately Gaussian. Therefore using the Normal Scaling preserves its distribution.
- A lot of algorithms assume standardized features for optimal performance. Therefore this scaling will perform well with those algorithms.