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

Learning from data and related challenges and linear models

Walgampaya H.K.B. - 220675T

EN3150 - Pattern Recognition

Submitted in partial fulfillment of the requirements for the module

Date

21/08/2025

Contents

1 Linear regression impact on outliers

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.

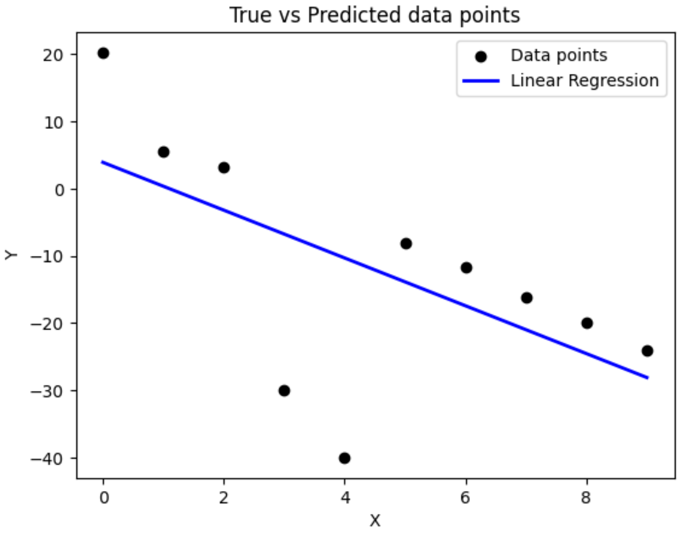
- Method 1 ⇒ Using built-in Functions

```python
x = np.arange(10)
y = np.array([20.26,5.61,3.14,-30.00,-40.00,-8.13,-11.73,-16.08,-19.95,-24.03])
x_reshaped = x.reshape(-1,1)
y_reshaped = y.reshape(-1,1)
model = LinearRegression()
model.fit(x_reshaped, y_reshaped) #fit the model
print("w0 = " + str(model.intercept_))
print("w1 = " + str(model.coef_[0]))

y_predicted = model.predict(x_reshaped) #predit the model based on the weights and data points
plt.scatter(x_reshaped, y_reshaped, color="black", label='Data points')
plt.plot(x_reshaped, y_predicted, color="blue", linewidth=2, label=r'Linear Regression')
```

- Method 2 ⇒ Using the matrix method $\hat{\beta} = (X^T X)^{-1} X^T y$

```python
ones_col = np.ones((x_reshaped.shape[0],1))
matX = np.hstack((ones_col,x_reshaped))
matXT = matX.T
matXTX = np.dot(matXT,matX)
matXTX_inv = np.linalg.inv(matXTX)
matXTy = np.dot(matXT,y_reshaped)
result = np.dot(matXTX_inv,matXTy)
w0 = result[0][0]
w1 = result[1][0]
print("w0 = " + str(result[0]))
print("w1 = " + str(result[1]))
```



Results obtained from both methods are the same. For our linear model, $y = w_0 + w_1 x_1$:

- $w_0 =$ `3.91672727` ≈ 3.91
- $w_1 =$ `-3.55727273` ≈ -3.55

A robust estimator β is used to find the model parameters that reduce the following loss function.

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

4) For the given two models in task 3, calculate the loss function L(θ, β) values for all data samples using eq. (1) for $\beta = 1$, $\beta = 10^{-6}$ and $\beta = 10^3$ (you may use a computer program to calculate this).

```
N = x_reshaped.shape[0]
beta_vals = [1,10**(-6),10**3]
def loss_calc(y_true, y_estimate,beta):
    """This function is used to calculate the loss"""
    squared_diff = (y_true-y_estimate)**2
    return squared_diff/(squared_diff + beta**2)

y_predicted_model_1 = 12+((-4)*x_reshaped)
y_predicted_model_2 = 3.91+((-3.55)*x_reshaped)
#plot
plt.scatter(x_reshaped, y_reshaped, color="black", label=’Data points’)
plt.plot(x_reshaped, y_predicted_model_1, color="blue", linewidth=2, label=r’Model 1’)
plt.plot(x_reshaped, y_predicted_model_2, color="red", linewidth=2, label=r’Model 2’)
```

- **Loss Function for model 1**

```
for beta in beta_vals:
    total = 0
    for i in range(10):
        total += loss_calc(y_reshaped[i][0], y_predicted_model_1[i][0], beta)
    L = total / N
    print(f"for model = 1 and beta = {beta},        L = {L}")
print()
```

- **Loss Function for model 2**

```
for beta in beta_vals:
    total = 0
    for i in range(10):
        total += loss_calc(y_reshaped[i][0], y_predicted_model_2[i][0], beta)
    L = total / N
    print(f"for model = 2 and beta = {beta},        L = {L}")
print()
```

β	**Loss values for model 1**	**Loss values for model 2**
1	0.435416262490386	0.9728470518681676
1^{-6}	0.9999999998258206	0.999999999999718
1000	0.0002268287498440988	0.00018824684654645654

Table 1: Loss Function Values

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

Select $\beta = 1$

Reason:

- When $\beta = 10^{-6}$, it is a very small value so it becomes negligible in the denominator. The loss function can then be approximated to $\frac{error^2}{error^2} = 1$ for outliers and non-outliers with a significant error term. So even though the error from outliers has a upper bound of 1, the non outliers will also get a similar level of error. so the function will not work as expected

- When $\beta = 10^3$, it is relatively large and the error term in the denominator becomes negligible. The loss function can then be approximated to $\frac{error^2}{\beta^2} = 1$. and β is large this will be near zero for outliers as well as non-outliers. The loss function will now be almost flat, and the gradient of the loss function will be very small making it hard for the model to learn.

- When $\beta = 1$ it gives the best results. The reason is that for this β value, when the error term is small, the output of the loss function will also be small and won't go to zero unlike the previous case. Furthermore, when there are outliers, the error term will be large but the output of the loss function will saturate at 1 without giving a very large overwhelming value to the training process. So the loss function will work as expected.

So $\beta = 1$ is most suitable to mitigate the impact of the 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

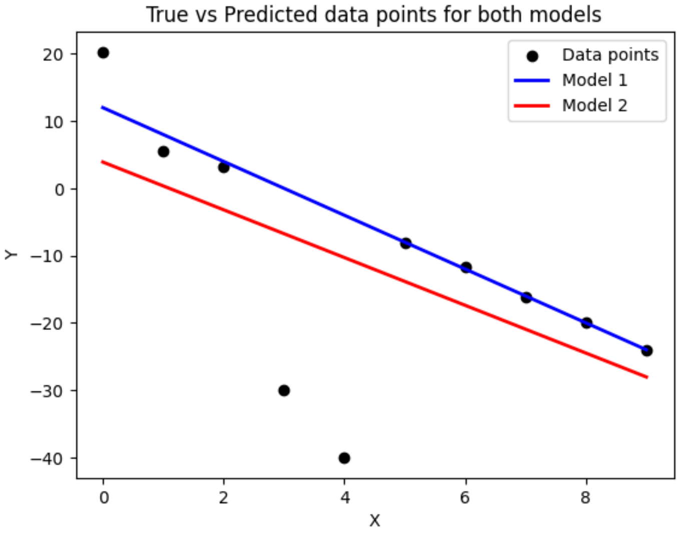


Figure 1: Model comparison

Model 1 $\Rightarrow y = -4x + 12$ is most suited
Reason:

- In the previous question we have selected that $\beta = 1$ is the most suitable value. We can select the best model depending on which model gives the minimum loss value. By looking at the loss comparison table 1 we can see that for $\beta = 1$, model 1 gives the smallest error $(0.4354 < 0.9728)$. This shows that the impact of the outliers has been mitigated by the robust estimator.
- Also by looking at the graphs for both models 1 we can see that when using model 1, the the impact of the outliers is less than in model 1, resulting in a line that fits more perfectly with the given data points.

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

The robust estimator reduces the impact of outliers by using a loss function that limits the maximum loss value for a large error. If we used MSE, it gives a very large squared error for outliers that are away from the correct line, overwhelming the system. But the loss function in the assignment has an upper bound of 1 for the loss value of a single data point.

As discussed previously, when $\beta = 1$, for a small error from non-outliers the loss value of the function will be small. But for an outlier the squared error term will be large and the loss will approximate to $\frac{error^2}{error^2} = 1$ providing a upper limit

This prevents a single extreme data point from influencing the whole model, which allows the model to be less sensitive towards outliers and find a better fit for the non-outlier data points.

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

Huber Loss Function can be used

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & \text{for } |y - \hat{y}| \leq \delta \\ \delta \cdot |y - \hat{y}| - \frac{1}{2}\delta^2, & \text{for } |y - \hat{y}| > \delta \end{cases}$$

- y = true value
- $\hat{y}$ = estimated value
- δ = A hyper parameter that defines the threshold.

In this loss function δ decides where the loss function changes between the linear and quadratic behavior. It treats the error differently depending on the threshold making it more robust. For small errors form the non-outliers we have a quadratic function, $L_\delta(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$ making it similar to MSE. It also provides a steep gradient. But for outliers the error is large($|y - \hat{y}| > \delta$) and the loss function becomes linear $\delta \cdot |y - \hat{y}| - \frac{1}{2}\delta^2$. So now the outlier will still contribute to the total loss. But since it is not squared, the impact will not be overwhelming as before. This will prevent extreme outliers from influencing the training process too much making the model robust.

2 Loss Function

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

True $y = 1$	**Prediction** $\hat{y}$	**MSE**	**BCE**
1	0.005	0.990025	5.29831737
1	0.01	0.9801	4.60517019
1	0.05	0.9025	2.99573227
1	0.1	0.81	2.30258509
1	0.2	0.64	1.60943791
1	0.3	0.49	1.20397280
1	0.4	0.36	0.91629073
1	0.5	0.25	0.69314718
1	0.6	0.16	0.51082562
1	0.7	0.09	0.35667494
1	0.8	0.04	0.22314355
1	0.9	0.01	0.10536051
1	1.0	0.00	9.992007×10^{-16}

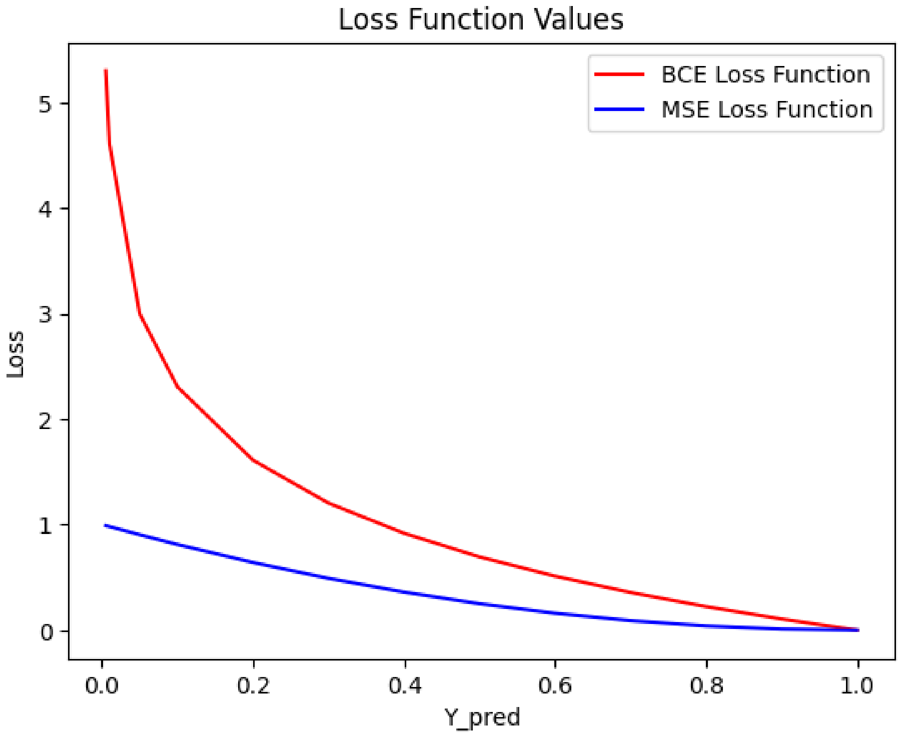


Figure 2: BCE vs MSE Comparison

The code used to generate the values and thee plots is a s follows

```python
y_pred_vals = 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

def calc_mse(y_pred):
    """Calculate the MSE"""
    return (y_true-y_pred)**2

def calc_bce(y_pred):
    """Claculate the BCE"""
    y_pred = np.clip(y_pred, 1e-15, 1 - 1e-15)
    return -(y_true*np.log(y_pred) + (1-y_true)*np.log(1-y_pred))

mse_vals = np.array([calc_mse(x) for x in y_pred_vals])
bce_vals = np.array([calc_bce(x) for x in y_pred_vals])

print(mse_vals.reshape(-1,1))
print()
print(bce_vals.reshape(-1,1))

plt.plot(y_pred_vals,bce_vals,color = "red", label = "BCE Loss Function")
plt.plot(y_pred_vals,mse_vals,color = "blue", label = "MSE Loss Function")

plt.legend(loc='best')
plt.xlabel('Y_pred')
plt.ylabel('Loss')
plt.title("Loss Function Values")
plt.show()
```

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

- Application 1: The dependent variable is continuous.
  Chose the MSE Loss Function

  Reason:

  - MSE measure the squared difference between the true value and the estimated value. If the error is large, then a larger penalty is given because of the squared term. This is needed when fitting a line to a continuous set of data points to ensure the best fit possible.
  - Moreover, in linear regression, we try to minimize the sum of squared differences and when the MSE is used as the loss function we can accomplish it.
  - MSE function is differentiable and convex. These are useful properties when applying optimization techniques such as gradient decent

  .

- Application 2: The dependent variable is discrete and binary (only takes 0 or 1).
  Chose the BCE Loss Function

  Reason:

  - BCE is the best choice when the output is a binary value of 0 or 1. This is because since the BCE uses log terms to determine the loss, when the prediction is wrong, a heavy penalty is given. This gives a strong gradient to the model and guides it towards the correct fit. This can be seen in the above comparison graph 2
  - On the other hand by looking at the graph 2 it can be seen that, compared to BCE, the loss of the MSE is relatively low even when the prediction is completely wrong (when 0 is predicted) resulting in a low gradient. This is because in the continuous case 0 and 1 are close values. However, when we do binary classifications, 0 and 1 are completely different so we need a loss function that heavily penalize the model for such errors. Therefore, BCE is the best choice for application 2.
  - Since BCE gives a strong gradient when the error is large, we can also prevent gradient saturation.

  .

3 Data pre-processing

1) Generate feature values of two features using the code given in listing 1. Considering scaling methods of (a) standard scaling, (b) min-max scaling, and (c) max-abs scaling. Select one scaling method for feature 1 and 2, ensuring that the chosen method preserves the structure/properties of the features. Justify your answer.

There are 2 ways to implement the scaling methods

- Method 1: Scaling using sklearn.preprocessing

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler

# (a) Standard Scaling
std_scaler = StandardScaler()
sparse_std = std_scaler.fit_transform(sparse_signal.reshape(-1,1))
epsilon_std = std_scaler.fit_transform(epsilon.reshape(-1,1))

# (b) Min-Max Scaling
minmax_scaler = MinMaxScaler()
sparse_minmax = minmax_scaler.fit_transform(sparse_signal.reshape(-1,1))
epsilon_minmax = minmax_scaler.fit_transform(epsilon.reshape(-1,1))

# (c) Max-Abs Scaling
maxabs_scaler = MaxAbsScaler()
sparse_maxabs= maxabs_scaler.fit_transform(sparse_signal.reshape(-1,1))
epsilon_maxabs= maxabs_scaler.fit_transform(epsilon.reshape(-1,1))
```

- Method 2: Scaling using the definition

```python
sparse_mean = np.mean(sparse_signal)
sparse_std = np.std(sparse_signal)
epsilon_mean = np.mean(epsilon)
epsilon_std = np.std(epsilon)

sparse_signal_min = np.min(sparse_signal)
sparse_signal_max = np.max(sparse_signal)
epsilon_min = np.min(epsilon)
epsilon_max = np.max(epsilon)

sparse_signal_max_abs_val = np.max(np.abs(sparse_signal))
epsilon_max_abs_val = np.max(np.abs(epsilon))

# (a) Standard Scaling
sparse_std = (sparse_signal-sparse_mean)/sparse_std
epsilon_std = (epsilon-epsilon_mean)/epsilon_std

# (b) Min-Max Scaling
sparse_minmax = (sparse_signal - sparse_signal_min) / (sparse_signal_max - sparse_signal_min)
epsilon_minmax = (epsilon - epsilon_min) / (epsilon_max - epsilon_min)

# (c) Max-Abs Scaling
sparse_maxabs = sparse_signal / sparse_signal_max_abs_val
epsilon_maxabs = epsilon / epsilon_max_abs_val
```

Results Obtained

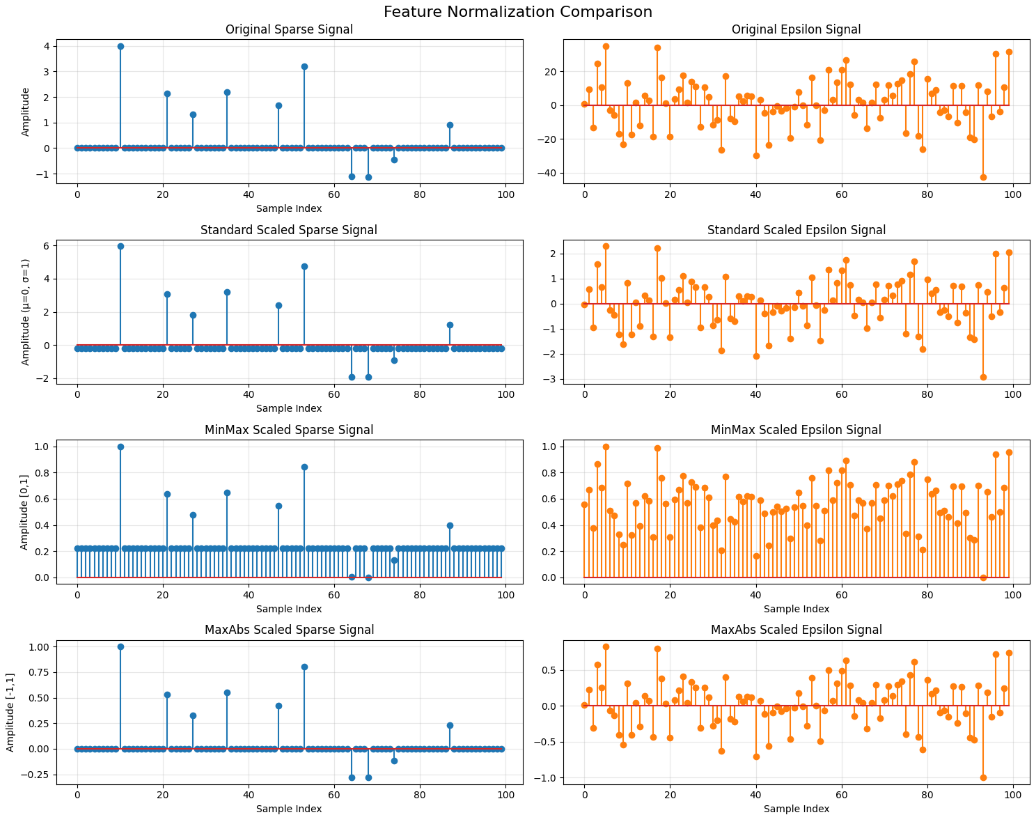


Figure 3: Normalized signals

Scaling Method	**Formula**
Standard Scaling	$x' = \frac{x-\mu}{\sigma}$
Min-Max Scaling	$x' = \frac{x-x_{\min}}{x_{\max}-x_{\min}}$
MaxAbs Scaling	$x' = \frac{x}{\lvert x_{\max}\rvert}$

Selecting suitable scaling methods and Reasoning

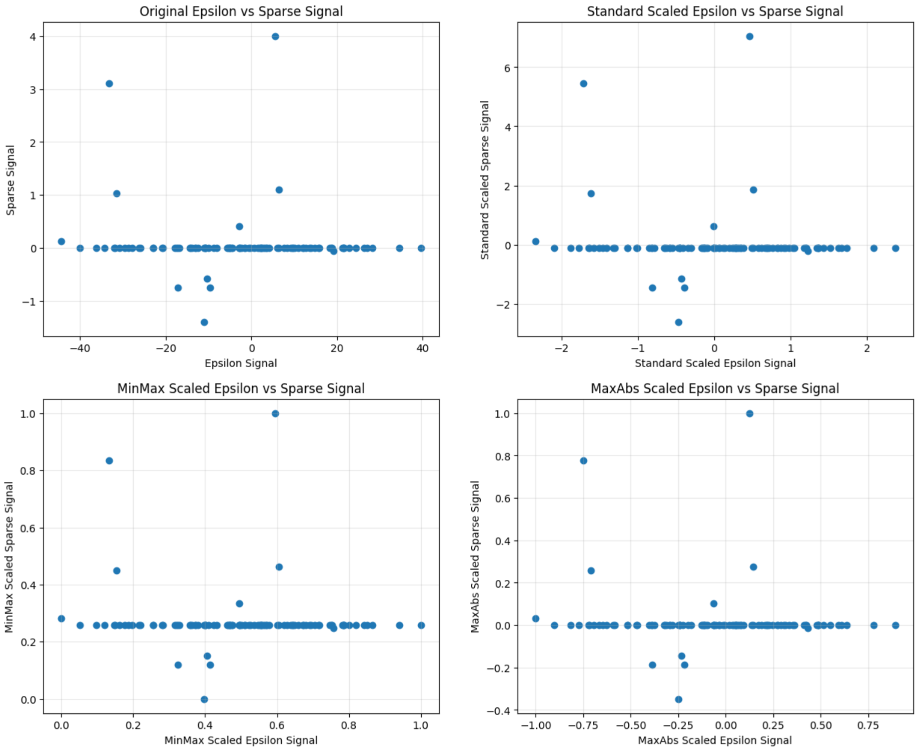
1. Feature 1: Sparse Signal

   - **Max-Abs scaling is most suitable**
   - This is a sparse signal so it has a lot of zero values and we need to maintain this property when scaling.
   - When max-abs scaling is used it just divides the signal values by the absolute maximum, so zero values stays at zero maintaining the sparsity. This can be seen by looking at the above figure 3 where the zero values remained unchanged This is important as some ML models are optimized to handle sparse data. Furthermore, in this method, the outlier at index 10 has been limited to 1.

- If standard scaling is used, it shifts all data points by the mean destroying the sparsity of the signal. As shown in the figure 3, the zero values have changed to non-zero values due to the mean being non zero. So it is not suitable.
- Similarly, Min-max scaling also transforms the zero values to a non-zero value by subtracting the minimum value of the signal, destroying the sparsity of the signal. The effect is even greater than the standard scaling as shown in the above figure. Hence, it is not suitable.

2. Feature 2: Epsilon

- **Standard Scaling is most suitable**
- This signals is generated using a normal distribution of mean 0. So the mean is already 0. So when using the standard scaling the data points will only get scaled and they will not get shifted. And finally they will have a mean of 0 and a standard deviation of 1 which is preferred in ML models. As show in Figure 3, the original structure is preserved.
- Min-max scaling is not suitable as it shifts the data, making the mean non-zero. This will ruin the structure of the signal. This can clearly be seen in Figure 3. Furthermore, this scaling would be more sensitive towards outliers making the whole distribution skewed ($x_{\max} - x_{\min}$ will be a large value in case of outliers).
- Even though Max-abs scaling does not shift the data points (shown in 3), it still fails to scale the data with zero mean and unit variance. Zero mean and unit variance is a main benefit of standard scaling. Moreover, max-abs scaling is more suited for sparse data and less suited for normally distributed data because of this.


Original Epsilon vs Sparse Signal
Sparse Signal
Epsilon Signal
Standard Scaled Epsilon vs Sparse Signal
Standard Scaled Sparse Signal
Standard Scaled Epsilon Signal
MinMax Scaled Epsilon vs Sparse Signal
MinMax Scaled Sparse Signal
MinMax Scaled Epsilon Signal
MaxAbs Scaled Epsilon vs Sparse Signal
MaxAbs Scaled Sparse Signal
MaxAbs Scaled Epsilon Signal