Machine Learning Assignment-1

Parth Sandeep Rastogi
IIIT DELHI
parth22352@iiitd.ac.in

1. Section A

1.1. a) Effect of Increasing Model Complexity on Bias and Variance

As the complexity of a machine learning model increases, for example by adding more features or higher-order polynomial terms, the following typically occurs:

Bias decreases: A complex model can capture more intricate patterns in the data, allowing it to better fit the true underlying relationship. As a result, the model makes fewer simplifying assumptions and reduces bias. However, when the model becomes too complex, it may try to fit every detail in the training data, even capturing noise.

Variance increases: A more complex model becomes highly sensitive to the fluctuations and noise present in the training data. This means that the model may perform well on the training set but will likely overfit, leading to high error when making predictions on unseen data.

The tradeoff between bias and variance is called the *bias-variance tradeoff*. As complexity increases, bias decreases, but variance rises. The ideal model complexity is one that strikes the right balance, minimizing total error (which includes both bias and variance).

This can be visualized with the following example:

In this analysis, we generate a dataset from the function $y = \log(x) - x\sin(x)$, with added noise. This function introduces a non-linear relationship between the input and output variables. We fit polynomial regression models of varying degrees to the data and observe the effects of increasing model complexity on bias, variance, and error.

Explanation using Example:

Bias: In this case, the bias represents the difference between the true function $y = \log(x) - x\sin(x)$ and the predictions made by the model. For simple models (e.g., lower-degree polynomials), bias is high because they

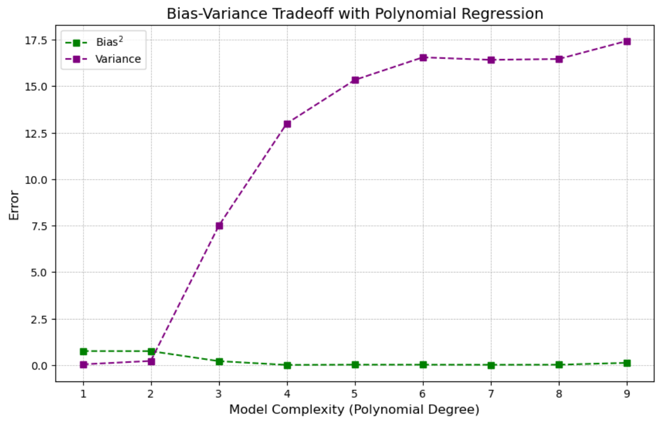

Figure 1. Bias-Variance Tradeoff for Polynomial Regression on $y = \log(x) - x\sin(x)$

cannot capture the non-linear structure of the true function.

Variance: Complex models, such as higher-degree polynomials, can closely fit training data, but they also pick up noise, These models fluctuate significantly based on small changes in the training data, leading to overfitting.

As seen in the graph, the **bias** decreases as model complexity increases because higher-degree polynomials can approximate the non-linear function more accurately. However, the **variance** increases due to overfitting, and this causes the total error (a combination of bias and variance) to first decrease, reach a minimum, and then increase again. The goal is to select model with complexity that minimizes the overall error, balancing between bias and variance.

1.2. (b) Evaluating Email Filtering System

We can evaluate the email filtering system using the following metrics:

- **True Positives (TP)**: Correctly classified spam emails.

$$TP = 200$$

- **False Negatives (FN)**: Spam emails incorrectly classified as legitimate.

$$FN = 50$$

- **True Negatives (TN)**: Correctly classified legitimate emails.
$$TN = 730$$
- **False Positives (FP)**: Legitimate emails incorrectly classified as spam.
$$FP = 20$$

The average classification performance can be assessed using **accuracy**:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Substituting the values:

$$\text{Accuracy} = \frac{200 + 730}{200 + 730 + 20 + 50} = \frac{930}{1000} = 0.93$$

Thus, the model's overall accuracy is 93%.

(b) Evaluating Email Filtering System

We can evaluate the email filtering system using the following metrics:

- **True Positives (TP)**: Correctly classified spam emails.
$$TP = 200$$
- **False Negatives (FN)**: Spam emails incorrectly classified as legitimate.
$$FN = 50$$
- **True Negatives (TN)**: Correctly classified legitimate emails.
$$TN = 730$$
- **False Positives (FP)**: Legitimate emails incorrectly classified as spam.
$$FP = 20$$

The average classification performance can be assessed using the following metrics:

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Substituting the values:

$$\text{Accuracy} = \frac{200 + 730}{200 + 730 + 20 + 50} = \frac{930}{1000} = 0.93$$

Thus, the model's overall accuracy is 93%.

Precision:

$$\text{Precision} = \frac{TP}{TP + FP}$$

This measures the proportion of positive identifications that were actually correct (i.e., how many of the emails predicted as spam were actually spam).

$$\text{Precision} = \frac{200}{200 + 20} = \frac{200}{220} \approx 0.91$$

Thus, the precision of the model is 91%.

Recall (Sensitivity):

$$\text{Recall} = \frac{TP}{TP + FN}$$

This measures the proportion of actual positives that were correctly identified (i.e., how many spam emails were correctly identified as spam).

$$\text{Recall} = \frac{200}{200 + 50} = \frac{200}{250} = 0.80$$

Thus, the recall of the model is 80%.

Negative Precision:

$$\text{Negative Precision} = \frac{TN}{TN + FN}$$

This measures how well the system identifies legitimate emails (i.e., how many emails predicted as legitimate were actually legitimate).

$$\text{Negative Precision} = \frac{730}{730 + 50} = \frac{730}{780} \approx 0.94$$

Thus, the negative precision of the model is 94%.

- **Specificity**:

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{730}{750} = 0.9733$$

The accuracy for identifying legitimate emails is 97.33

F1 Score:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

This provides a balance between precision and recall, particularly useful when there is an uneven class distribution.

$$\text{F1 Score} = 2 \times \frac{0.91 \times 0.80}{0.91 + 0.80} \approx 2 \times \frac{0.728}{1.71} \approx 0.851$$

Thus, the F1 score is 85.1%.

—

Weighted and Unweighted Average of Classwise Accuracy:

- **Unweighted Average Accuracy:**

$$\text{Unweighted Average} = \frac{\text{Accuracy}_{\text{Spam}} + \text{Accuracy}_{\text{Legitimate}}}{2}$$

$$= \frac{0.80 + 0.9733}{2} \approx 0.8867$$

Thus, the unweighted average accuracy is approximately 88.67%.

- **Weighted Average Accuracy:** To calculate the weighted average, we use the number of instances in each class:

$$\text{Weighted Average} = \frac{(\text{Acc}_{\text{Spam}} \times (TP + FN)) + (\text{Acc}_{\text{Legit}} \times (TN + FP))}{(TP + FN) + (TN + FP)}$$

Substituting the values:

$$\text{Weighted Average} = \frac{(0.80 \times 250) + (0.9733 \times 750)}{1000}$$

$$= \frac{200 + 729.975}{1000} \approx 0.929975$$

Thus, the weighted average accuracy is approximately 92.99%.

Explanation of the Metrics for Classwise accuracy:

- Precision: Measures how many predicted positives (spam) were actually correct. A high precision means the system makes fewer false positives, important when it's costly to misclassify legitimate emails as spam.

- Recall: Indicates how many of the actual positives (spam) were correctly identified. A higher recall means the system is good at catching most of the spam emails.

- Negative Precision: Measures how many emails predicted as legitimate were indeed legitimate. It's useful in cases where misclassifying legitimate emails as spam is a concern.

- Classwise Accuracy: This gives a better understanding of how well the model performs for each class individually. In this case, spam and legitimate email classes have different accuracies.

- F1 Score: The harmonic mean of precision and recall, F1 is a single number that balances the two. It's particularly helpful when the distribution between spam and legitimate emails is uneven, ensuring the model performs well on both classes without over-prioritizing one.

1.3. (c) Finding the Equation of the Regression Line

The regression line is in the form $y = mx + c$, where m is the slope and c is the y-intercept. To find m and c, we use the following formulas:

$$m = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

$$c = \frac{\sum y - m(\sum x)}{n}$$

Using the data:

$$x = [3, 6, 10, 15, 18], \quad y = [15, 30, 55, 85, 100]$$

We calculate:

$$\sum x = 52, \quad \sum y = 285, \quad \sum xy = 3850, \quad \sum x^2 = 694$$

$$m = \frac{5(3850) - (52)(285)}{5(694) - (52)^2} = \frac{19250 - 14820}{3470 - 2704} = \frac{4430}{766} \approx 5.783$$

$$c = \frac{285 - 5.783(52)}{5} = \frac{285 - 300.716}{5} \approx -3.143$$

Thus, the regression line is:

$$y = 5.783x - 3.143$$

To predict y when $x = 12$:

$$y = 5.783(12) - 3.143 = 69.396 - 3.143 = 66.253$$

Hence, when $x = 12$, $y \approx 66.253$.

1.4. Empirical Risk and Generalization

In the context of machine learning, empirical risk is defined as the average loss over the training dataset. Given a model f and a dataset with features X and labels Y, we can denote the predicted output as $\hat{f}(X)$ and the loss function as $L(\hat{f}(X), Y)$. The empirical risk $R(f)$ for a model f is then calculated as:

$$R(f) = \frac{1}{n} \sum_{i=1}^{n} L(\hat{f}(x_i), y_i)$$

Consider two models f_1 and f_2 with the following empirical risks:

$$R(f_1) < R(f_2)$$

This indicates that model f_1 has a lower empirical risk on the training set compared to model f_2. However, lower empirical risk on the training dataset does not guarantee better generalization to unseen data.

Toy Example:

Let us consider a simple toy dataset where the features and labels are as follows:

$$X = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]$$

$$Y = [5000, 7000, 8000, 8500, 8900, 9200, 9500, 9700, 9800, 9900]$$

Assume we fit two different models to this data: - f_1: A high-degree polynomial regression model (degree 8). - f_2: A linear regression model (degree 1).

The Mean Squared Errors (MSE) for both models are as follows:

- For the linear model f_2 (degree 1): - Train MSE: 502467.67 - Test MSE: 117317.55

- For the polynomial model f_1 (degree 8): - Train MSE: 0.00 - Test MSE: 146282.19

In this scenario, f_1 achieves a lower empirical risk by perfectly fitting the training data, resulting in a train MSE of 0.00. However, it has a higher test MSE compared to the linear model, which indicates that while f_2 has a higher training error, it generalizes better to unseen data.

As illustrated in the plot below, we can observe the Mean Squared Error (MSE) for both models on the training and test datasets:

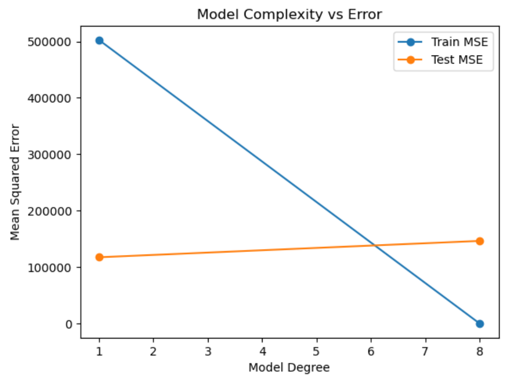

Figure 2. Model Complexity vs Error

This example highlights the critical distinction between empirical risk minimization and the generalization performance of a model, emphasizing that a model with lower training error does not necessarily imply better performance on unseen data.

2. Section B :- Scratch Based implementation

2.1. Necessary Preprocessing

The dataset initially had an imbalanced distribution of class labels, with significantly fewer data points for Class 1 compared to Class 0. This imbalance caused the model to be biased towards predicting Class 0, resulting in high overall accuracy but poor performance for Class 1.

To mitigate this issue, I applied upsampling to the Class 1 data points after splitting the data into training, validation, and test sets. I used the random.choice method with replacement to increase the number of Class 1 samples to match those of Class 0. The following is the class distribution before and after upsampling:

Before Upsampling:

- Class 0: 2521
- Class 1: 446

After Upsampling:

- Class 0: 2521
- Class 1: 2521

Additionally, for handling missing values in the dataset, I performed for All the variables, I used median substitution, which effectively handles outliers compared to mean imputation and also as median comes from real data and mean may give a value that shouldn't even be an possible value and also median accounts for the skewness of the data

2.2. Logistic Regression using Batch Gradient Descent (3 marks)

2.2.1 (a) Implementation:

I implemented Logistic Regression using Batch Gradient Descent for binary classification. The model was trained using a fixed learning rate, and the weights were updated iteratively to minimize the loss function. For evaluation, both training and validation sets were used to monitor the performance of the model at each iteration.

2.2.2 (b) Plots:

Two plots were generated to track the performance of the model during training:

1. **Training vs. Validation Loss:** This plot shows the training loss and validation loss over the iterations. The training loss converges around 0.65, while the validation loss converges slightly lower at around 0.66.
2. **Training vs. Validation Accuracy:** This plot displays the accuracy for both the training and validation sets as the model progresses through the iterations. The accuracy converges at approximately 60% for both training and validation sets.

Algorithm 1 Batch Gradient Descent

```
1: Input: Training data $X_{\text{train}}, y_{\text{train}}$, Validation data $X_{\text{val}}, y_{\text{val}}$, learning rate $\alpha$, number of epochs $E$
2: Output: Weights, Bias, Training losses, Training accuracies, Validation losses, Validation accuracies
3: Initialize weights $w \leftarrow 0$ and bias $b \leftarrow 0$
4: Initialize lists: losses, accuracies, val_losses, val_accuracies
5: for epoch = 1 to $E$ do
6:     $z \leftarrow X_{\text{train}} \cdot w + b$
7:     $y_{\text{hat}} \leftarrow \text{sigmoid}(z)$
8:     $dw \leftarrow \frac{1}{m} \cdot X_{\text{train}}^T \cdot (y_{\text{hat}} - y_{\text{train}})$
9:     $db \leftarrow \frac{1}{m} \cdot \sum(y_{\text{hat}} - y_{\text{train}})$
10:    $w \leftarrow w - \alpha \cdot dw$
11:    $b \leftarrow b - \alpha \cdot db$
12:    $loss \leftarrow \text{cross\_entropy}(y_{\text{train}}, y_{\text{hat}})$
13:    Append $loss$ to losses
14:    $y_{\text{pred\_class}} \leftarrow \text{round}(y_{\text{hat}})$
15:    $accuracy \leftarrow \text{accuracy\_score}(y_{\text{train}}, y_{\text{pred\_class}})$
16:    Append $accuracy$ to accuracies
17:    $z_{\text{val}} \leftarrow X_{\text{val}} \cdot w + b$
18:    $y_{\text{val\_hat}} \leftarrow \text{sigmoid}(z_{\text{val}})$
19:    $val\_loss \leftarrow \text{cross\_entropy}(y_{\text{val}}, y_{\text{val\_hat}})$
20:    Append $val\_loss$ to val_losses
21:    $y_{\text{val\_pred\_class}} \leftarrow \text{round}(y_{\text{val\_hat}})$
22:    $val\_accuracy \leftarrow \text{accuracy\_score}(y_{\text{val}}, y_{\text{val\_pred\_class}})$
23:    Append $val\_accuracy$ to val_accuracies
24: end for
25: Return: weights, bias, losses, accuracies, val_losses, val_accuracies
```

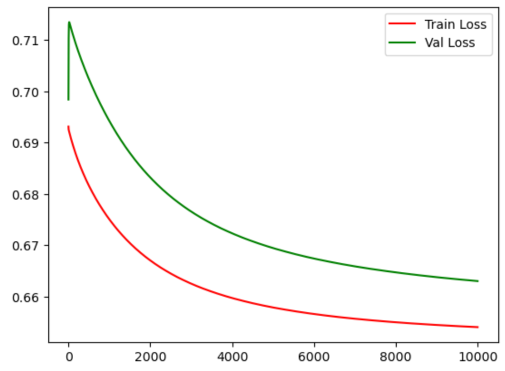

Figure 3. Training vs Validation Loss

2.2.3 (c) Analysis:

From the plots, it is evident that the model converges reasonably well, with both training and validation loss stabilizing after several iterations. Although the Training loss is slightly lower than the validation loss, indicating good generalization, the overall accuracy is around 61%, which suggests that the model may not be sufficiently complex to capture all patterns in the data. The confusion matrix for the predictions is as follows:

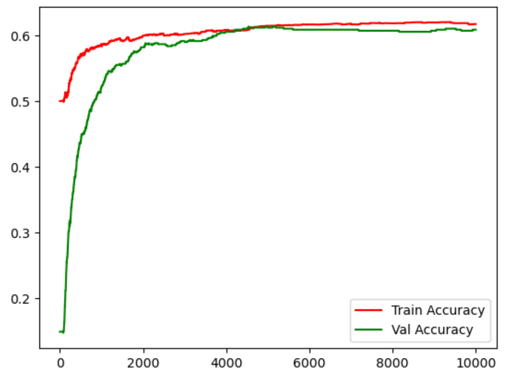

Figure 4. Training vs Validation Accuracy

Predicted	0	1
Actual 0	324	217
Actual 1	32	63

From the confusion matrix, we observe that while the model performs reasonably well in predicting both classes.

The following metrics summarize the model's performance for Class 1:

- **Precision:** 0.225
- **Recall:** 0.66
- **F1 Score:** 0.336

Tuning of hyperparameters such as the learning rate to 0.0001 and epochs to 10000 helped in generalization and convergence

2.3. Min Max Scaling Methods (2 marks)

2.3.1 (a) Investigation:

Formula Used for scaling is here the max and min are taken on train and so any new point that we want to predict will go through this scaling

$$\text{x} = \frac{x - \max(x)}{\max(x) - \min(x)}$$

To analyze the impact of feature scaling on the performance of Logistic Regression, I compared the model's behavior with min max scaling and without scaling:

- **Min-Max Scaling:** In this method, each feature is scaled to a fixed range, typically between 0 and 1. This ensures that all features contribute equally to the model, without any dominating due to different scales.

- **No Scaling:** In this case, the model was trained with raw feature values, without any normalization or scaling.

2.3.2 (b) Plots:

The following plots illustrate the performance of the model with both scaling methods:

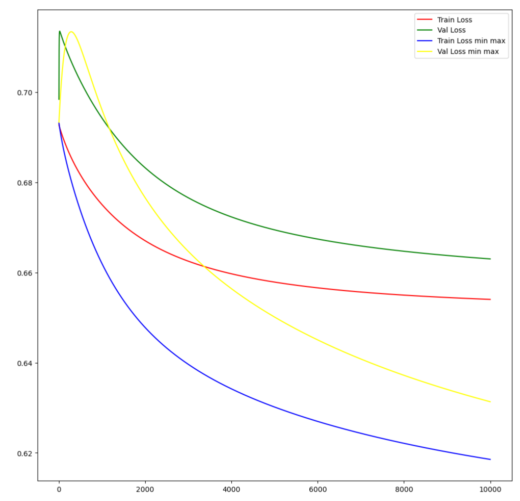

Figure 5. Training vs Validation Loss for Min-Max Scaling and No Scaling

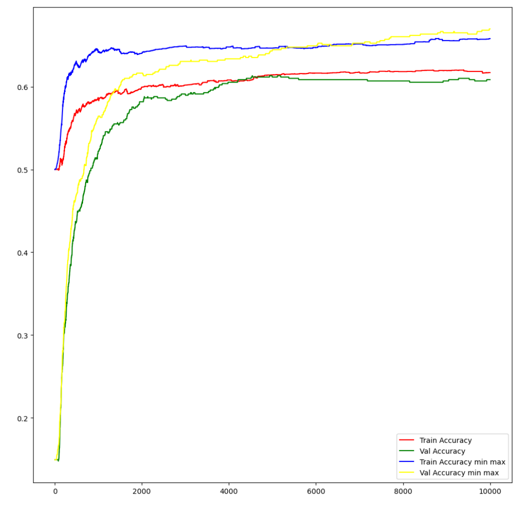

Figure 6. Training vs Validation Accuracy for Min-Max Scaling and No Scaling

2.3.3 (c) Discussion:

Feature scaling has a significant impact on the convergence of the model:

- **With Min-Max Scaling:** The gradient descent optimization initially converged slower with the same learning rate, requiring a tenfold increase in the learning rate to achieve proper convergence. Despite this adjustment, Min-Max scaling led to a lower final loss, with a training loss of 0.61 and a validation loss of 0.63. Accuracy improved from 60% to 66% on the validation set, as shown in Figures 5 and 6.

- **Without Scaling:** The model converged more quickly with the initial learning rate, but this resulted in a higher final loss (training loss of 0.65 and validation loss of 0.66) and lower accuracy (61%). The lack of scaling caused some features with larger ranges to dominate the gradient updates, making the optimization process less effective.

2.3.4 (D) Conclusion :

Min-Max scaling slows model convergence due to smaller gradient updates as seen in the figure, as scaled features have reduced impact. However, it ultimately achieves a lower final loss, as scaling ensures all features contribute equally, allowing the model to better optimize. Despite slower convergence, scaling improves accuracy by making the model more sensitive to subtle patterns and reducing the dominance of any single feature. This balance helps the model generalize better, making the slower convergence worthwhile for improved performance.

2.4. Confusion Matrix and Performance Metrics

For the validation set, we calculate and present the confusion matrix along with performance metrics such as precision, recall, F1 score, and ROC-AUC score for both Min-Max scaled and no scaling cases. These metrics provide insights into the model's performance by evaluating its ability to correctly classify instances of each class.

2.4.1 Confusion Matrix

Actual	Predicted 0	Predicted 1
0	324	217
1	32	63

Table 1. Confusion Matrix for no scaling

Actual	Predicted 0	Predicted 1
0	360	181
1	29	66

Table 2. Confusion Matrix for scaled model

2.4.2 Inference From Confusion Matrix

In Scaling The Class 0 prediction improved noticably and Class 1 prediction improved slightly

2.4.3 Performance Metrics

Metric	Min-Max Scaling	No Scaling
Precision	0.267	0.225
Recall	0.694	0.6631
F1 Score	0.385	0.336
ROC-AUC Score	0.73	0.67

Table 3. Performance Metrics for Validation Set

2.4.4 ROC-AUC and precision recall Curve

ROC-AUC and precision recall curve for Min-Max Scaling

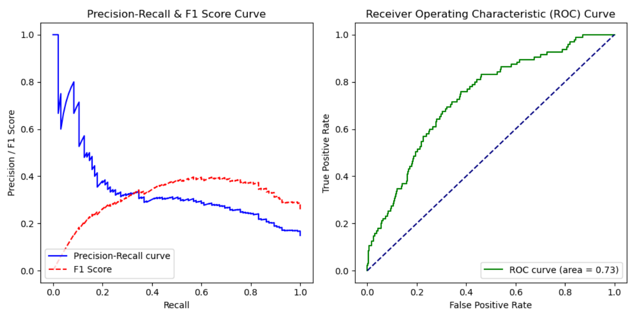

Figure 7. ROC-AUC Curve and Precision Recall Curve for Min-Max Scaling

ROC-AUC Curve and precision recall curve for No Scaling

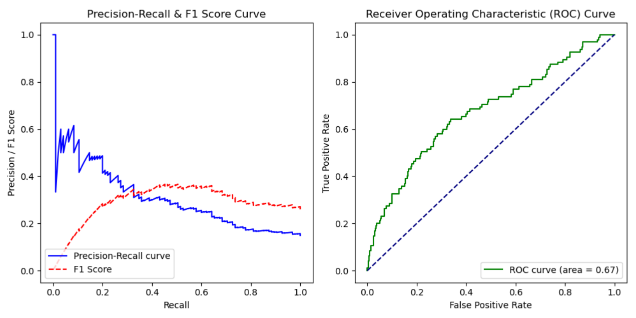

Figure 8. ROC-AUC Curve and Precision Recall Curve for No Scaling

2.4.5 Inference and Discussion

The confusion matrix provides a visual summary of the model's classification performance by showing the counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Precision, recall, and the F1 score offer additional insights into the model's performance:

- **Precision** measures the proportion of positive predictions that are actually correct and is calculated as Precision $= \frac{TP}{TP+FP}$. A higher precision indicates fewer false positives.
- **Recall** (or Sensitivity) measures the proportion of actual positives that were correctly identified by the model and is calculated as Recall $= \frac{TP}{TP+FN}$. Higher recall indicates fewer false negatives.
- **F1 Score** is the harmonic mean of precision and recall and is calculated as F1 Score $= 2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$. A higher F1 score indicates a better balance between precision and recall.
- **ROC-AUC Score** measures the model's ability to distinguish between classes across all possible thresholds and ranges from 0 to 1, where 1 indicates perfect classification, and 0.5 indicates no discriminatory power. A higher ROC-AUC score reflects better model performance.

Analysis:

- The **ROC-AUC score** is 0.73 with Min-Max scaling compared to 0.67 without scaling. This indicates that scaling improves the model's ability to distinguish between classes, leading to better overall performance.
- **Precision** increases slightly with Min-Max scaling (0.267 vs. 0.225), reflecting a reduction in false positives.
- **Recall** also improves with Min-Max scaling (0.694 vs. 0.663), showing that scaling helps the model better identify positive instances.
- The **F1 score** is higher with Min-Max scaling (0.385 vs. 0.336), demonstrating a better balance between precision and recall.

In summary, Min-Max scaling enhances the model's classification performance by improving precision, recall, and the ROC-AUC score, leading to a more accurate and balanced model.

2.5. Stochastic Gradient Descent (SGD) and Mini-Batch Gradient Descent

2.5.1 Introduction

In this section, we compare the performance of Stochastic Gradient Descent (SGD) and Mini-Batch Gradient Descent (MBGD) with varying batch sizes-10, 50, 100. We plot the loss vs. iteration and accuracy vs. iteration for each method to analyze their convergence properties and stability.

2.5.2 Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent

```
1: Input: Training data $X_{\text{train}}$, labels $y_{\text{train}}$, validation data $X_{\text{val}}$, labels $y_{\text{val}}$, learning rate $\alpha$, number of epochs $E$
2: Initialize weights $w$ and bias $b$ to zero
3: for epoch = 1 to $E$ do
4:     for each training sample $(x_i, y_i)$ do
5:         Compute prediction $\hat{y}_i = \text{sigmoid}(x_i^T w + b)$
6:         Compute gradients $dw$ and $db$
7:         Update weights $w \leftarrow w - \alpha \cdot dw$
8:         Update bias $b \leftarrow b - \alpha \cdot db$
9:     end for
10:    Compute loss and accuracy on training and validation data
11:    Store loss and accuracy
12: end for
```

2.5.3 Mini-Batch Gradient Descent (MBGD)

Algorithm 3 Mini-Batch Gradient Descent

```
1: Input: Training data $X_{\text{train}}$, labels $y_{\text{train}}$, validation data $X_{\text{val}}$, labels $y_{\text{val}}$, learning rate $\alpha$, number of epochs $E$, batch size $B$
2: Initialize weights $w$ and bias $b$ to zero
3: for epoch = 1 to $E$ do
4:     Shuffle training data
5:     for each mini-batch $(X_{\text{batch}}, y_{\text{batch}})$ do
6:         Compute prediction $\hat{y} = \text{sigmoid}(X_{\text{batch}} \cdot w + b)$
7:         Compute gradients $dw$ and $db$
8:         Update weights $w \leftarrow w - \alpha \cdot dw$
9:         Update bias $b \leftarrow b - \alpha \cdot db$
10:    end for
11:    Compute loss and accuracy on training and validation data
12:    Store loss and accuracy
13: end for
```

2.5.4 Plots

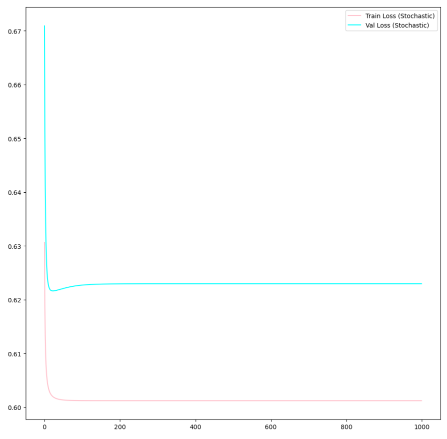

Figure 9. SGD Training and Val Loss vs. Iteration

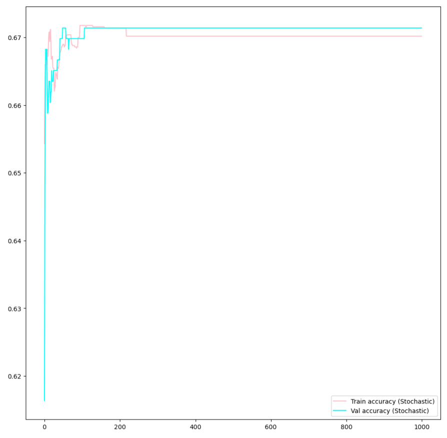

Figure 10. SGD Training and Val accuracy vs. Iteration

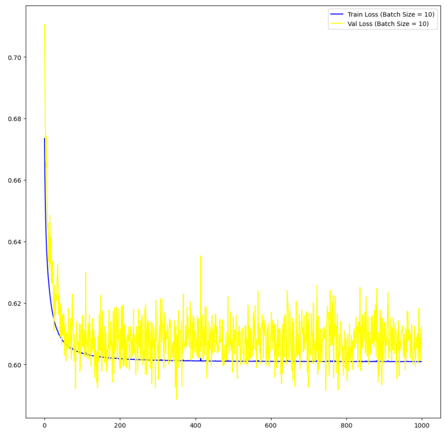

Figure 11. MBGD Batch size 10 Training and Val Loss vs. Iteration

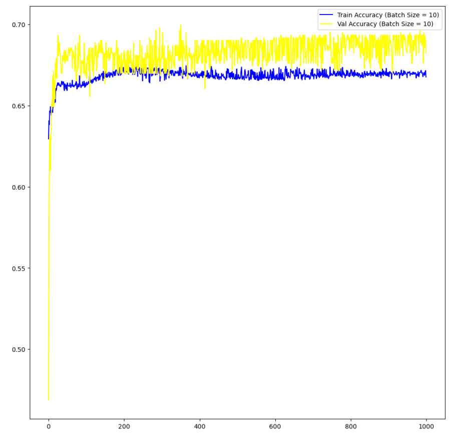

Figure 12. MBGD Batch size 10 Training and Val accuracy vs. Iteration

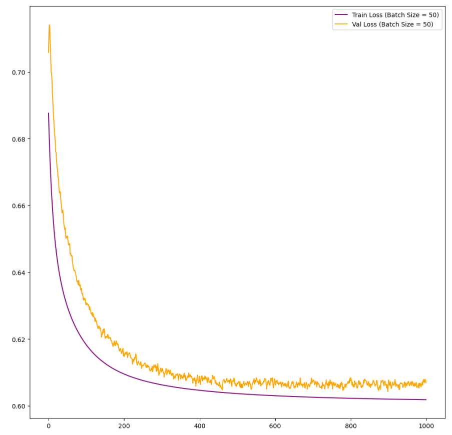

Figure 13. MBGD Batch size 50 Training and Val Loss vs. Iteration

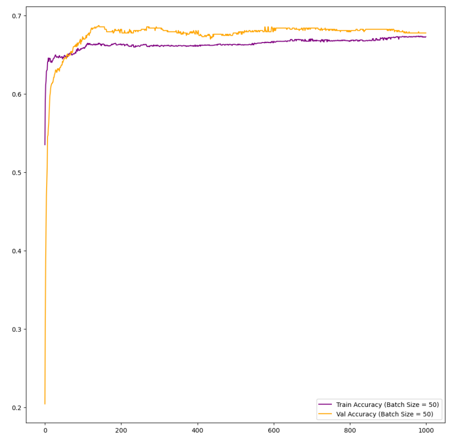

Figure 14. MBGD Batch size 50 Training and Val accuracy vs. Iteration

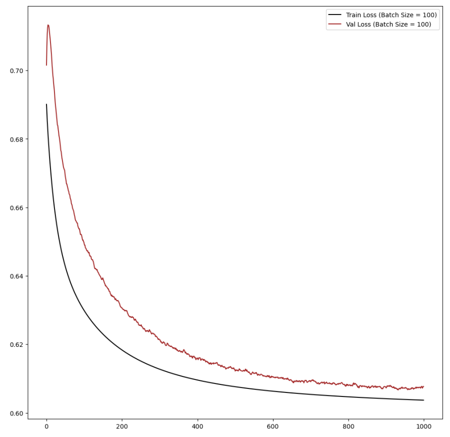

Figure 15. MBGD Batch size 100 Training and Val Loss vs. Iteration

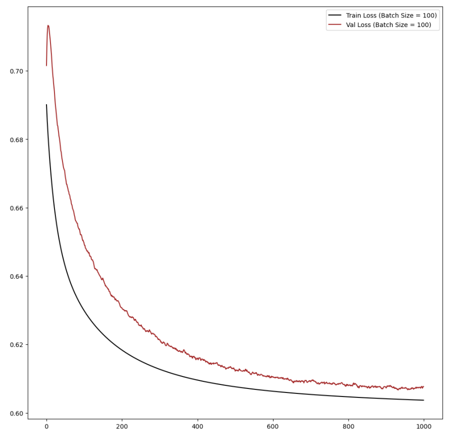

Figure 16. MBGD Batch size 100 Training and Val accuracyvs. Iteration

2.5.5 Discussion

The Figure 46 indicate that Stochastic Gradient Descent (SGD) converges faster compared to Mini-Batch Gradient Descent, as evidenced by quicker reduction in training loss and increase in accuracy. This is due to the more frequent updates in SGD, which leads to faster adjustments. On the other hand, Mini-Batch Gradient Descent, especially with larger batch sizes, shows slower convergence. And another correlation is more the batch size faster is the convergence This is attributed to fewer updates per iteration, which results in a more gradual approach to optimization.

The validation accuracy is generally higher for Mini-Batch and Stochastic Gradient Descent methods compared to training accuracy. This is because the training set was upsampled to improve generalization, while the validation set, drawn from a different distribution, reflects a more realistic performance measure.

In conclusion, while SGD provides faster convergence, Mini-Batch Gradient Descent with larger batch sizes achieves more stable convergence but takes longer to reach an optimal solution.

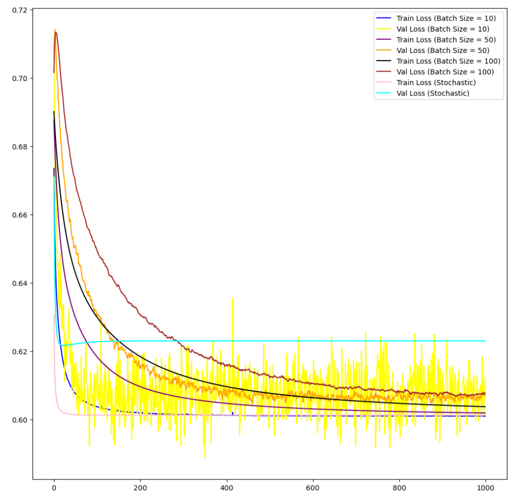

Figure 17. Comparison of convergence speed for SGD and Mini-Batch Gradient Descent with varying batch sizes.

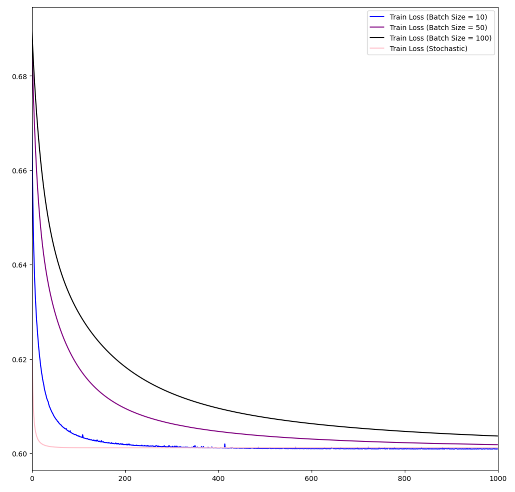

Figure 18. Comparison of Just train Losses

2.6. K-Fold Cross-Validation (k=5) with Mini-Batch Gradient Descent

To assess the robustness of the model, we performed 5-fold cross-validation using mini-batch gradient descent with a batch size of 50. The evaluation metrics used were accuracy, precision, recall, and F1 score. Below are the results of the model across the five folds, including the average and standard deviation for each metric.

Fold	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Fold 1	0.6919	0.2776	0.6484	0.3888
Fold 2	0.6635	0.2500	0.6585	0.3624
Fold 3	0.6588	0.2795	0.7132	0.4017
Fold 4	0.6470	0.2682	0.6571	0.3810
Fold 5	0.7013	0.2596	0.6379	0.3691
Mean(μ)	0.6729	0.2678	0.6630	0.3808
Sigma(σ)	0.0196	0.0109	0.0286	0.0145

Table 4. K-Fold Cross-Validation Results for Accuracy, Precision, Recall, and F1 Score

- **Accuracy:** The average accuracy across the folds is 0.67 with a standard deviation of 0.02. This indicates a relatively stable performance with minimal fluctuation in accuracy across different folds.
- **Precision:** The average precision is 0.27 with a standard deviation of 0.01. Precision shows low variability, suggesting that the model's ability to correctly identify positive cases is consistent across folds.
- **Recall:** The average recall is 0.66 with a standard deviation of 0.03. Recall exhibits slightly more variability compared to accuracy and precision, implying that the model's ability to identify all relevant positive instances varies more across folds.
- **F1 Score:** The average F1 score is 0.38 with a standard deviation of 0.01. This metric combines precision and recall, and its relatively low variability suggests that the model maintains a balanced performance across different folds.

It is important to note that the training data was upsampled after each train/validation split. This ensures that the model learns effectively from a balanced dataset, which is crucial when dealing with imbalanced data. Upsampling helps improve the model's ability to correctly identify instances from underrepresented classes.
These results suggest that the model is relatively stable across different folds with low variance, especially in terms of accuracy and F1 score. However, recall shows slightly higher variability, indicating that the model's ability to correctly identify positive cases fluctuates more across the folds.

2.7. Early Stopping in Mini-Batch Gradient Descent with Regularization

In this experiment, we implemented early stopping in mini-batch gradient descent with a batch size of 50 to prevent overfitting and improve generalization. Early stopping halts the training process when the validation loss fails to improve for a set number of iterations (patience), in this case, 1000 iterations. patience is a hyperparameter used to control the early stopping mechanism, balancing the need for continued training with the risk of overfitting

We explored different combinations of learning rates and regularization techniques (L1 and L2 regularization) to analyze their impact on the model's performance. The regularization constants used were $\lambda = 0.1$ and $\lambda = 0.01$, and the model's performance was evaluated by plotting training loss versus iterations for each combination.

2.7.1 Regularization Constants and Early Stopping

Two regularization constants were used:

- $\lambda = 0.1$: The higher regularization rate flattened the loss curve earlier, but resulted in higher overall loss and later stopping.
- $\lambda = 0.01$: The lower regularization rate caused the loss curve to flatten later, but achieved lower loss overall and stopped training earlier.

2.7.2 L1 vs L2 Regularization

The experiment compared the effects of L1 and L2 regularization:

- **L1 Regularization:** Promoted sparsity in the model's weights and led to slightly more loss compared to L2. The stopping point was later, and the regularization caused the loss curve to flatten more gradually.
- **L2 Regularization:** Regularized more aggressively, leading to a smoother loss curve with slightly lower loss, and earlier stopping in comparison to L1.

2.7.3 Performance Comparison: Loss Curves

The plot below shows the training loss versus iterations for different combinations of λ and regularization techniques:

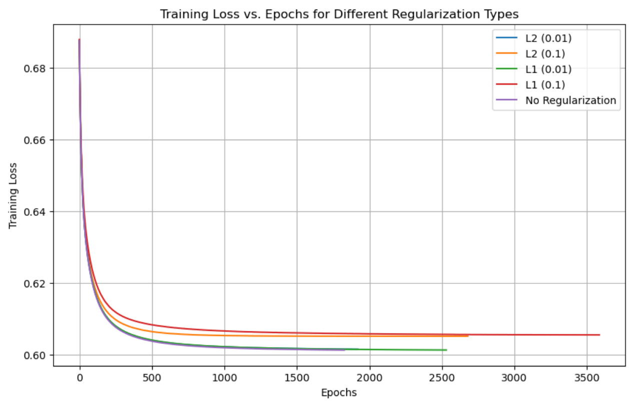

Figure 19. Training Loss vs Iterations for L1 and L2 Regularization with $\lambda = 0.1$ and $\lambda = 0.01$ (with Early Stopping)

2.7.4 Analysis of Regularization and Early Stopping

- **Effect of Regularization Rate:** The higher regularization constant ($\lambda = 0.1$) caused the loss curve to flatten earlier, but the model incurred a higher overall loss and stopped later. In contrast, the lower regularization constant ($\lambda = 0.01$) flattened the curve later, but achieved a lower loss and triggered early stopping earlier.
- **L1 vs L2 Regularization:** L1 regularization induced sparsity but stopped later with slightly higher loss compared to L2, which provided smoother generalization and earlier stopping.

In conclusion, the regularization constant and type significantly influenced both the stopping point and the loss achieved. While a higher regularization rate ($\lambda = 0.1$) caused earlier flattening of the loss curve but with higher loss, a lower regularization rate ($\lambda = 0.01$) resulted in lower loss and earlier stopping. L2 regularization provided smoother generalization, while L1 promoted sparsity but led to a slightly higher loss.

3. C) Package Based Implementation

This report details the exploratory data analysis (EDA), dimensionality reduction, and linear regression model evaluation performed on the Electricity Bill dataset. The tasks include creating various visualizations, using the UMAP algorithm for dimensionality reduction, and evaluating the linear regression model's performance based on metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R2 score, Adjusted R2 score, and Mean Absolute Error (MAE).

3.1. Section A: Exploratory Data Analysis (EDA)

3.1.1 Pair Plot, Box Plots, and Violin Plots

To understand the distribution of the dataset features, pair plots, box plots, and violin plots were created. Below are five insights derived from these visualizations:

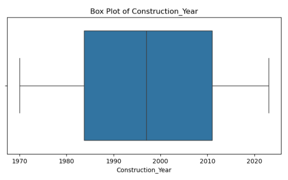

Figure 20. Construction Year box

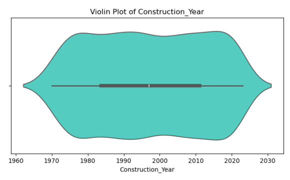

Figure 21. Construction Year violin

- **Construction Year:** The distribution is approximately normal, with most buildings constructed around the year 1997.

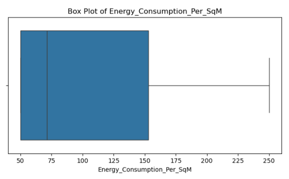

Figure 22. Energy Consumption per Square Meter

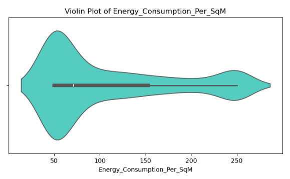

Figure 23. Energy Consumption per Square Meter violin

- **Energy Consumption per Square Meter:** Values are concentrated in the lower range, mostly between 50 and 100, indicating lower energy usage in most buildings.

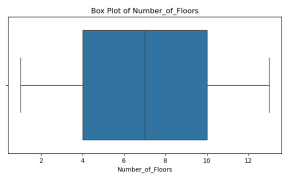

Figure 24. Number of Floors box

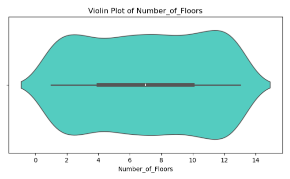

Figure 25. Number of Floors violin

- **Number of Floors:** The data follows a roughly normal distribution centered around 7 floors.

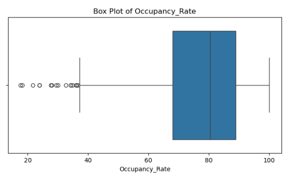

Figure 26. Occupancy Rate box

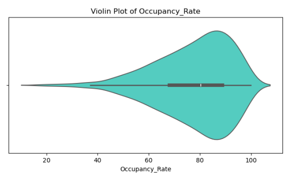

Figure 27. Occupancy Rate violin

- **Occupancy Rate:** The distribution is right-skewed, showing a high concentration of outliers, with most values on the lower end.

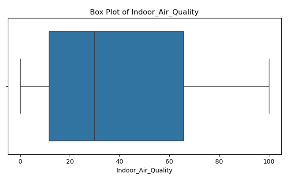

Figure 28. Indoor Air Quality box

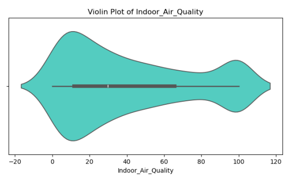

Figure 29. Indoor Air Quality violin

- **Indoor Air Quality:** Skewed to the left, air quality values are concentrated between 10 to 30, with a similar number of values in the 30 to 65 range.

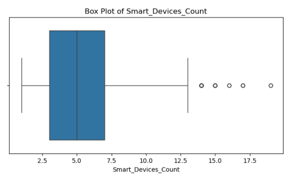

Figure 30. Smart Device Count box

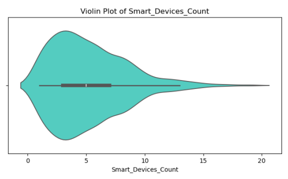

Figure 31. Smart Device Count violin

- **Smart Device Count:** The average is around 5 devices, with a long right tail indicating higher counts in some cases.

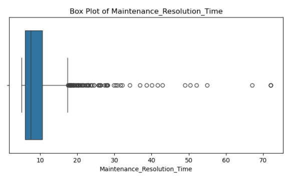

Figure 32. Maintenance Resolution Time box

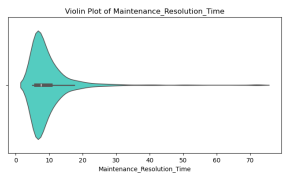

Figure 33. Maintenance Resolution Time violin

- **Maintenance Resolution Time:** This feature shows left-skewness, with most resolution times between 5 to 12, and a long right tail.

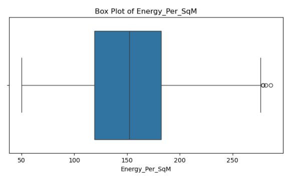

Figure 34. Energy per Square Meter box

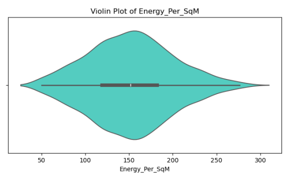

Figure 35. Energy per Square Meter violin

- **Energy per Square Meter:** Slightly skewed towards lower values, with a concentration around 150.

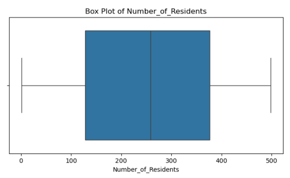

Figure 36. Number of Residents box

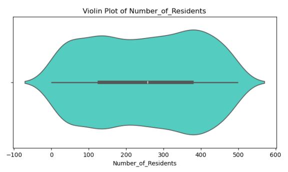

Figure 37. Number of Residents violin

- **Number of Residents:** The distribution leans slightly to the left, with most values clustering around 270 residents.

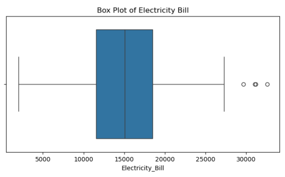

Figure 38. Electricity Bill box

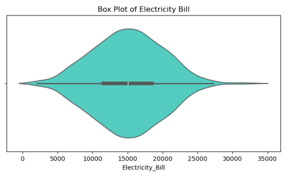

Figure 39. Electricity Bill violin

- **Electricity Bill:**The average bill is around 15,000, with a slight skew to the left, indicating some variability in higher bills.

3.1.2 Categorical Features Count Plots

The count plots of the categorical features show the following:

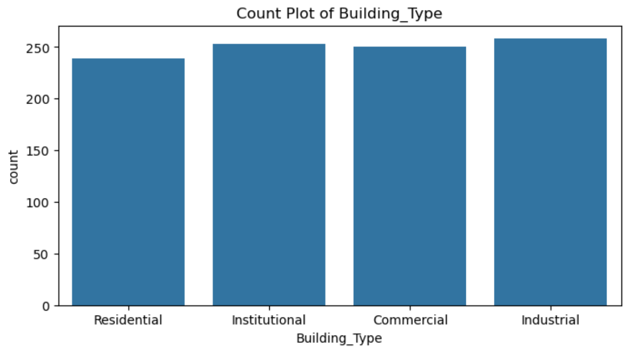

Figure 40. Building Type

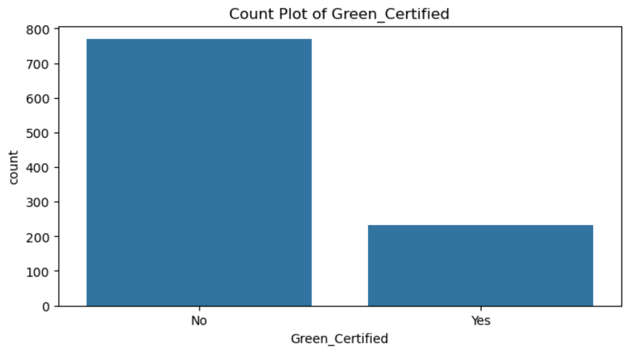

Figure 41. Green Certified

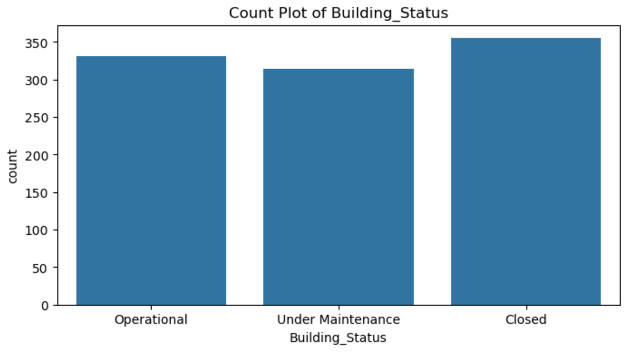

Figure 42. Building Status

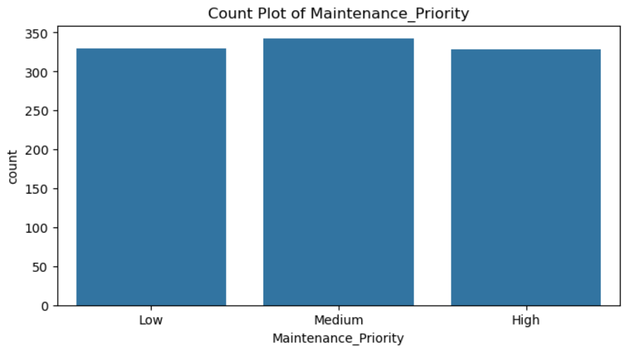

Figure 43. Maintenance Priority

3.1.3 Correlation Heatmap

A correlation heatmap was generated for numerical features, and the following relationships were identified:

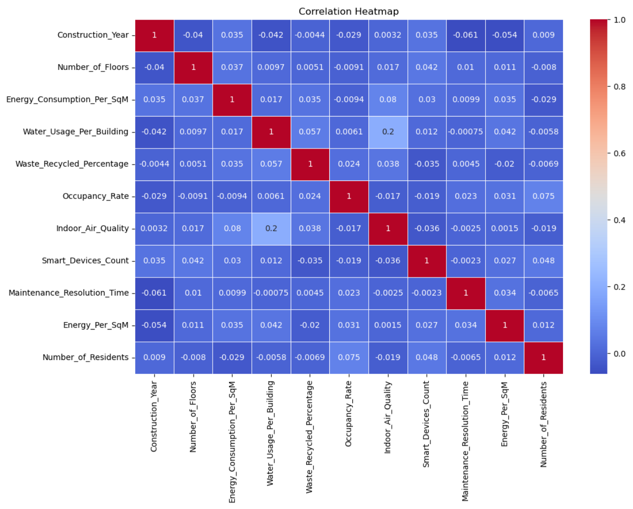

Figure 44. Correlation Heatmap

1. **Occupancy Rate and Number of Residents-0.075:** There is a mild positive correlation between the occupancy rate and the number of residents. This makes

sense as higher occupancy rates generally suggest more people using the building, which would increase the number of residents.

2. **Indoor Air Quality and Water Usage-0.2:** A moderate positive correlation is visible between indoor air quality and water usage per building. This might indicate that buildings with better air quality management also tend to focus on resource management, like water.

3. **Number of Floors and Smart Devices-0.042:** There's a weak positive correlation between the number of floors and the count of smart devices. Taller buildings may require more smart devices for things like security, lighting, and other automated systems.

4. **Energy Consumption per SqM and Indoor Air Quality-0.08:** A mild positive correlation suggests that energy consumption is somewhat linked to indoor air quality. This makes sense as maintaining better air quality often involves HVAC systems, which can consume significant energy.

5. **Maintenance Resolution Time and Construction Year-(-0.061):** A weak negative correlation shows that older buildings (constructed earlier) might have longer maintenance resolution times. This is intuitive because older buildings tend to have more wear and tear, leading to more complex or frequent repairs.

3.1.4 Combined Box and Violin

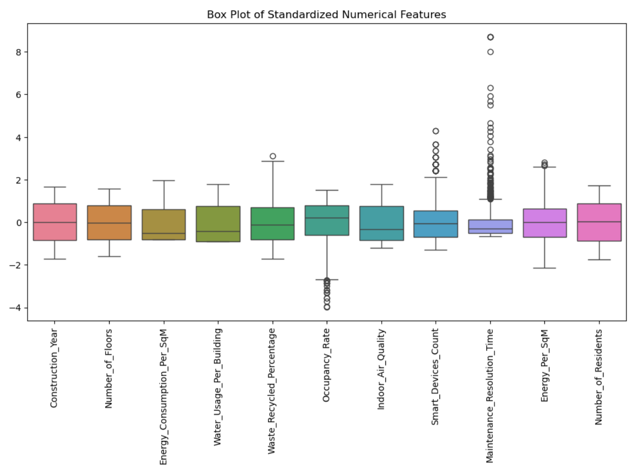

Figure 45. Combined Box Plot

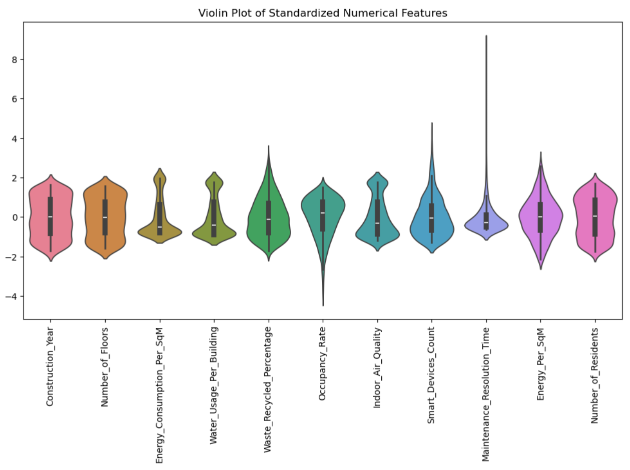

Figure 46. Combined Violin Plot

3.2. Section B: UMAP Dimensionality Reduction

The Uniform Manifold Approximation and Projection (UMAP) algorithm was used to reduce the data dimensions to 2. The UMAP scatter plot shows 2 distinct clusters in two main regions, indicating that the algorithm successfully identified groups of similar data points. However, the target variable (Electricity Bill) was evenly distributed across the clusters, suggesting that other features drove the clustering more than the target variable.

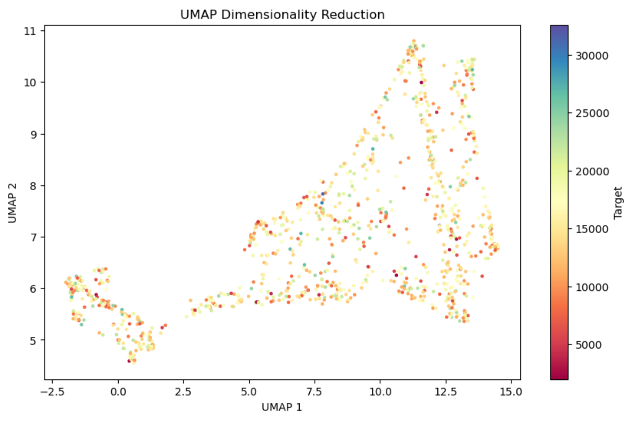

Figure 47. UMAP Dimensionality Reduction Scatter Plot

3.3. Section C: Algorithm Implementation and Model Evaluation

3.3.1 Pre-processing

The dataset was pre-processed by standardizing the numerical features and encoding categorical features using label encoding. No missing values were present in the dataset. The dataset was split into 80% training data and 20% test data for model evaluation.

3.3.2 Linear Regression Model

A linear regression model was trained on the pre-processed data. The performance of the model was evaluated using the following metrics:

Table 5. Model Performance on Test and Train Data

Metric	Test Data	Train Data
MSE	24,278,016.16	24,475,013.17
RMSE	4,927.27	4,947.22
MAE	3,842.41	4,006.33
R2	0.000037	0.0139
Adj R2	-0.064	-0.0493

3.3.3 Model Insights

The results show that the linear regression model's performance is poor, with near-zero or negative R2 scores for both test and train datasets. The MSE and RMSE values indicate significant errors in prediction, suggesting that the model is not well-suited for this dataset in its current form.

3.4. Part d: Feature Selection using Recursive Feature Elimination (RFE)

In this part, we perform Recursive Feature Elimination (RFE) on the dataset to select the three most important features. The selected features are: *Building_Type*, *Green_Certified*, and *Number_of_Residents*. A linear regression model is then trained using only these selected features, and its performance is evaluated on both the training and test datasets. The results obtained are compared with those from Part c, where all features were used for training the linear regression model.

3.4.1 Insights

Feature selection using RFE helps reduce model complexity by selecting only the most important features, potentially improving the interpretability of the model. However, based on the results obtained, the performance metrics, including the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R^2 score, and Adjusted R^2 score, do not show significant improvement when using the selected features. In fact, the model's performance slightly deteriorates when trained with only the selected features, as seen in both the training and test data evaluations.

3.4.2 Comparison of Results (Part c vs. Part d)

Metric	Part c (All Features)	Part d (Selected Features)
Test Data		
MSE	24,278,016.16	23,941,409.06
RMSE	4927.27	4892.99
MAE	3842.41	3813.95
R^2 Score	0.000037	0.01390
Adjusted R^2	-0.06406	0.00188
Train Data		
MSE	24,475,013.17	24,569,032.91
RMSE	4947.22	4956.72
MAE	4006.33	4006.47
R^2 Score	0.01392	0.01013
Adjusted R^2	-0.04929	-0.05332

Table 6. Comparison of Linear Regression Model Performance (Part c vs. Part d)

As shown in Table 7, the metrics for both the test and train data indicate marginal differences between using all features (Part c) and selected features (Part d). In the test data, the model with selected features slightly improves in terms of MSE, RMSE, and MAE, but the R^2 and Adjusted R^2 scores remain low, indicating the model's weak ability to explain the variance in the data. Similarly, for the train data, performance degrades slightly with the selected features.

In conclusion, while feature selection can reduce the complexity of the model, the results suggest that in this case, using all features may provide slightly better performance.

3.5. Part e: One-Hot Encoding and Ridge Regression

In this part, we encode the categorical features using One-Hot Encoding and apply Ridge Regression on the pre-processed data. The Ridge Regression model is evaluated using the same metrics as in Part c: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R^2 score, and Adjusted R^2 score. The results are compared with those obtained in Part c, where Linear Regression was used with all features.

3.5.1 Insights

One-Hot Encoding allows the model to better handle categorical data by converting it into a numerical form without imposing any ordinal relationship between categories. Ridge Regression introduces regularization, which helps prevent overfitting by penalizing large coefficients. From the results, we observe that the Ridge Regression model with One-Hot Encoded features shows a slight improvement in test performance over the standard Linear Regression model from Part c, especially in terms of Mean Absolute Error (MAE) but worse in sense of mse and r2

However, the R^2 score remains low, suggesting that the model still struggles to explain the variance in the data. The Adjusted R^2 score also remains negative for both train and test datasets, indicating that the model is not a good fit.

3.5.2 Comparison of Results (Part c vs. Part e)

Metric	Part c (Linear)	part d(Ridge)
Test Data		
MSE	24,278,016.16	24,297,993.52
RMSE	4927.27	4929.30
MAE	3842.41	3841.97
R^2 Score	0.000037	-0.00079
Adjusted R^2	-0.06406	-0.06494
Train Data		
MSE	24,475,013.17	24,479,024.10
RMSE	4947.22	4947.63
MAE	4006.33	4003.46
R^2 Score	0.01392	0.01376
Adjusted R^2	-0.04929	-0.04946

Table 7. Comparison of Model Performance (Part c vs. Updated Values)

As illustrated in Table 7, the Ridge Regression model with One-Hot Encoded features performs marginally better on both and r2 the train and test datasets in terms of MAE but less in terms of MSE . The R^2 score and Adjusted R^2 score, however, remain low, highlighting that neither model is fully capturing the underlying patterns in the data.

In summary, One-Hot Encoding combined with Ridge Regression offers slight improvements over standard Linear Regression, though the model's overall performance is still limited.

3.6. Part f: Independent Component Analysis (ICA) on One-Hot Encoded Data

In this part, we perform Independent Component Analysis (ICA) on the one-hot encoded dataset and evaluate the model performance by varying the number of components (4, 5, 6, and 8). Linear Regression is applied to the transformed datasets, and the results are compared using various evaluation metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R^2 score, and Adjusted R^2 score.

3.6.1 Insights

ICA is a dimensionality reduction technique that aims to separate a multivariate signal into independent components. When applied to the one-hot encoded dataset, ICA transforms the feature space, and we test how reducing the number of independent components affects the model's predictive performance.

From the results, we observe that the performance does not significantly improve with a higher number of components. In fact, all the models, regardless of the number of components, yield relatively high errors and low R^2 scores. This indicates that ICA may not capture enough of the important structure in the dataset for improving regression performance, at least with the component numbers tried.

The best performance is observed with 4 components, but even here, the improvements are marginal compared to the other component choices.

3.6.2 Comparison of Results with Different Numbers of Components

Metric	4 Comp	5 Comp	6 Comp	8 Comp
Test Data				
MSE	24,167,148	24,204,712	24,253,843	24,221,157
RMSE	4916.01	4917.80	4924.82	4921.60
MAE	3818.89	3816.67	3829.86	3830.14
R^2 Score	0.00460	0.00320	0.00103	0.00234
Adjusted R^2	-0.01165	-0.01400	-0.02363	-0.03078
Train Data				
MSE	24,701,058	24,707,153	24,682,728	24,674,425
RMSE	4970.02	4970.63	4968.17	4967.34
MAE	4010.99	4010.46	4009.41	4009.05
R^2 Score	0.00481	0.00457	0.00555	0.00589
Adjusted R^2	-0.01143	-0.00349	-0.01900	-0.02711

Table 8. Comparison of ICA Performance with Different Components

3.6.3 Inference

The performance metrics show that none of the models significantly outperform the others, with all models yielding similar errors and low R^2 values. In all cases, the R^2 and Adjusted R^2 scores remain near zero or negative, indicating poor model fit.

Comparing the test results across the different components, the model with 4 components performed slightly better in terms of MSE and R^2, but the difference is not substantial enough to consider it superior. The error metrics like RMSE and MAE are also very similar across different models.

In conclusion, ICA does not appear to improve the model's ability to capture the underlying variance in this dataset, suggesting that this approach may not be ideal for this specific task.

3.7. part g)Insights from Test Results using ElasticNet Regularization

The ElasticNet regularization approach combines both L1 and L2 penalties, making it useful for handling high-dimensional datasets. Below is a summary of insights from the test dataset evaluation across different values of α and a comparison with the results from part (c).

3.7.1 Insights from Test Results (ElasticNet)

- **Alpha 0.1**:
  - Mean Squared Error (MSE): 24,297,850.74
  - Root Mean Squared Error (RMSE): 4929.29
  - R2 Score: -0.00078
  - Adjusted R2 Score: -0.06493
  - Mean Absolute Error (MAE): 3841.96

  This suggests that with a small α value, the model is underperforming, showing little improvement in terms of fitting and accuracy.

- **Alpha 0.5**:
  - Mean Squared Error (MSE): 24,325,002.08
  - Root Mean Squared Error (RMSE): 4932.04
  - R2 Score: -0.00190
  - Adjusted R2 Score: -0.06612
  - Mean Absolute Error (MAE): 3839.56

  A small increase in α brings negligible changes, with the performance still remaining negative in terms of the R2 score.

- **Alpha 1**:
  - Mean Squared Error (MSE): 24,328,092.85
  - Root Mean Squared Error (RMSE): 4932.35
  - R2 Score: -0.00203
  - Adjusted R2 Score: -0.06626
  - Mean Absolute Error (MAE): 3838.93

  The model performance does not change significantly, reflecting the limitations of linear fitting for this dataset, with higher penalty values contributing little to model accuracy.

- **Alpha 2**:
  - Mean Squared Error (MSE): 24,326,783.41
  - Root Mean Squared Error (RMSE): 4932.22
  - R2 Score: -0.00197
  - Adjusted R2 Score: -0.06620
  - Mean Absolute Error (MAE): 3838.28

  The slight decrease in MSE and minor changes in MAE and R2 indicate that the higher α values are not improving the model's predictive accuracy.

- **Alpha 5**:
  - Mean Squared Error (MSE): 24,331,970.15
  - Root Mean Squared Error (RMSE): 4932.74
  - R2 Score: -0.00218
  - Adjusted R2 Score: -0.06643
  - Mean Absolute Error (MAE): 3838.82

  With a medium α value, we observe consistent results, with the model still underperforming, and the metrics remaining almost flat.

- **Alpha 10**:
  - Mean Squared Error (MSE): 24,341,595.55
  - Root Mean Squared Error (RMSE): 4933.72
  - R2 Score: -0.00258
  - Adjusted R2 Score: -0.06685
  - Mean Absolute Error (MAE): 3839.99

  The highest α value in this range also fails to improve the model performance. The results reflect that ElasticNet with these α values struggles with this dataset.

3.7.2 Comparison with Part (c)

In part (c), where simpler linear regression models were likely used, the R2 and adjusted R2 scores were also poor, and performance metrics such as MSE and RMSE remained at a similar scale. The ElasticNet method, despite adding regularization, does not significantly improve the prediction performance compared to the original regression model. The slight increase in penalties through different α values has minimal effect on reducing MSE or improving accuracy metrics, suggesting that regularization alone is not sufficient for this dataset.

Conclusion: Both parts exhibit poor model performance, with little improvement in ElasticNet over traditional regression. Further exploration with more complex models or feature selection could be necessary to improve prediction accuracy.

3.8. h)Gradient Boosting Regressor on Preprocessed Dataset

In this subsection, we evaluate the performance of the Gradient Boosting Regressor (GBR) on the preprocessed dataset, using the evaluation metrics of Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R2 score, Adjusted R2 score, and Mean Absolute Error (MAE). We then compare the results obtained with those from parts (c) and (g) using ElasticNet regularization.

3.8.1 Evaluation Metrics for Gradient Boosting Regressor

Test Data Evaluation:

- Mean Squared Error (MSE): 24,293,716.54
- Root Mean Squared Error (RMSE): 4928.87
- Mean Absolute Error (MAE): 3835.23
- R2 Score: -0.00061
- Adjusted R2 Score: -0.06475

Train Data Evaluation:

- Mean Squared Error (MSE): 24,631,779.11
- Root Mean Squared Error (RMSE): 4963.04
- Mean Absolute Error (MAE): 3994.45
- R2 Score: 0.00761
- Adjusted R2 Score: -0.05601

3.8.2 Comparison with Results from Parts (c) and (g)

Comparison with Part (c): In part (c), the model used was likely a simple linear regression model. The R2 score and adjusted R2 score were similarly poor, with no significant predictive accuracy achieved. The performance of GBR shows a slight improvement in the MAE compared to the linear models from part (c), although the R2 scores remain negative for the test data, indicating continued underfitting. GBR manages to capture slightly more of the variability in the training set but does not generalize well to the test set.

Comparison with Part (g) (ElasticNet): In part (g), ElasticNet regularization was applied, which performed similarly poorly in terms of MSE, RMSE, and R2 score, with the best results showing negligible improvement. Comparing the results:

- The best ElasticNet model (with $\alpha = 0.1$) gave a test MSE of 24,297,850.74 and R2 of -0.00078, whereas GBR has a slightly lower MSE of 24,293,716.54 and an R2 of -0.00061.
- Both ElasticNet and GBR perform poorly on the R2 score for test data, though GBR shows marginally better results across all metrics.
- In the training data, GBR produces a slightly better R2 score of 0.00761, whereas ElasticNet models struggled to achieve significant positive R2 scores, indicating slight overfitting by the boosting method.

3.8.3 Conclusion

Although GBR outperforms the ElasticNet model slightly, neither model performs well on this dataset, with both showing negative R2 scores and poor test data prediction. The overall poor performance across all models suggests that the dataset may require more complex feature engineering or nonlinear models for significant improvement.