Indian Institute of Technology, Ropar

AI211

Machine Learning

LAB REPORT 4

LOGISTIC REGRESSION

Student Name	Student ID
Kamakshi Gupta	2023AIB1008

Submission Date : 14/02/2025

1 Introduction

Logistic regression is a statistical model used for binary classification, predicting the probability of an outcome that can have two possible values (e.g., yes/no, 0/1). It uses the logistic (sigmoid) function to map the linear combination of input features to a value between 0 and 1, which represents the probability of the positive class.

2 Methodology

1. DATA PREPROCESSING

   (a) The loan default dataset was imported as a DataFrame using the Pandas library in Python.

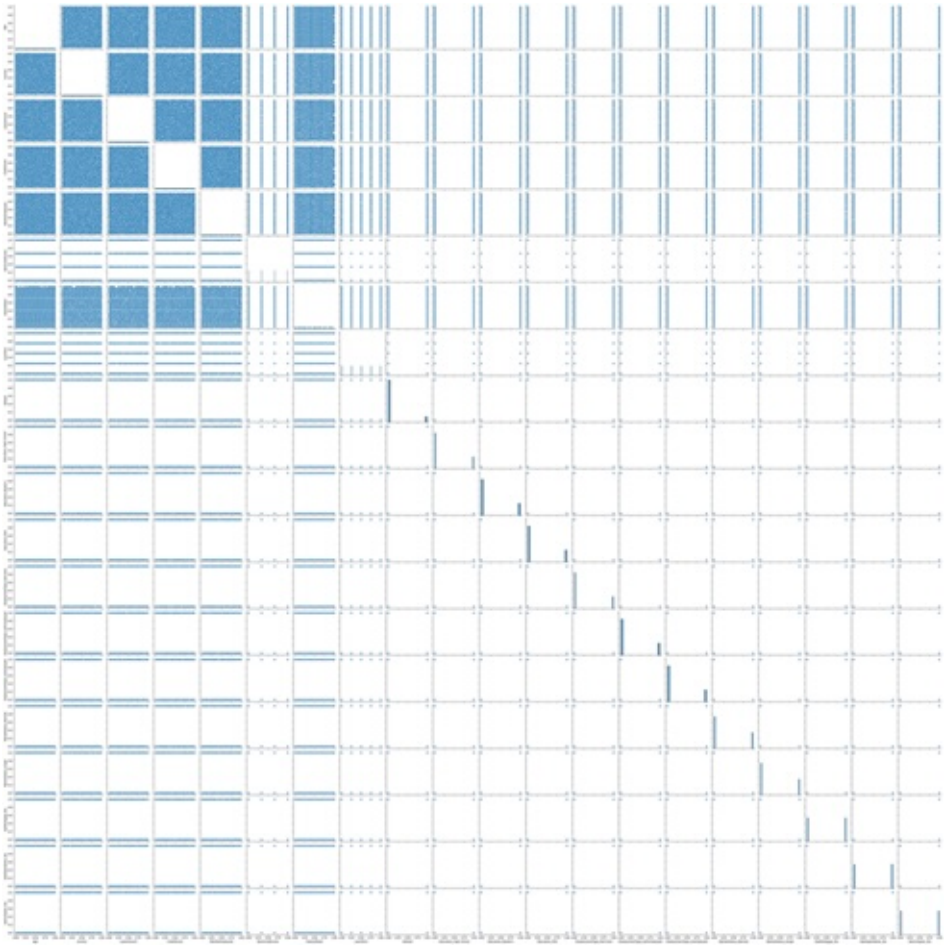
   (b) Dropped the 'LoanID' and 'LoanPurpose' column as it was not relevant to the predictive analysis.

   (c) Converted categorical variables 'Education', 'EmploymentType', 'MaritalStatus', 'HasMortgage', 'HasDependents', and 'HasCoSigner' into dummy variables to represent them as numerical values, ensuring the model can effectively interpret and utilize these features during training.

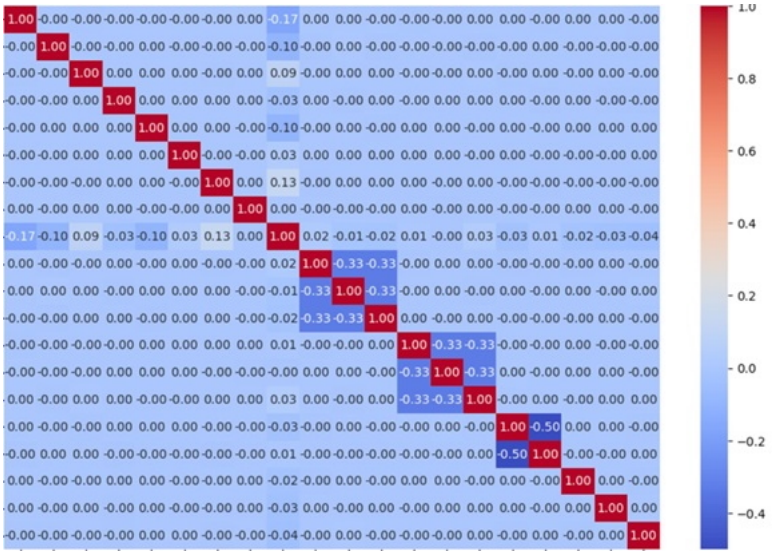
   (d) Dropped the above mentioned columns so that all data was now numerical.

   (e) Checked the skewness of numerical features to identify any significant asymmetry in their distribution. Applied scaling to standardize the data, ensuring consistent magnitude across features for improved model performance.

   (f) Generated pairplots to visualize relationships between features and the target variable, helping to identify patterns, correlations, and potential outliers.



(g) Examined multicollinearity by analyzing the correlation matrix, identifying highly correlated features to avoid redundancy and improve model performance.



2. DATA SPLITTING

(a) Implemented two data splitting strategies: sequential splitting to maintain the order of data and random splitting to ensure randomness and reduce bias, enhancing model evaluation reliability.

(b) Conducted experiments using multiple train-test splits (85-15, 60-40, 50-50, and 30-70) to evaluate model accuracy and robustness.

3. GRADIENT DESCENT VISUALIZATIONS

(a) Stochastic Gradient Descent (SGD) is an optimization algorithm used to minimize the cost function in logistic regression.

(b)
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

(c) Cost Function
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

(d) Update Rule
$$\theta_j := \theta_j - \alpha \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

(e) Decision Boundary
$$\hat{y} = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

(f) Gradient descent is an optimization technique used to find the best values for the parameters of a model, such as the weights and bias in logistic regression. It works by gradually adjusting these parameters to minimize the difference between the model's predictions and the actual values. The process repeats until the model's error is as small as possible, meaning the model is making accurate predictions.

(g) GD Loss Function
$$L(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \right]$$

(h) GD Gradient Descent Update
$$w = w - \alpha \frac{\partial L}{\partial w}$$

(i) GD Bias Update
$$b = b - \alpha \frac{\partial L}{\partial b}$$

(j) Gradient Calculation for weights
$$\frac{\partial L}{\partial w} = (\hat{y} - y)x$$

(k) Gradient Calculation for bias
$$\frac{\partial L}{\partial w} = (\hat{y} - y)x$$

(l) BGD - In Batch Gradient Descent, the model updates the weights and bias using the average gradient calculated over the entire training dataset. This approach ensures a more stable convergence but can be computationally expensive for large datasets.

(m) Gradient Calculation for weights

$$\frac{\partial L}{\partial w} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)x_i$$

4. TESTING

(a) We started out with some pre-defined learning rates and epochs and tested the 3 techniques over the sequential 85-15 dataset split.

(b) SGD and GD had nearly the same result and an accuracy higher than BGD

	Model	Learning Rate	Epochs	Accuracy
0	SGD	0.005	5	0.885100
1	SGD	0.005	10	0.885100
2	SGD	0.005	12	0.885100
3	SGD	0.010	5	0.885048
4	SGD	0.010	10	0.885048
5	SGD	0.010	12	0.885048
6	GD	0.005	5	0.885100
7	GD	0.005	10	0.885100
8	GD	0.005	12	0.885100
9	GD	0.010	5	0.885048
10	GD	0.010	10	0.885048
11	GD	0.010	12	0.885048
12	BGD	0.005	5	0.884578
13	BGD	0.005	10	0.884578
14	BGD	0.005	12	0.884578
15	BGD	0.010	5	0.884578
16	BGD	0.010	10	0.884578
17	BGD	0.010	12	0.884578

3 Comparisons and Analysis

3.1 Confusion matrices by models built from scratch

	Model	Data Split	Accuracy	Confusion Matrix
0	SGD	Seq 85-15	0.885100	[[33709, 173], [4228, 193]]
1	SGD	Seq 60-40	0.883815	[[89950, 217], [11650, 322]]
2	SGD	Seq 50-50	0.884432	[[112296, 455], [14300, 623]]
3	SGD	Seq 30-70	0.884885	[[157548, 422], [20154, 619]]
4	SGD	Random 85-15	0.885727	[[33738, 144], [4233, 188]]
5	SGD	Random 60-40	0.884520	[[89805, 388], [11407, 539]]
6	SGD	Random 50-50	0.884785	[[112309, 499], [14211, 655]]
7	SGD	Random 30-70	0.884644	[[157212, 706], [19913, 912]]
8	GD	Seq 85-15	0.885100	[[33709, 173], [4228, 193]]
9	GD	Seq 60-40	0.883815	[[89950, 217], [11650, 322]]
10	GD	Seq 50-50	0.884432	[[112296, 455], [14300, 623]]
11	GD	Seq 30-70	0.884885	[[157548, 422], [20154, 619]]
12	GD	Random 85-15	0.885727	[[33738, 144], [4233, 188]]
13	GD	Random 60-40	0.884520	[[89805, 388], [11407, 539]]
14	GD	Random 50-50	0.884785	[[112309, 499], [14211, 655]]
15	GD	Random 30-70	0.884644	[[157212, 706], [19913, 912]]
16	BGD	Seq 85-15	0.884578	[[33882, 0], [4421, 0]]
17	BGD	Seq 60-40	0.882787	[[90167, 0], [11972, 0]]
18	BGD	Seq 50-50	0.883116	[[112751, 0], [14923, 0]]
19	BGD	Seq 30-70	0.883783	[[157970, 0], [20773, 0]]
20	BGD	Random 85-15	0.884578	[[33882, 0], [4421, 0]]
21	BGD	Random 60-40	0.883042	[[90193, 0], [11946, 0]]
22	BGD	Random 50-50	0.883563	[[112808, 0], [14866, 0]]
23	BGD	Random 30-70	0.883492	[[157918, 0], [20825, 0]]

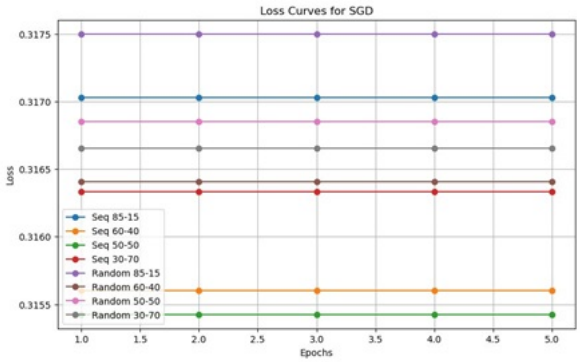
1. All models (SGD, GD, BGD) show similar accuracy, ranging from approximately 88.2% to 88.6%.

2. Stochastic Gradient Descent (SGD) and Gradient Descent (GD) consistently achieve slightly higher accuracy compared to Batch Gradient Descent (BGD).

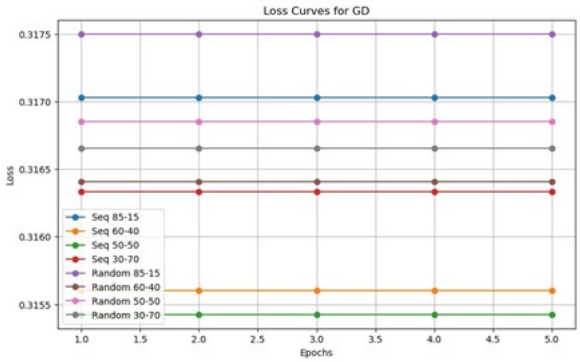
3. Random splits generally show a slight edge in accuracy over sequential splits.

4. BGD consistently reports zero False Positives and False Negatives. This suggests potential issues in learning or prediction, possibly due to poor convergence or model overfitting to the majority class.

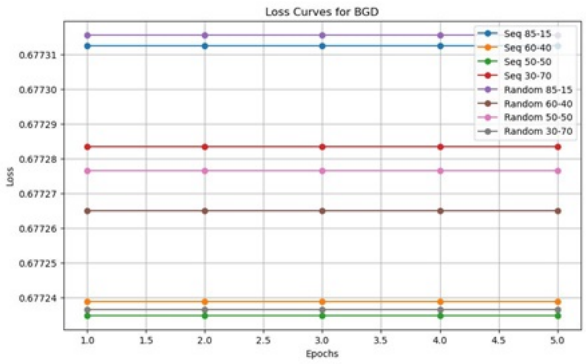
3.2 Loss vs Epochs Visualization



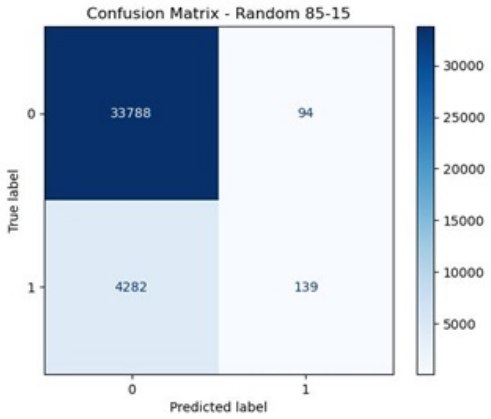
The training loss might appear as a flat line if the model quickly converges to a suboptimal point or fails to learn. The validation loss could also be flat, indicating no improvement or generalization.

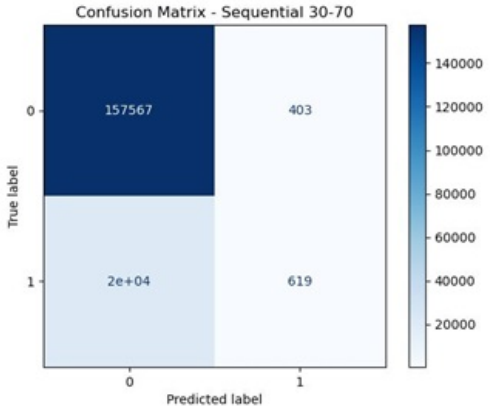


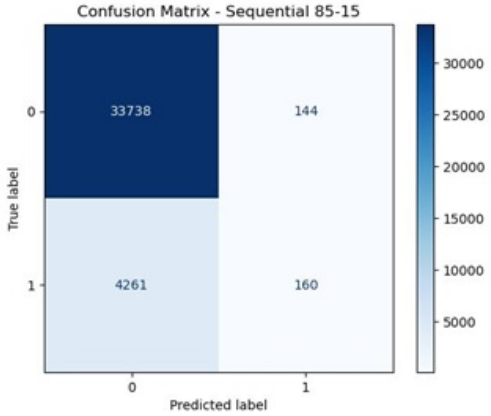
This represents a scenario where the loss remains constant across epochs, indicating no learning or convergence issues.

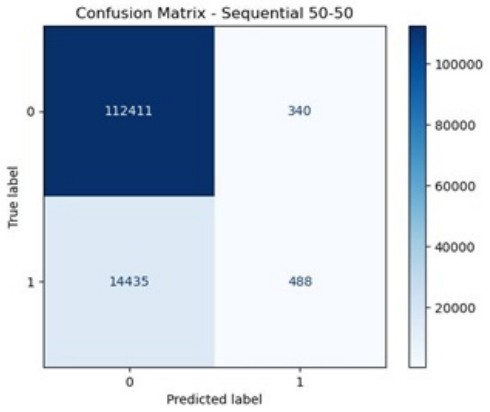


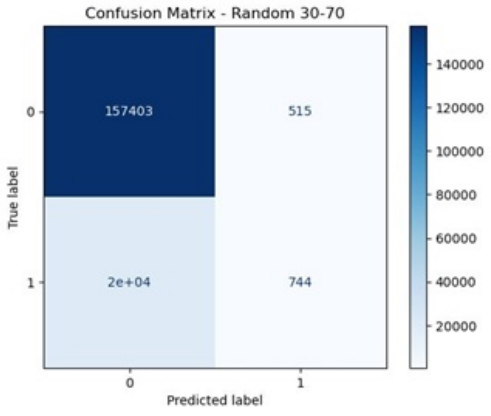
4 Builtin Model Results

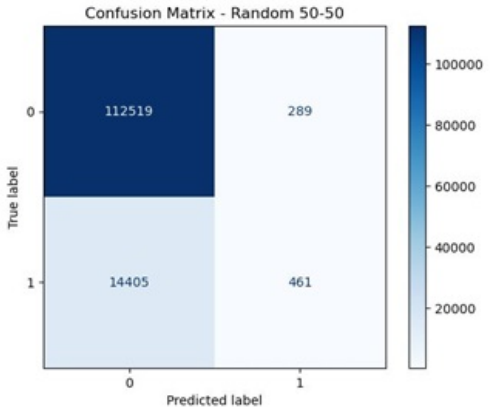


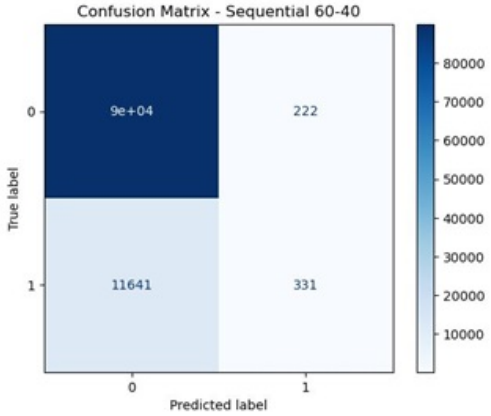


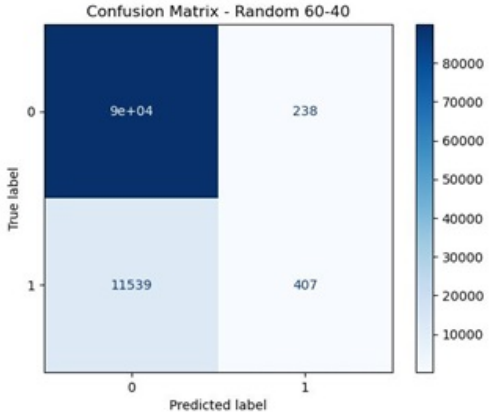












1. Random Splitting: Shows a slightly better balance in True Positives (TP) and False Negatives (FN) compared to Sequential. This is likely due to the random distribution of both classes across training and testing datasets.

2. Sequential Splitting: Consistently has higher False Negatives, indicating that the model struggles to generalize on temporally ordered data, possibly due to temporal dependencies or distribution shifts.

3. The built-in logistic regression model in Scikit-Learn performs reasonably well on the majority class (label 0) but consistently underperforms on the minority class (label 1), especially with sequential splitting. The model is sensitive to train-test ratios, with more balanced splits yielding better generalization. This indicates that the model relies on the distribution of data rather than learning robust features. Addressing class imbalance and experimenting with different train-test strategies may improve performance.

4. The built-in model in Scikit-Learn is ideal for rapid prototyping and reliable performance. However, building from scratch is invaluable for educational purposes, deeper understanding, and experimenting with novel modifications. For practical applications, the built-in model is preferred, whereas custom implementations are suited for research and learning.

	Split	Accuracy	Confusion Matrix
0	Sequential 85-15	0.884996	[[33738, 144], [4261, 160]]
1	Sequential 60-40	0.883854	[[89945, 222], [11641, 331]]
2	Sequential 50-50	0.884276	[[112411, 340], [14435, 488]]
3	Sequential 30-70	0.884991	[[157567, 403], [20154, 619]]
4	Random 85-15	0.885753	[[33788, 94], [4282, 139]]
5	Random 60-40	0.884696	[[89955, 238], [11539, 407]]
6	Random 50-50	0.884910	[[112519, 289], [14405, 461]]
7	Random 30-70	0.884773	[[157403, 515], [20081, 744]]