Lab 4 Assignment Report

Assignment

CS503: Machine Learning

Name	Entry No.
Anshika	2021CSB1069

Instructor: Dr. Santosh Vipparthi

Teaching Assistant: Nishchala Thakur

Indian Institute of Technology Ropar
Punjab, India
February 16, 2025

Contents

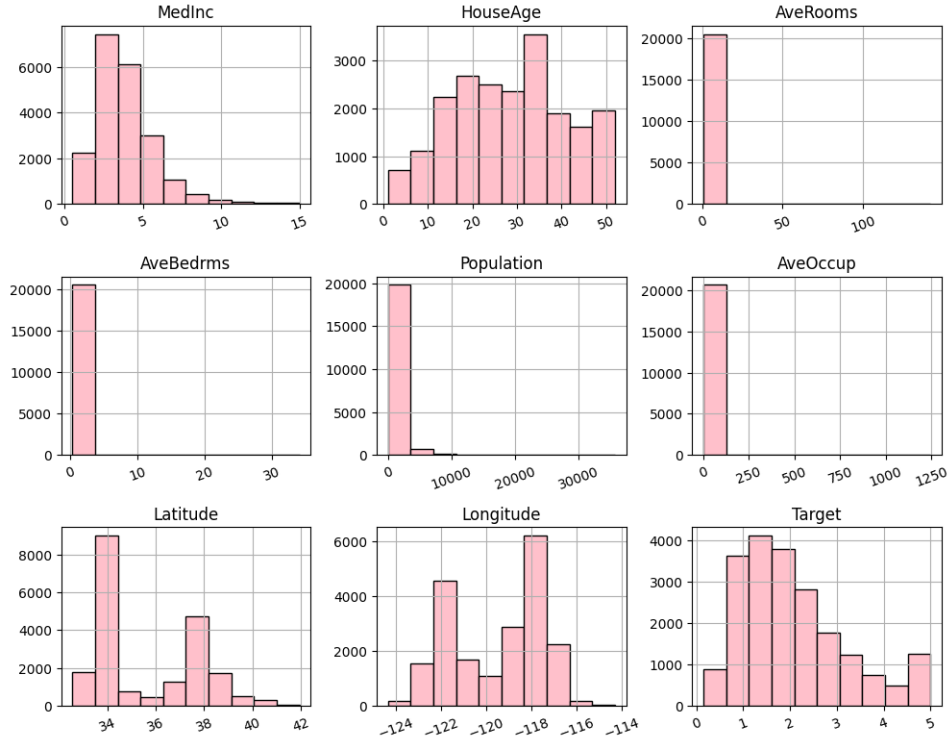
1 | Linear Regression Experiment

Datasets Used:

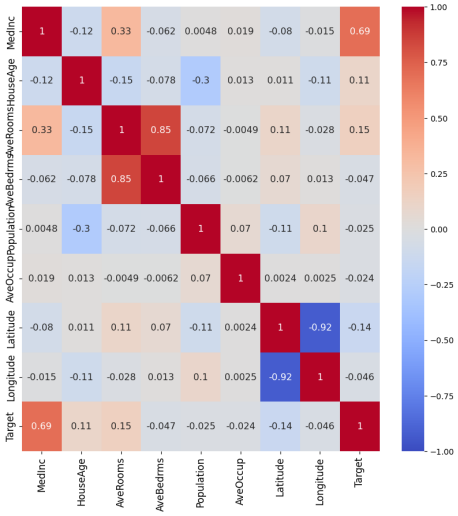
- Diabetes Dataset
- California Housing Dataset

1.1 | EDA on California Dataset

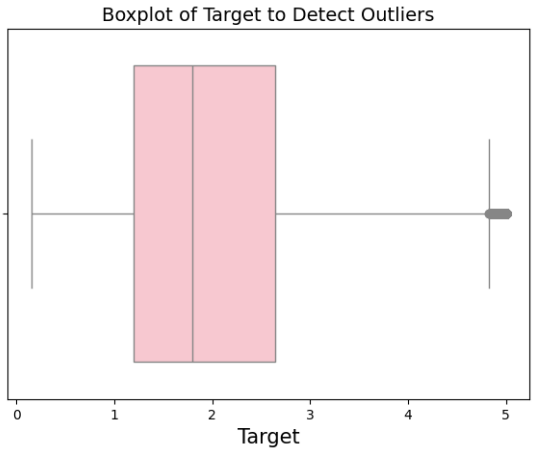
- Frequency Distributions of the independent variables and dependent variable Target



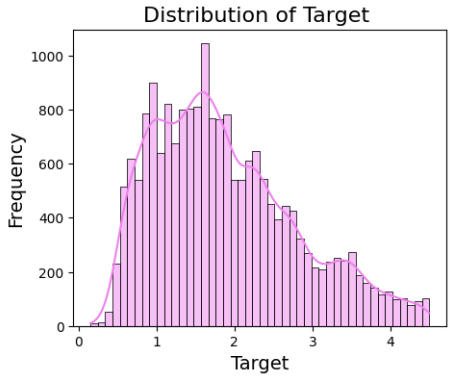
- Heatmap of independent variables to check Correlation



- Outliers detection

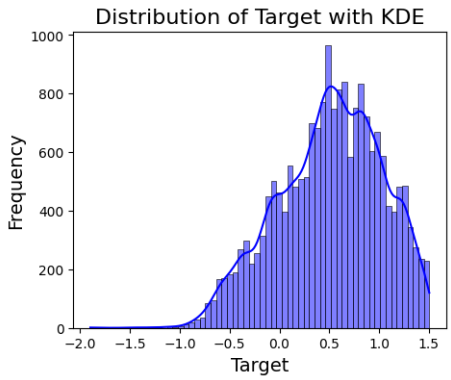


- Skewness of Target Variable
  By Sharpio Test, it is: 0.6834



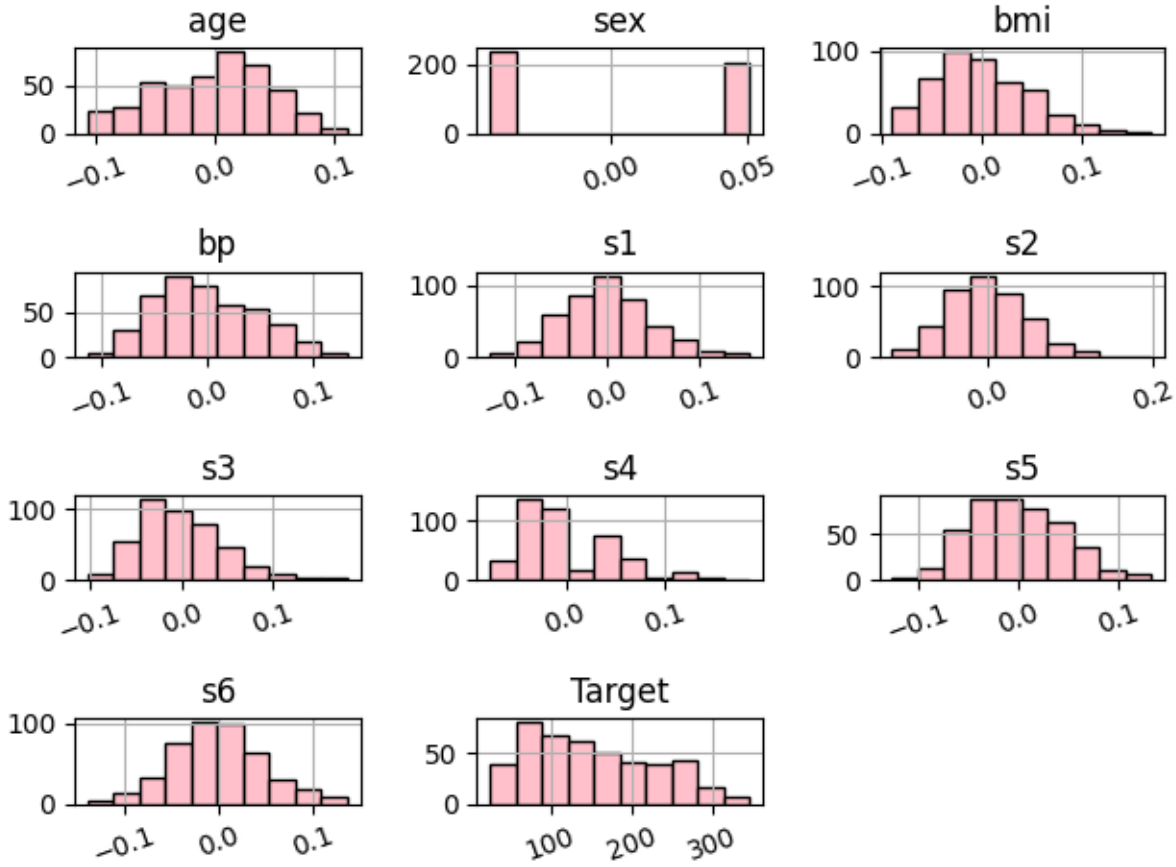
So we do log transformation of dependent variable

- Skewness of Target Variable after Log Transformation
  By Sharpio Test, it is: -0.3480

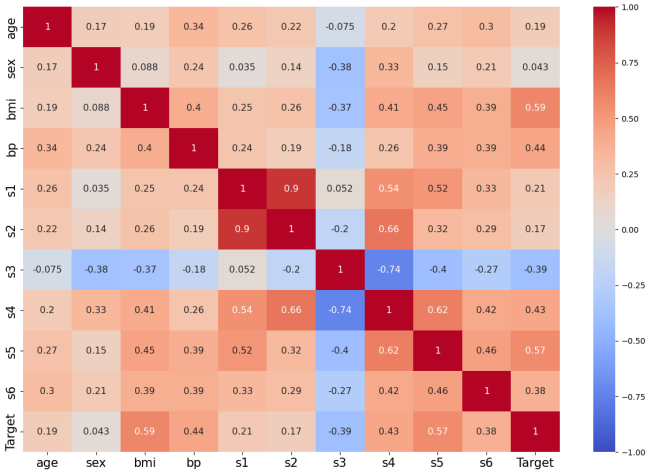


1.2 | EDA on Diabetes Dataset

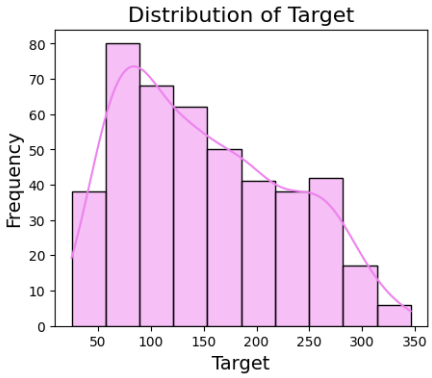
- Frequency Distributions of the independent variables and dependent variable Target



- Heatmap of independent variables to check Correlation

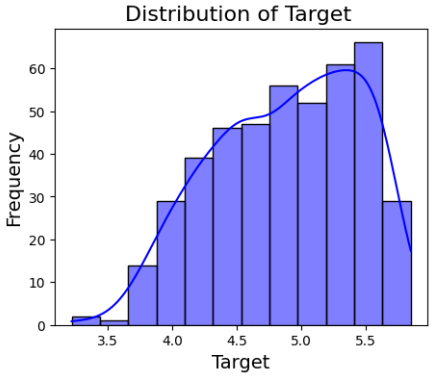


- Skewness of Target Variable
  By Sharpio Test, it is: 0.949058

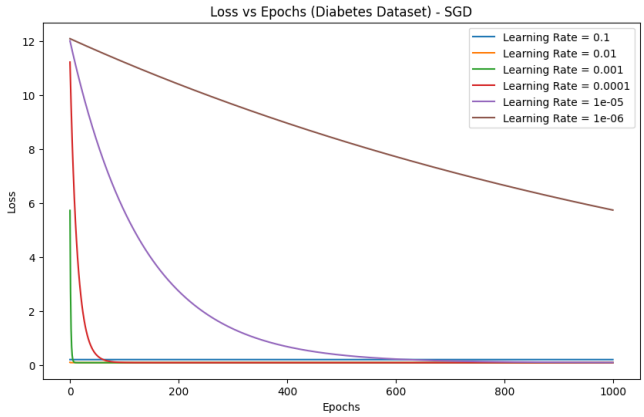


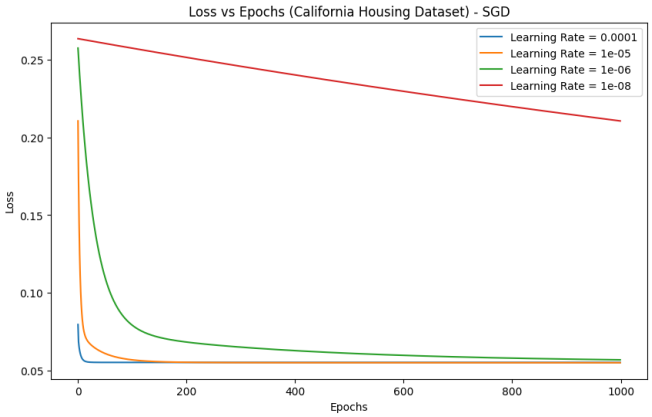
So we do log transformation of dependent variable

- Skewness of Target Variable after Log Transformation
  By Sharpio Test, it is: -0.332



1.3 | Epoch vs Loss for Custom SGD implemented





1.4 | Comparison of Custom SGD and SGDRegressor

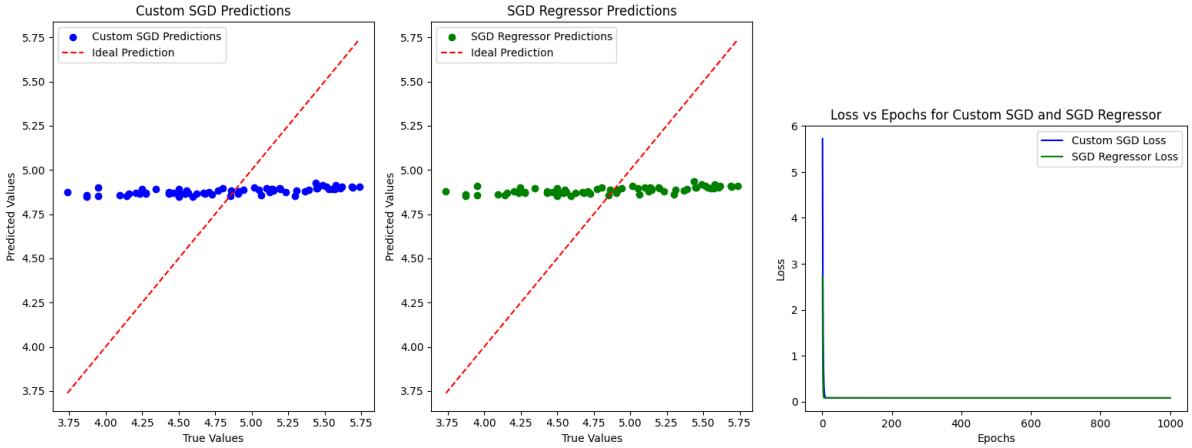
Using the best learning rate obtained with loss vs epoch analysis

Dataset	Metric	Custom SGD	SGDRegressor
Diabetes	RMSE	0.5224	0.5343
	MAE	0.4482	0.4548
	R^2	0.0402	-0.0041
California Housing	RMSE	0.7064	0.7252
	MAE	0.5384	0.6141
	R^2	-0.7946	-0.8911

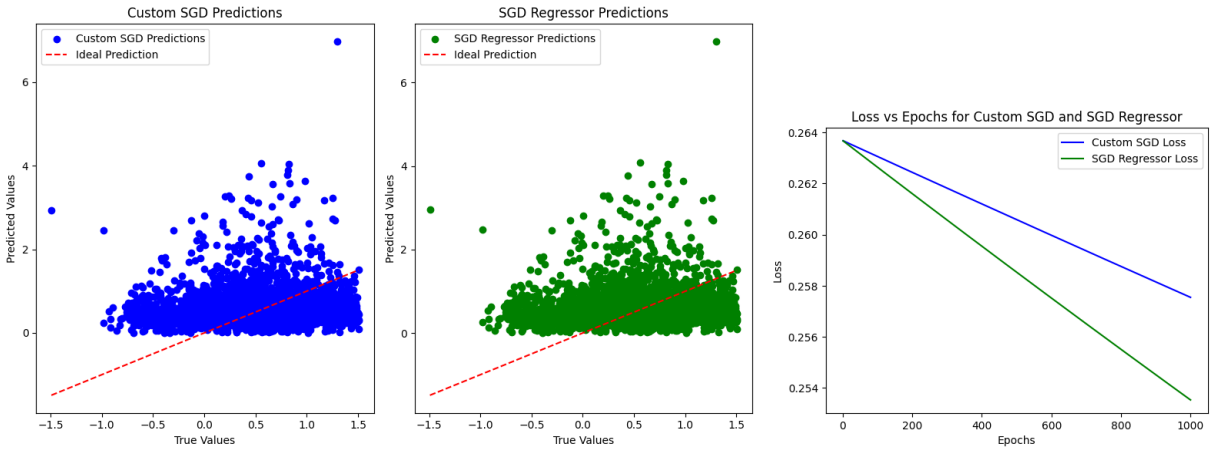
Table 1.1: Comparison of RMSE, MAE, and R^2 for Custom SGD and SGDRegressor on Diabetes and California Housing Datasets.

1.5 | Comparison of Predictions of custom SGD and SGD regressor

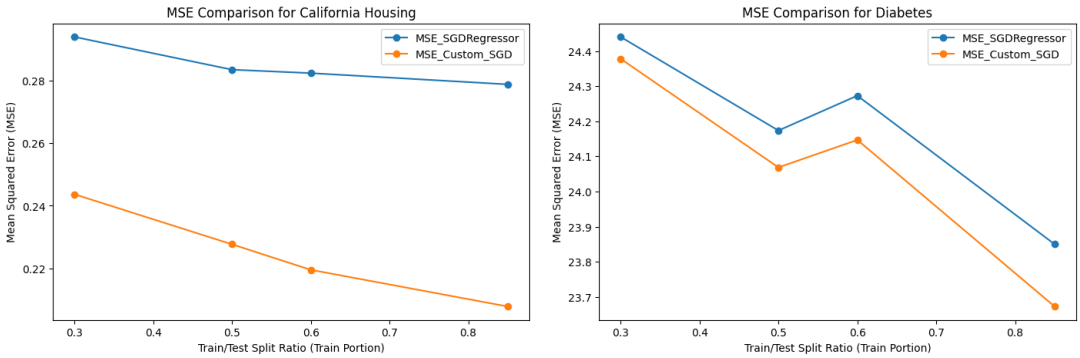
For diabetes dataset



For California dataset



1.6 | Comparison of MSE for various Training splits



Dataset	Train Size	Test Size	Split Ratio	MSE_SGDRegressor	MSE_Custom_SGD
Diabetes	375	67	0.85/0.15	23.849979	23.850090
Diabetes	265	177	0.60/0.40	24.272911	24.272985
Diabetes	221	221	0.50/0.50	24.173676	24.173737
Diabetes	132	310	0.30/0.70	24.440137	24.440176
California Housing	16475	2908	0.85/0.15	0.278811	0.207785
California Housing	11629	7754	0.60/0.40	0.282393	0.219510
California Housing	9691	9692	0.50/0.50	0.283502	0.227666
California Housing	5814	13569	0.30/0.70	0.293975	0.243662

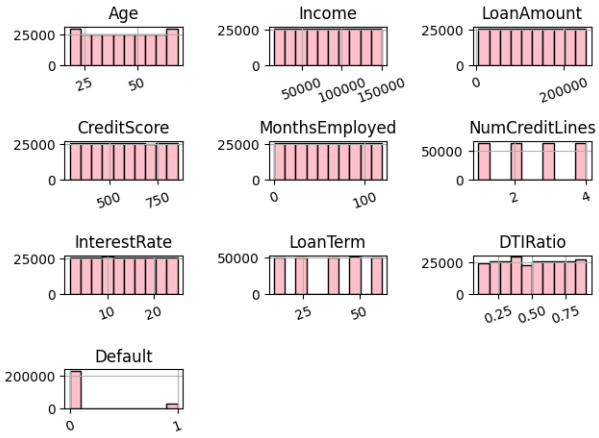
Table 1.2: Comparison of MSE values for SGDRegressor and Custom SGD on Diabetes and California Housing datasets.

2 | Logistic Regression Experiment

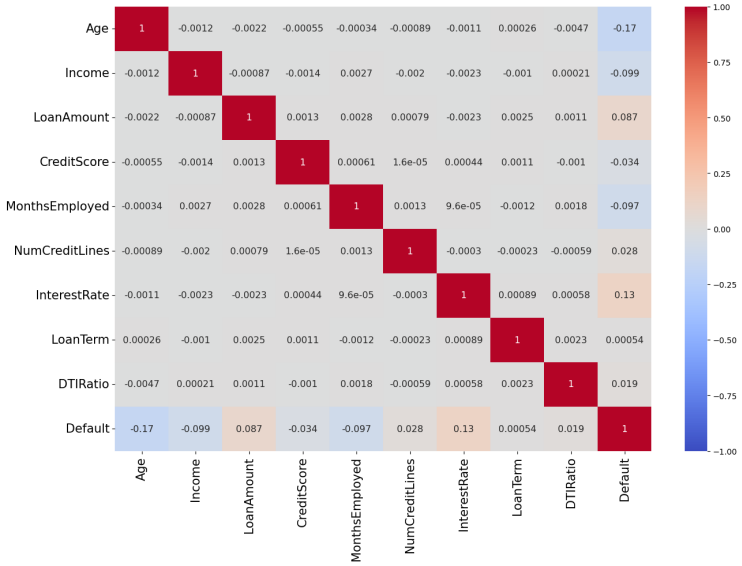
This lab assignment report focuses on implementing Logistic Regression to predict loan defaults using the Loan Default Prediction Dataset.

2.1 | EDA on Loan Default Prediction Dataset

- Frequency Distributions of the independent variables and dependent variable Target



- Heatmap of independent variables to check Correlation



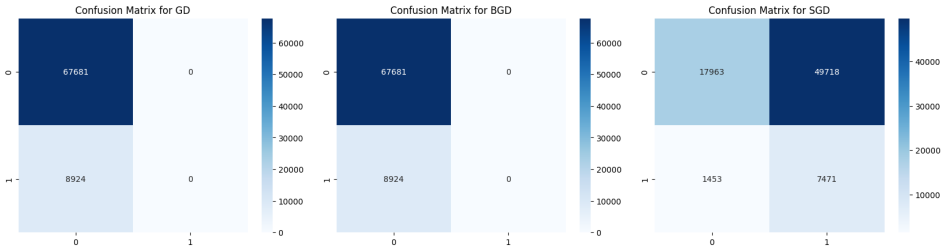
- Then we do dummy encoding of categorical variables.

2.2 | Implementation and Results of SGD, GD, BGD

Method	Gradient Descent	Stochastic Gradient Descent	Batch Gradient Descent
Dataset Size	Small	Large	Medium
Memory Usage	High	Low	Medium
Speed	Slow	Fast	Medium
Convergence	Smooth but slow	May escape local minima	Balanced
Use Case	high accuracy	time/space-constrained	Real-time, faster convergence

Table 2.1: Comparison of Gradient Descent Methods

The following is the comparison of confusion matrices of BGD, GD and SGD Results on learning rate = 0.0001, epochs = 1000, and train/test split is 0.7/0.3. Details are given in 2.2



Model	Train Accuracy	Test Accuracy
GD	0.8840	0.8835
SGD	0.3295	0.3320
BGD	0.8840	0.8835

Table 2.2: Model Performance Summary

2.3 | Methodology of Implementation

2.3.1 |

sectionSigmoid Function
The sigmoid function, $\sigma(z)$, is used to map any real number to a range between 0 and 1, which is common for logistic regression models.
*Steps:

1. The input z is converted to a NumPy array and cast to an integer type for processing.
2. The function returns the sigmoid value:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is the input.

2.3.2 | Gradient Descent (GD)

Gradient Descent is an iterative optimization algorithm used to minimize the loss function by updating the weights.
*Steps:

1. Initialize the weights w as zeros.
2. For each epoch, compute the predictions using the sigmoid of the dot product of input features X and weights w.

3. Compute the gradient of the loss with respect to the weights.
4. Update the weights by subtracting the gradient scaled by the learning rate.
5. Track the loss (cross-entropy loss) at each epoch and print it every 10 epochs.

2.3.3 | Stochastic Gradient Descent (SGD)

SGD updates the model weights based on each individual training example, which makes the updates noisier but can lead to faster convergence.
*Steps:

1. For each epoch, iterate over each training example one by one.
2. For each example, calculate the prediction using the sigmoid function and update the weights based on the prediction error.
3. Calculate the average loss (mean squared error in this case) at each epoch and print it every 10 epochs.

2.3.4 | Mini-Batch Gradient Descent (BGD)

BGD divides the dataset into smaller batches and performs the weight update for each batch, combining the benefits of both GD and SGD.
*Steps:

1. Shuffle the dataset at each epoch.
2. For each batch in an epoch, compute the predictions and gradients for that batch.
3. Update the weights using the gradients calculated from the batch.
4. Compute the overall loss (cross-entropy) using the entire dataset after processing all batches, and track it for every epoch, printing every 10 epochs.

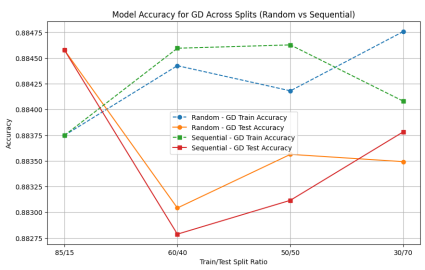
2.3.5 | Key Parameters:

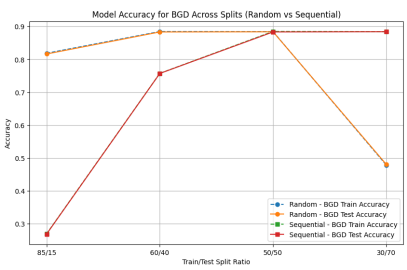
- X: Input feature matrix.
- y: Output labels.
- learning_rate: Step size for weight updates.
- epochs: Number of iterations over the entire dataset.
- batch_size (for BGD): Number of samples in each mini-batch for weight updates.

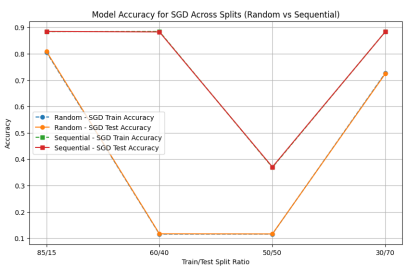
2.3.6 | Key Differences in Algorithms:

- **GD:** Uses the entire dataset to compute the gradient, resulting in stable but slower convergence.
- **SGD:** Updates weights for each individual sample, leading to faster but noisier convergence.
- **BGD:** Balances between GD and SGD by using batches of data to compute updates, reducing noise while increasing computational efficiency.

2.4 | Model Performance Analysis and Visualization







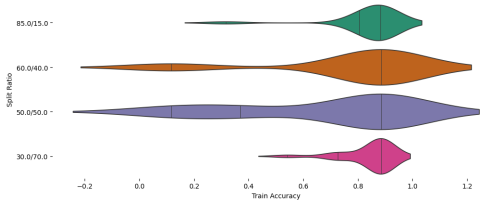
Method	Split Ratio	Strategy	Train Accuracy	Test Accuracy
GD	85.0/15.0	Random	0.883747	0.884578
SGD	85.0/15.0	Random	0.805892	0.809127
BGD	85.0/15.0	Random	0.730115	0.726967
GD	85.0/15.0	Sequential	0.883747	0.884578
SGD	85.0/15.0	Sequential	0.883747	0.884578
BGD	85.0/15.0	Sequential	0.883747	0.884578
GD	60.0/40.0	Random	0.884425	0.883042
SGD	60.0/40.0	Random	0.115575	0.116958
BGD	60.0/40.0	Random	0.884425	0.883042
GD	60.0/40.0	Sequential	0.884595	0.882787
SGD	60.0/40.0	Sequential	0.884595	0.882787
BGD	60.0/40.0	Sequential	0.884595	0.882787
GD	50.0/50.0	Random	0.884181	0.883563
SGD	50.0/50.0	Random	0.115819	0.116437
BGD	50.0/50.0	Random	0.884181	0.883563
GD	50.0/50.0	Sequential	0.884627	0.883116
SGD	50.0/50.0	Sequential	0.368778	0.370490
BGD	50.0/50.0	Sequential	0.884627	0.883116
GD	30.0/70.0	Random	0.884758	0.883492
SGD	30.0/70.0	Random	0.727351	0.724722
BGD	30.0/70.0	Random	0.884758	0.883492
GD	30.0/70.0	Sequential	0.884079	0.883783
SGD	30.0/70.0	Sequential	0.884079	0.883783
BGD	30.0/70.0	Sequential	0.884079	0.883783

Table 2.3: Comparison of Different Methods on Training and Test Accuracies

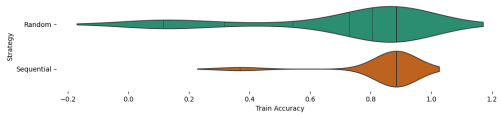
2.5 | Comparison with Inbuilt Logistic Regression

Log Loss vs Epochs for 1000 epochs

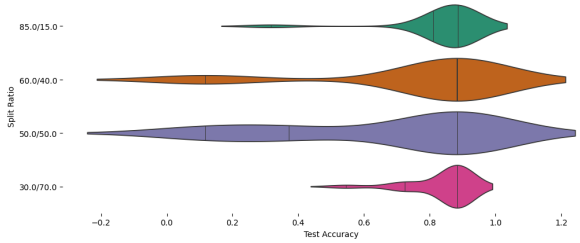
Violinplot Split ratio vs Train accuracy



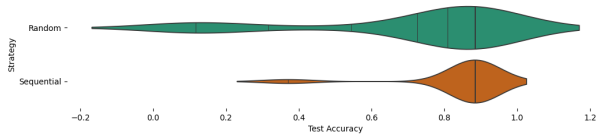
Violinplot Split Strategy vs Train accuracy



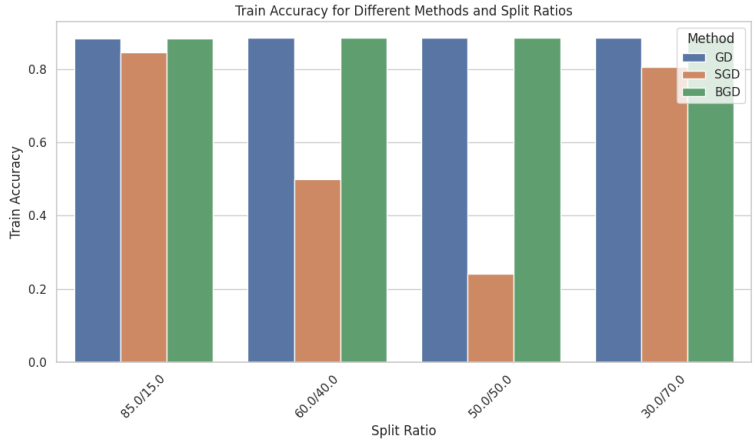
Violinplot Split ratio vs Test accuracy

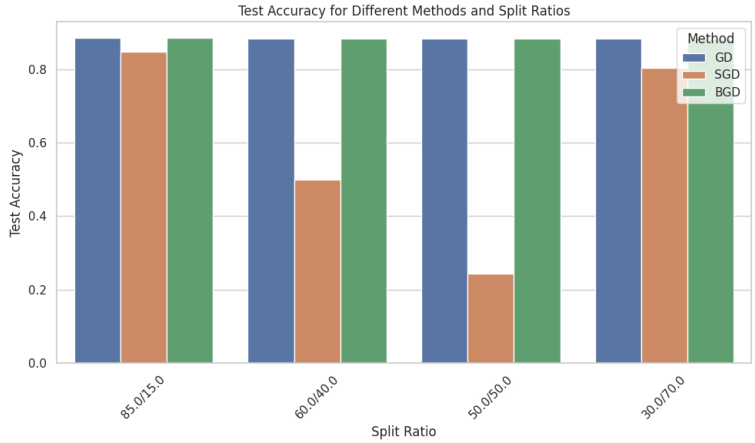


Violinplot Split Strategy vs Test accuracy



Overall Comparative Barplot





Inbuilt logistic regression performs consistently well, but Custome methods show variations depending on split ratio and strategies.

Method	Split Ratio	Strategy	Train Accuracy	Test Accuracy
Inbuilt Logistic Regression	85.0/15.0	Random	0.884775	0.885022
Inbuilt Logistic Regression	85.0/15.0	Sequential	0.884908	0.885361
Inbuilt Logistic Regression	60.0/40.0	Random	0.885248	0.884080
Inbuilt Logistic Regression	60.0/40.0	Sequential	0.885626	0.883639
Inbuilt Logistic Regression	50.0/50.0	Random	0.884995	0.884605
Inbuilt Logistic Regression	50.0/50.0	Sequential	0.885606	0.884150
Inbuilt Logistic Regression	30.0/70.0	Random	0.885646	0.884600
Inbuilt Logistic Regression	30.0/70.0	Sequential	0.885163	0.884762
GD	85.0/15.0	Random	0.883747	0.884578
SGD	85.0/15.0	Random	0.805892	0.809127
BGD	85.0/15.0	Random	0.730115	0.726967
GD	85.0/15.0	Sequential	0.883747	0.884578
SGD	85.0/15.0	Sequential	0.883747	0.884578
BGD	85.0/15.0	Sequential	0.883747	0.884578
GD	60.0/40.0	Random	0.884425	0.883042
SGD	60.0/40.0	Random	0.115575	0.116958
BGD	60.0/40.0	Random	0.884425	0.883042
GD	60.0/40.0	Sequential	0.884595	0.882787
SGD	60.0/40.0	Sequential	0.884595	0.882787
BGD	60.0/40.0	Sequential	0.884595	0.882787
GD	50.0/50.0	Random	0.884181	0.883563
SGD	50.0/50.0	Random	0.115819	0.116437
BGD	50.0/50.0	Random	0.884181	0.883563
GD	50.0/50.0	Sequential	0.884627	0.883116
SGD	50.0/50.0	Sequential	0.368778	0.370490
BGD	50.0/50.0	Sequential	0.884627	0.883116
GD	30.0/70.0	Random	0.884758	0.883492
SGD	30.0/70.0	Random	0.727351	0.724722
BGD	30.0/70.0	Random	0.884758	0.883492
GD	30.0/70.0	Sequential	0.884079	0.883783
SGD	30.0/70.0	Sequential	0.884079	0.883783
BGD	30.0/70.0	Sequential	0.884079	0.883783

Table 2.4: Comparison of Train and Test Accuracy for Inbuilt and Custom Methods

3 | Conclusion

Based on the data, it is evident that both Batch Gradient Descent (BGD) and Gradient Descent (GD) consistently demonstrate better performance compared to Stochastic Gradient Descent (SGD) across different split ratios and strategies (random and sequential). BGD and GD maintain high train and test accuracy values across all configurations, showing their reliability and stability in model training. On the other hand, SGD shows lower accuracy, particularly in random splits with larger data partitions, which highlights its variability and dependence on data order. Therefore, BGD and GD emerge as more robust methods for achieving optimal model performance in this context.