Scilab case study project on Optimisation in Scilab Using the Gradient Descent Algorithm

Naini Diwan

Indian Institute of Science Education and Research, Bhopal

Machine Learning, Statistical Modelling, Data Science

Date: 18-04-2025

Abstract

Gradient descent is an optimization technique designed to minimize a function by iteratively moving in the direction of the steepest descent. The algorithm's simplicity and computational efficiency make it a popular choice for a wide range of models, from linear regression to deep learning networks. This project presents a comprehensive analysis of three gradient descent optimization algorithms—Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD), and Mini-Batch Gradient Descent (Mini-BGD)—applied to logistic regression for binary classification. The study highlights the importance of carefully selecting hyperparameters such as learning rate and batch size, as these can significantly impact the convergence rate and the accuracy of the solution.

Moreover, the study investigates the convergence behaviour and stability of each optimization method. Data standardization is used to enhance convergence speed and stability. The significance of these findings extends to a variety of domains. In deep learning, it is fundamental for optimizing neural network weights and biases. In fields such as economics, game theory, and market equilibrium modelling, it supports the analysis and resolution of complex systems. The findings provide practical insights and recommendations for selecting appropriate gradient descent strategies in logistic regression tasks, emphasizing the importance of algorithm choice, learning rate tuning, and data preprocessing in achieving efficient and robust model training.

1. Introduction

Imagine you're standing at the top of a large, bumpy hill and want to reach the lowest point, but you can't see the entire landscape. What would you do? You'd start by taking small steps in the direction that feels steepest, hoping it leads you downhill. With each step, you'd check if you're getting closer to the bottom, adjusting your path based on the terrain underfoot. This is essentially how gradient descent works: it's a method that takes repeated, careful steps in the direction that most quickly decreases a value, aiming to find the lowest point—or minimum—of a function. Now, consider a faster way to descend the hill. Instead of bringing your entire group of explorers (all your data points) with you on every step, you might send just a few at a time, or even just one. This approach lets you move faster and react more quickly to changes in the landscape. In optimization, this is the idea behind stochastic gradient descent and mini-batch gradient descent: by using only a subset of data for each update, these methods can speed up the journey to the minimum and make the process more efficient. By comparison of algorithms in context of logistic regression for binary classification, the findings are intended to guide the selection of the most suitable method based on dataset characteristics and computational needs.

2. Problem Statement

Many real-world functions encountered in machine learning are non-convex, meaning they may have multiple local minima. For instance, the loss functions from a Convolutional Neural Network (CNN) used in image recognition are so complicated that it is not worthwhile to establish their global minimum. Optimizing the parameters is critical, especially in binary classification tasks. The gradient descent algorithm is an iterative optimization technique that adjusts model parameters to minimize a loss function. The following gradient descent algorithms are exploited:

i) Batch Gradient Descent (BGD)
ii) Stochastic Gradient Descent (SGD)
iii) Mini-Batch Gradient Descent (Mini-BGD)

The convergence properties, computational efficiency, and prediction accuracy of each optimization method are analysed. Finally, we investigate how factors such as learning rate, iteration count, and data standardization affect model performance.

3. Basic concepts related to the topic

- **Cost function**

  It measures the performance of an algorithm for given data set by quantifying the error between predicted and expected values.

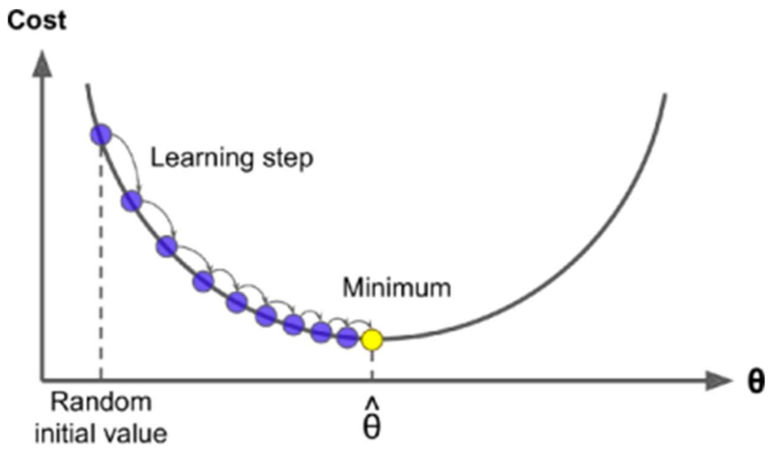
- **Learning rate (λ)**

  The learning rate is a hyperparameter which determines the step size taken in each iteration, influencing the speed and stability of convergence.

- **Gradient Descent**

  Gradient descent is an optimization algorithm that seeks to minimize the cost function by iteratively adjusting the parameters in the direction of steepest slope, with the goal of finding the optimal set of parameters.

  Consider the task of minimizing the value of ω for a given loss function. Initially, ω is assigned random values (random initialization). From there, the algorithm makes gradual improvements by taking small steps, each designed to reduce the loss function, until it converges to the minimum value.



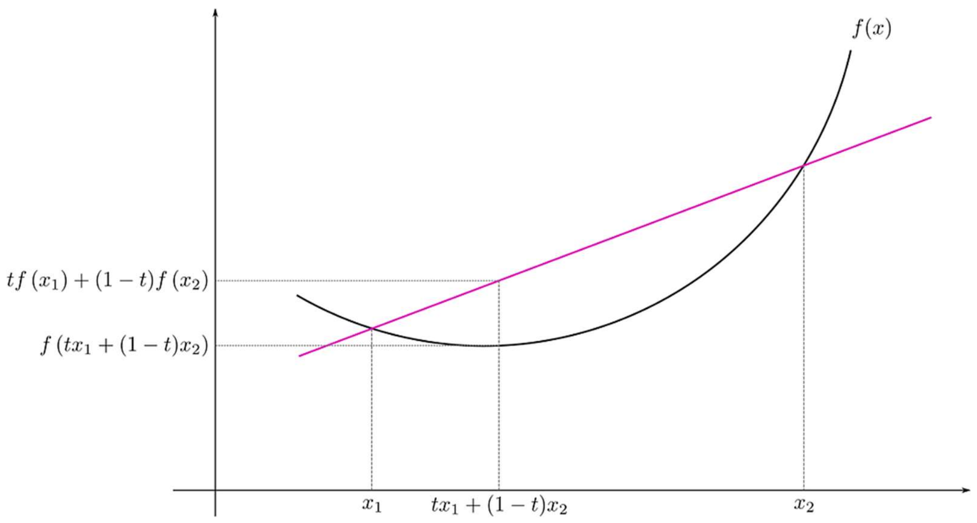
metaphysic.ai

Understanding the algorithm:

$$\boxed{\begin{array}{c}\textit{Repeat until convergence -}\\ \omega^{new} = \omega^{old} - \lambda\dfrac{dL}{d\omega}\end{array}}$$

- **Convex functions**

  Convex functions have a unique global minimum. So, if a convex function is optimised, the best solution is guaranteed by searching for the minimum value of the function. This makes optimization easier and more reliable.



Graph of a convex function : https://en.wikipedia.org/wiki/Convex_function

Mathematically, a function f : R n → R is convex if its domain is a convex set and for all x, y in its domain, and all λ ∈ [0, 1], we must have

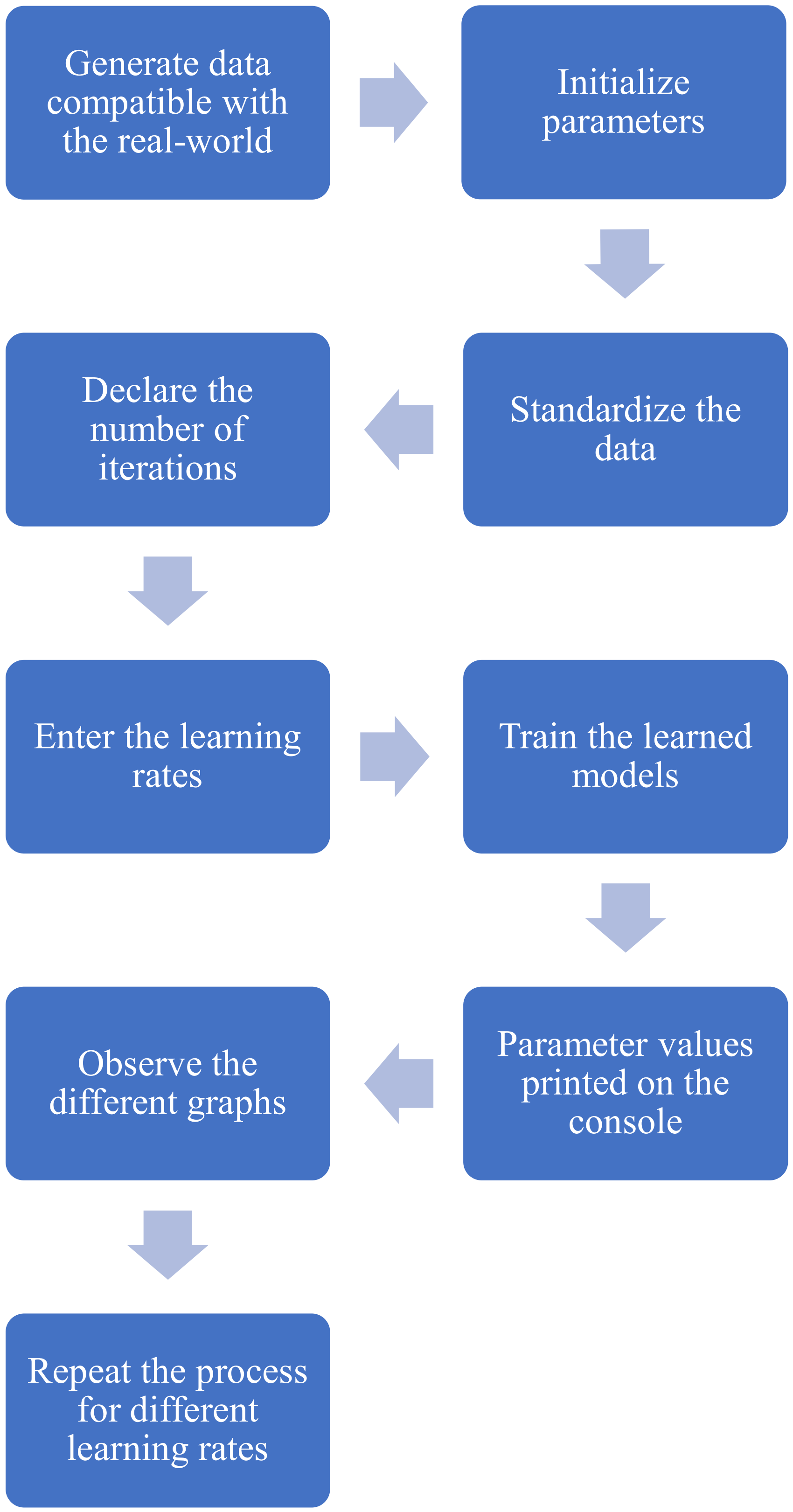
$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

- **Types of Gradient Descent Algorithms**

  Batch gradient descent is ideal for small datasets, it can be computationally expensive and slow, particularly for large datasets. Stochastic gradient descent is better suited for large datasets. SGD randomly selects a training sample, computes the gradient of the cost function for that sample, and adjusts the parameters accordingly. It is computationally efficient and can converge more quickly than batch gradient descent. However, it can be noisy and may not always converge to the global minimum.

  Mini-batch gradient descent offers a balance between the two and is commonly used in practice. Mini-batch gradient descent updates the model's parameters by calculating the gradient based on a small, randomly selected subset of the training data, known as a mini-batch. It computes the average gradient for the mini-batch and adjusts the parameters in the opposite direction.

4. Flowchart


Generate data compatible with the real-world
Initialize parameters
Standardize the data
Declare the number of iterations
Enter the learning rates
Train the learned models
Parameter values printed on the console
Observe the different graphs
Repeat the process for different learning rates

5. Software/Hardware used

- Operating System: Windows 11
- Toolbox: None
- Hardware: Personal Computer with 12th Gen Intel Core Processor, 16GB RAM
- Software: Scilab Version: 2024.0.0 and Microsoft Office 2021

6. Procedure of execution

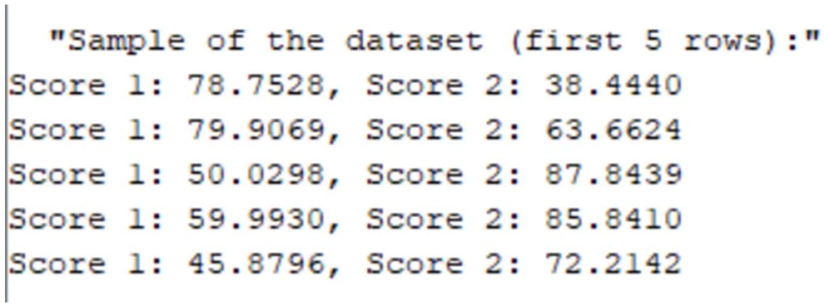
i. Launch Scilab on the computer.
ii. Open the SciNotes file "Grad_Descent_Alg.sce" on Scilab interface .
iii. From the ribbon, select "Save and Execute" .
iv. Observe the plots on the graphic windows.
v. From the console, note the optimum parameter values.
vi. Repeat the process for different hyperparameter values and compare the results.

7. Results

a) Dataset Overview and Model Implementation

The study utilized a dataset containing 100 records, each with two test scores as its features. We implemented a logistic regression model using three different gradient descent methods: Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD), and Mini-Batch Gradient Descent (Mini-BGD).

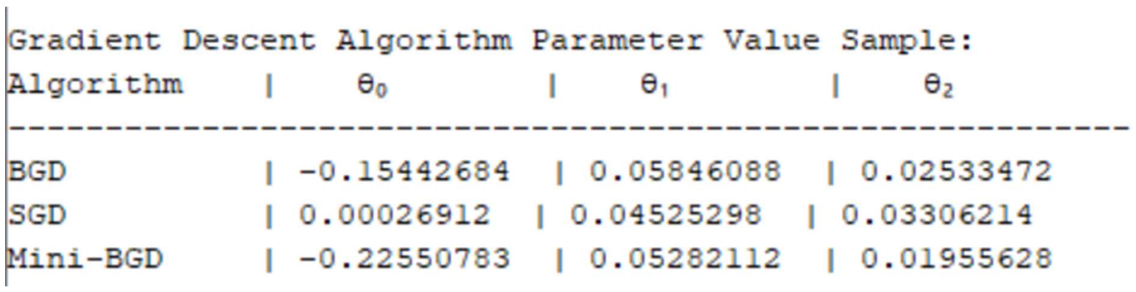
A sample of the dataset is shown below:



```
  "Sample of the dataset (first 5 rows):"
Score 1: 78.7528, Score 2: 38.4440
Score 1: 79.9069, Score 2: 63.6624
Score 1: 50.0298, Score 2: 87.8439
Score 1: 59.9930, Score 2: 85.8410
Score 1: 45.8796, Score 2: 72.2142
```

b) Parameter Convergence Analysis

After running 8000 iterations for each algorithm, we obtained the following parameter values:



```
Gradient Descent Algorithm Parameter Value Sample:
Algorithm    |    θ₀         |    θ₁         |    θ₂
-------------------------------------------------------------
BGD          | -0.15442684   | 0.05846088   | 0.02533472
SGD          | 0.00026912   | 0.04525298   | 0.03306214
Mini-BGD     | -0.22550783   | 0.05282112   | 0.01955628
```

i) SGD with high learning rate introduced significant parameter fluctuations, deviating more from the optimal path compared to its low-rate counterpart.

ii) Standardized data improved parameter consistency across the methods, with Mini-BGD smoother convergence.

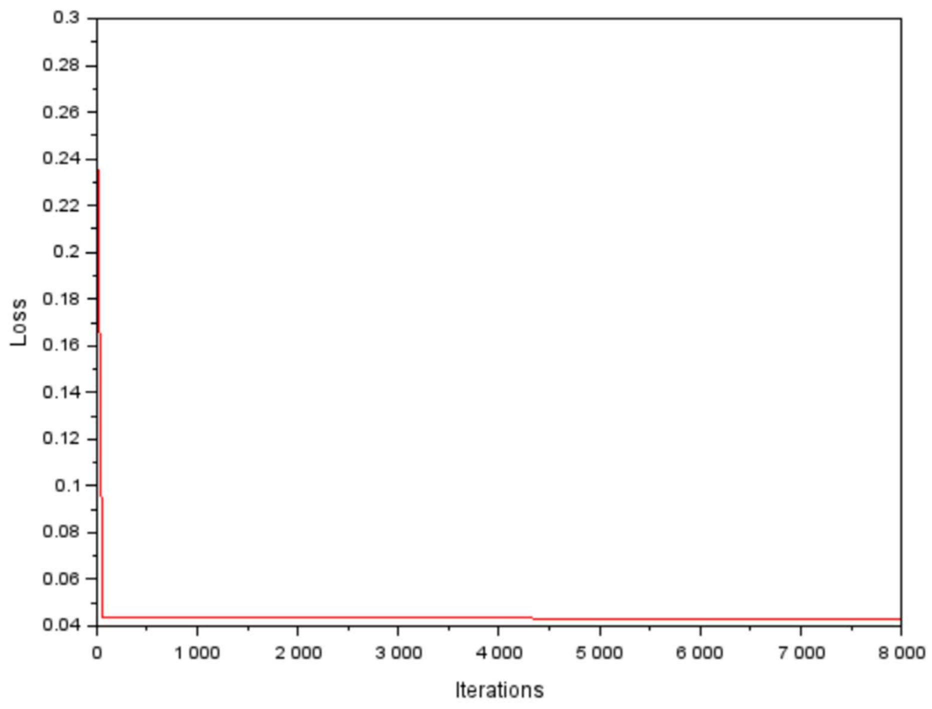
iii) BGD maintained the steadiest parameter trajectory, aligning with its deterministic nature, while Mini-BGD balanced stability and speed by averaging mini-batch gradients.

c) Convergence Speed and Stability Analysis

i) Batch Gradient Descent (BGD)

BGD demonstrated stable convergence with a smooth, monotonic decrease in the loss function. The loss function decreased steadily but slowly, requiring the full dataset for each parameter update. This resulted in a more predictable optimization path but with higher computational cost per iteration.

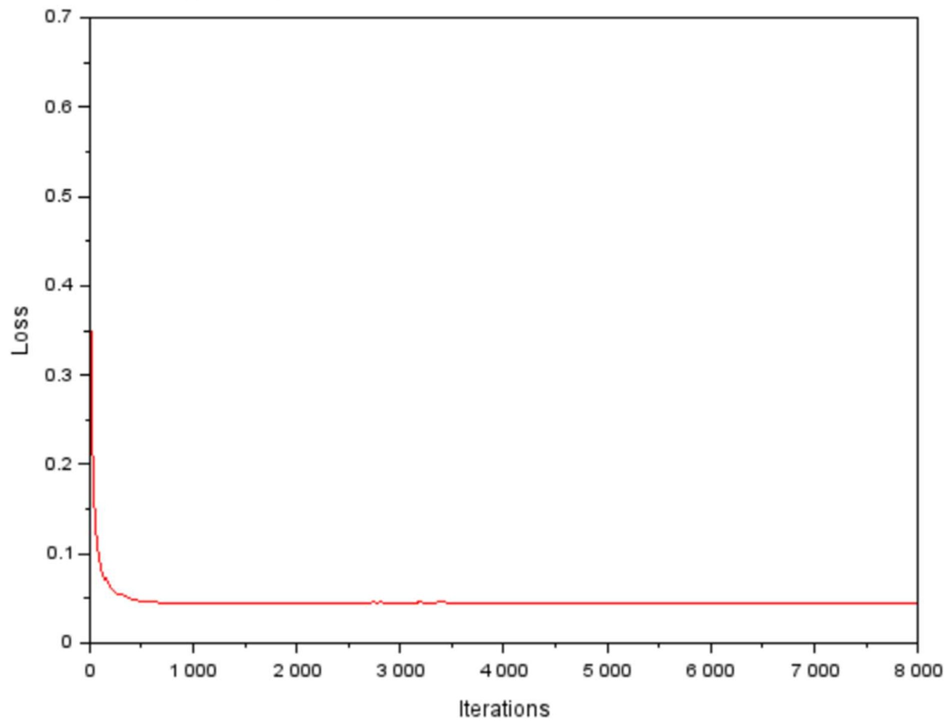
Fig. 1. Diagram of loss and number of iterations under BGD



ii) Stochastic Gradient Descent (SGD)

SGD showed faster initial convergence but with notable fluctuations in the loss function:

Fig. 2. Diagram of loss and number of iterations under SGD

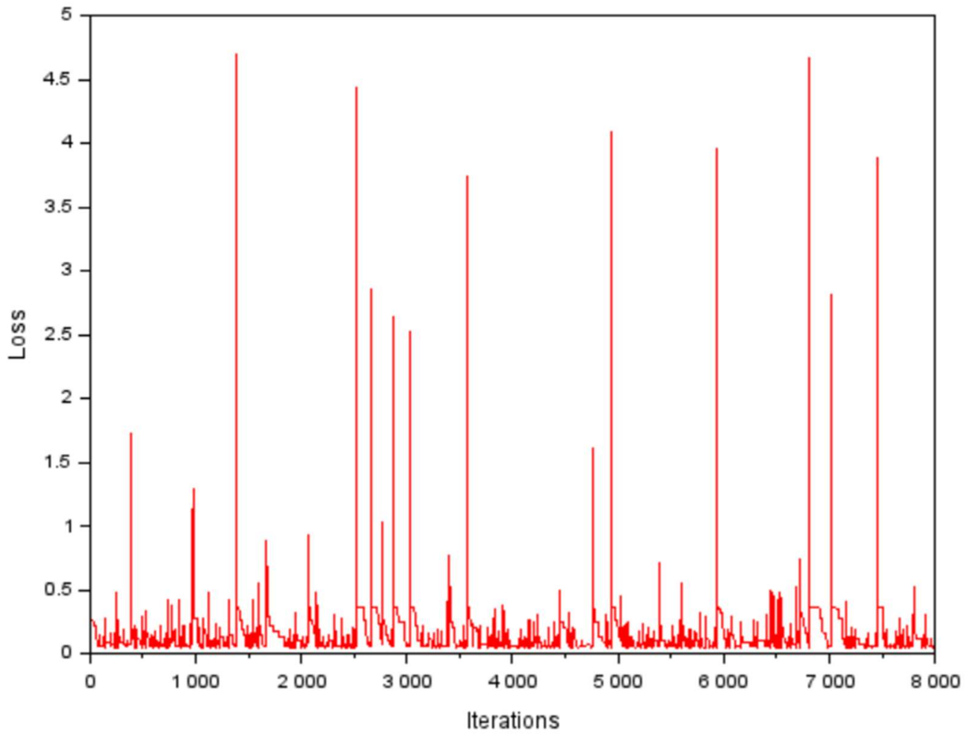


The noisy optimization path is a consequence of updating parameters based on individual samples. Despite the fluctuations, SGD reached lower loss values more quickly than BGD in terms of iterations, though the final result had slightly higher variance.

iii) Mini-Batch Gradient Descent (Mini-BGD)

Mini-BGD with a batch size of 10 demonstrated a balanced approach between BGD and SGD:

Fig. 3. Diagram of loss and number of iterations under Mini-BGD



The convergence pattern showed moderate fluctuations—less pronounced than SGD but more than BGD—while maintaining a faster convergence rate than BGD.

d) Impact of Learning Rate

We experimented with different learning rates for SGD to observe their effects:

Fig. 4. SGD, Iterations=5000, learning rate=0.001

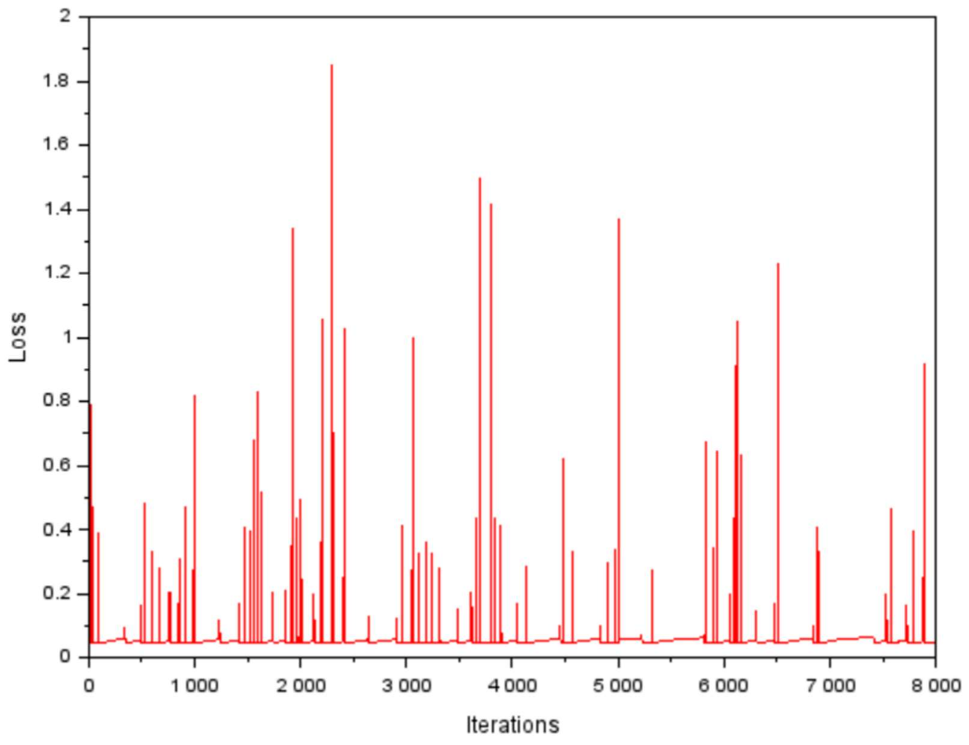
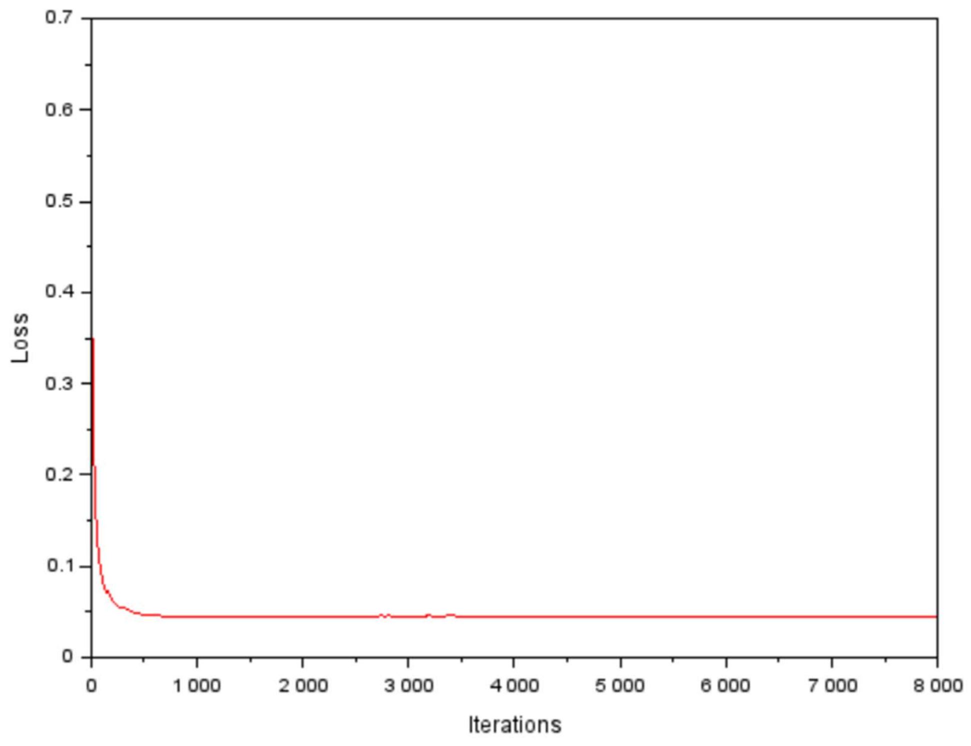


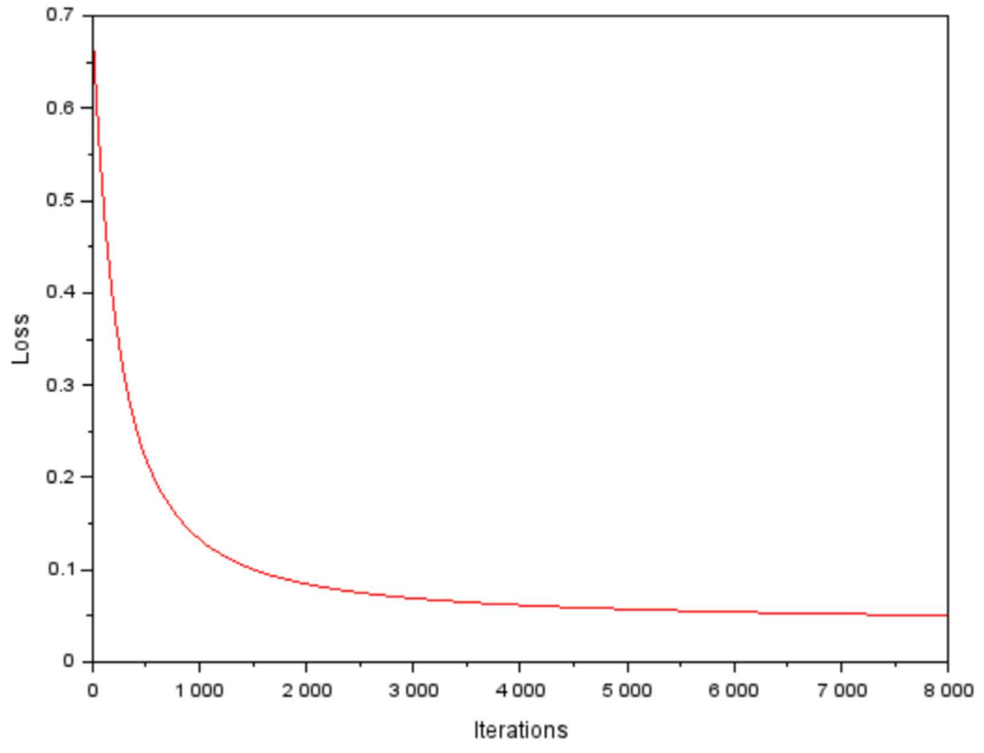
Fig. 5. SGD, Iterations=15000, learning rate=0.00002



e) Effect of Data Standardisation

We also evaluated the impact of standardizing the feature data:

Fig. 6. Diagram of loss and number of iterations under Mini-BGD (Standardized)



Standardization significantly improved the convergence rate and stability for both SGD and Mini-BGD methods. With standardized data, Mini-BGD achieved lower loss values with fewer iterations compared to non-standardized data.

f) Computational Efficiency Comparison

While SGD required the fewest iterations to reach the target loss value, it had higher computational cost per iteration, as compared to BGD due to the overhead of frequent parameter updates. Mini-BGD offered the best balance, requiring fewer iterations than BGD while maintaining reasonable computational efficiency per iteration.

8. Conclusion

Our case study demonstrated that while all three gradient descent variants eventually converge to similar parameter values, they differ significantly in their convergence paths, stability, and computational efficiency. Mini-BGD emerges as a practical compromise between the stability of BGD and the speed of SGD, making it a preferred choice for many machine learning applications, particularly those involving moderately sized datasets

Based on the results, we can make the following recommendations:

1. **For small datasets**: BGD provides stable convergence and may be preferred when computational resources are not a limiting factor.
2. **For large datasets**: Mini-BGD with an appropriate batch size (10-100) typically offers the best balance between convergence speed and stability.
3. **For very large datasets**: SGD with a carefully tuned learning rate can provide efficient training, particularly when computational resources are limited.
4. **Data preprocessing**: Standardization of features is highly recommended as it significantly improves convergence speed and stability for all methods, but especially for SGD and Mini-BGD.
5. **Learning rate selection**: Use larger learning rates for BGD (around 0.01), moderate rates for Mini-BGD (0.001-0.01), and carefully tuned rates for SGD (typically 0.0001-0.001) to balance convergence speed and stability.

The results underscore the importance of algorithm selection based on dataset characteristics, computational constraints, and the specific requirements of the application. By understanding these tradeoffs, practitioners can make informed decisions about which gradient descent variant to use for logistic regression and other machine learning tasks.

9. Challenges encountered

Although gradient descent is an effective optimization algorithm, it comes with trade-offs which include:

- **Local Optima**: Gradient descent may converge to local optima rather than the global optimum, particularly when the cost function contains multiple peaks and valleys.
- **Learning Rate Selection**: The choice of learning rate plays a crucial role in the performance of gradient descent. A learning rate that is too high may cause the algorithm to overshoot the minimum, while a rate that is too low may result in excessively slow convergence.
- **Overfitting**: Gradient descent can lead to overfitting, particularly if the model is overly complex or the learning rate is too large. This can degrade the model's ability to generalize to new, unseen data.
- **Convergence Rate**: For large datasets or high-dimensional spaces, gradient descent can exhibit slow convergence, making it computationally expensive and time-consuming.
- **Saddle Points**: In high-dimensional spaces, the presence of saddle points can cause the gradient of the cost function to become zero, resulting in a plateau where gradient descent fails to move towards a minimum.

To address these challenges, researchers have developed various modifications of the gradient descent algorithm, such as adaptive learning rate methods, momentum-based approaches, and second-order techniques. Additionally, selecting appropriate regularization methods, model architectures, and hyperparameters can significantly enhance the performance of the gradient descent algorithm.

10. References

- https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9603742
- SGDLibrary: For stochastic gradient descent algorithms
- https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0012475&type=printable