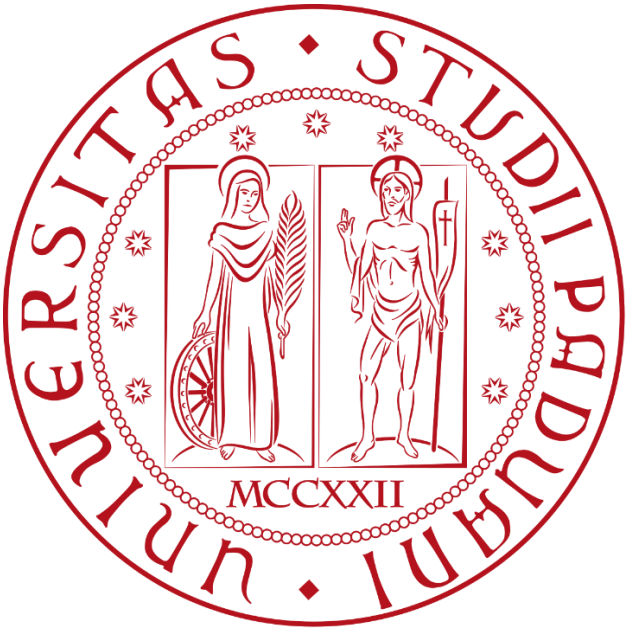

Analysis of Gradient Descent and BCGD with GS rule for Multi-Class Logistic Regression Problems

Optimization for Data Science (2023/2024)

Roya Ghamari

ID: 2071969

Contents

1 Introduction **3**

2 Datasets **3**

3 Methodology **4**

3.1 Multi-Class logistic Regression 4

3.2 Gradient 5

3.3 Accuracy 5

3.4 Loss 6

3.5 Gradient Descent 6

3.6 BCGM Gauss-Southwell 7

4 Implementation **7**

4.1 Preprocessing 8

4.2 Gradient Descent 9

4.3 BCGM Gauss-Southwell 9

5 Results and Comparisons **10**

5.1 Synthetic Data 10

5.2 FMNIST data 13

5.3 IRIS data 14

6 Conclusion **16**

1 Introduction

In this report, the application and comparison of two optimization algorithms, Gradient Descent (GD) and Block Coordinate Gradient Descent (BCGD) with the Gauss-Southwell rule, are investigated on multiple datasets, specifically Fashion-MNIST, Iris, and a custom-generated synthetic dataset. The focus is on multi-class logistic regression problems, where each method is applied to handle datasets with varying characteristics and complexity.

The synthetic dataset was created to systematically analyze the behaviors of GD and BCGD under controlled settings, adjusting parameters such as the learning rate and the number of iterations to refine performance. The Fashion-MNIST dataset, which consists of clothing items, serves as a more complex benchmark commonly used in machine learning to evaluate algorithm performance due to its diversity and real-world applicability. Additionally, the simpler and smaller Iris dataset is used to examine the scalability of these methods for problems with fewer features and classes.

Throughout the study, the experimental setup, implementation specifics, and results obtained are detailed, providing a comprehensive analysis of the performance differences between GD and BCGD. This includes insights into training efficiency, accuracy, and convergence behaviors of the loss function under varying conditions. The findings aim to contribute valuable insights into the effective application of these fundamental optimization techniques in machine learning, particularly in the context of multi-class logistic regression.

2 Datasets

In the presented work, three datasets are used for analysis as follows:

1. A synthetic dataset to simulate a multi-class classification scenario. The dataset comprises a feature matrix A with dimensions $m \times d$, where m is the number of samples and d represents the number of features. Each element in the feature matrix is generated from a standard normal distribution $N(0,1)$. Additionally, a parameter matrix x of size $d \times k$ is created, with elements drawn from a normal distribution $N(0,1)$, where k is the number of classes. To introduce variability and mimic real-world data noise, a noise matrix e, of the same size as the score matrix $m \times k$, is added to the linear combination of A and x. Class labels for each sample are assigned based on the highest value in the noisy linear combination, reflecting the most likely class.

2. The Fashion-MNIST introduced by: Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv preprint arXiv:1708.07747.

This dataset is a collection of article images designed as a more challenging replacement for the traditional MNIST dataset for benchmarking machine learning algorithms. It consists of 70,000 grayscale images, divided into a training set of 60,000 images and a test set of 10,000 images. Each image is 28x28 pixels, representing one of 10 fashion categories, including T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

3. The Iris dataset introduced by: Fisher, R.A. (1936). "The use of multiple measurements in taxonomic problems." Annals of Eugenics, 7(2), pp. 179-188.

This dataset consists of 150 samples from each of three species of Iris flowers (Iris setosa, Iris virginica, and Iris versicolor). Four features were measured from each sample: the lengths and the widths of the sepals and petals.

3 Methodology

A standardized experimental setup was established to ensure comparability across different datasets and optimization algorithms. Each dataset was split into training and testing sets with an 80-20 ratio. A uniform number of iterations, set at 200, was employed for each run to maintain consistency in the evaluation of the optimization processes and also because the trends of the algorithms can be clearly seen given the chosen number of iterations. Furthermore, the initial model parameters for each dataset were standardized using the same initial values derived from a normal distribution, thereby ensuring that any observed differences in algorithm performance were attributable to the algorithms themselves and not to initial conditions.

A fixed step size rule for the learning rate was applied across all experiments. Since the goal was comparing the given algorithms, other optimization methods regrading to stopping condition and step size have been skipped in this study. This approach simplifies the comparison by removing dynamic adjustments to the learning rate that could potentially obscure the intrinsic differences between Gradient Descent (GD) and Block Coordinate Gradient Descent with the Gauss-Southwell rule (BCGD). The use of a fixed learning rate provides a clear basis for evaluating the efficiency and effectiveness of each optimization strategy under controlled conditions.

This section outlines the overarching methodology used to compare the efficacy of Gradient Descent (GD) and Block Coordinate Gradient Descent with the Gauss-Southwell rule (BCGD) across various datasets. The primary goal was to assess these optimization techniques under uniform experimental conditions, which involved a consistent setup across different types of data. By maintaining standardized testing protocols, the study aims to evaluate the inherent capabilities and performance variations between the two methods for different datasets. Details on specific implementation strategies, parameter configurations, and the experimental setup are further elaborated in Section 4, where the practical aspects of applying these algorithms are discussed.

More details about the methods for calculating the gradient, accuracy, and loss functions, as well as the specific implementations of Gradient Descent and BCGD Gauss-Southwell, will be expounded upon in subsequent subsections of the methodology.

3.1 Multi-Class logistic Regression

Given a dataset with samples belonging to multiple classes, the objective is to model the probability that a given sample belongs to a particular class based on its features. This is achieved through a logistic model that generalizes the binary logistic regression to multiple classes.

In this project, the multi-class logistic regression model is formulated to handle classification tasks where each input sample can belong to one of several classes. The objective is to minimize the logistic loss function across all samples, So the problem is in the form of:

$$\min_{x} \sum_{i=1}^{m} \left(-x_{b_i}^T a_i + \log\left(\sum_{c=1}^{k} e^{x_c^T a_i} \right) \right)$$

where:

- m is the number of samples.
- a_i represents the feature vector of the i-th sample.
- x_c is the parameter vector corresponding to class c.
- b_i is the true class label for the i-th sample.
- k denotes the total number of classes.

The model estimates the probability that a given sample belongs to a specific class using the softmax function, which is expressed as:

$$p(b_i | a_i, x) = \frac{e^{x_{b_i}^T a_i}}{\sum_{\acute{c}=1}^{k} e^{x_{\acute{c}}^T a_i}}$$

Here, $p(b_i | a_i, x)$ denotes the predicted probability of the i-th sample belonging to class c.

3.2 Gradient

The gradient of the loss function with respect to the parameters for each class $x_c$is crucial for optimizing the logistic regression model. It is computed as follows:

$$\frac{\partial L(x)}{\partial x_{jc}} = -\sum_{i=1}^{m} a_{ij} \left[I(b_i = c) - \frac{e^{x_c^T a_i}}{\sum_{\acute{c}=1}^{k} e^{x_{\acute{c}}^T a_i}} \right]$$

Where $I(b_i = c)$ is an indicator function that equals 1 if $b_i = c$ and 0 otherwise.

3.3 Accuracy

In this project, accuracy is used to assess the performance of the logistic regression model by comparing the predicted class labels against the true labels and calculating the percentage of correct predictions. The process involves a softmax function which converts logits (model outputs before activation) into probabilities by normalizing the exponential of each logit. Then we determine the predicted class for each instance as the one with the highest probability from the softmax output. Accuracy is calculated as the mean of instances where the predicted labels match the true labels.

This method provides a straightforward metric to evaluate the model's ability to correctly classify instances across multiple classes, serving as a crucial indicator of performance in the multi-class logistic regression framework.

3.4 Loss

In this project, the loss function used is the cross-entropy loss, which quantifies the difference between the true labels and the predicted probabilities of the logistic regression model. This function is pivotal in evaluating model performance and guiding the optimization process. The process involves:

1. **Logit Calculation**: Computes the matrix product of feature matrix A and the parameter matrix x, resulting in logits.
2. **Softmax Application**: Transforms logits into probabilities using the softmax function. This normalization ensures that the output probabilities are non-negative and sum to one across the classes for each instance.
3. **Cross-Entropy Computation**: Calculates the negative log-likelihood of the true class probabilities. For each sample, it focuses on the probability assigned to the correct label, emphasizing the model's confidence in the actual outcome.
4. **Loss Aggregation**: Averages the negative log likelihoods over all instances to get the overall loss, which the model aims to minimize during training.

The cross-entropy loss function for multi-class logistic regression can be expressed as:

$$L(x) = \sum_{i=1}^{m} \left(-x_{b_i}^T a_i + \log \left(\sum_{c=1}^{k} e^{x_c^T a_i} \right) \right)$$

This formula captures the penalization for incorrect predictions, emphasizing correct predictions with high confidence, and is the foundation for computing gradients used in optimizing x.

3.5 Gradient Descent

In this project for the multi-class logistic regression problem, the gradient descent method is applied to optimize the parameters based on the log-likelihood function, adjusted for multiple classes. The method progresses through these key steps:

Initialization: The model parameters x are initialized with random values drawn from a normal distribution. This sets the starting point for the optimization process.

Gradient Computation: At each iteration, the gradient of the loss function with respect to the parameters x is computed. This gradient indicates the direction of the steepest ascent in the loss landscape.

Parameter Update: The model parameters are updated by moving in the opposite direction of the computed gradient. This is achieved using the formula:

$$x \leftarrow x + \alpha \nabla_x L(x)$$

Where α is the learning rate, a small positive scalar that controls the step size in the gradient descent update.

Iterative Optimization: The process repeats for a fixed number of iterations, refining the parameters progressively to reduce the loss. At specific intervals, the algorithm logs metrics such as accuracy on the training and test datasets, and the value of the loss function, to monitor the optimization progress.

3.6 BCGM Gauss-Southwell

The BCGD method relies on partitioning the parameter matrix x ∈ $R^{d \times k}$ into smaller blocks or individual columns corresponding to each class. This approach exploits the structure of the optimization problem to enhance computational efficiency.

Gradient Calculation: For each class c, compute the gradient of the loss function with respect to the corresponding column x_c of the parameter matrix. This gradient reflects how the parameters for class c should be adjusted to minimize the loss function.

Block Selection via Gauss-Southwell Rule: Instead of updating all parameters simultaneously, the Gauss-Southwell rule selects the block (or column) of x corresponding to the class with the highest gradient norm. This strategy targets the most significant update at each iteration, potentially speeding up convergence:

$$c^* = arg \max_c \left\| \nabla_{x_c} L(x) \right\|$$

Parameter Update: Once the block is selected, the parameters for that class are updated using a step in the direction of the negative gradient scaled by the learning rate.

The selective update mechanism of BCGD makes it highly efficient, especially for problems where the number of parameters is large and not all parameters contribute equally to the model's performance. By focusing computational resources on the most impactful parameters, BCGD can achieve faster convergence and improved scalability.

4 Implementation

A standardized experimental setup was established to ensure comparability across different datasets and optimization algorithms. Each dataset was split into training and testing sets with an about 80-20 ratio. A uniform number of iterations, set at 300, was employed for each run to maintain consistency in the evaluation of the optimization processes and also because the trends of the algorithms can be clearly seen given the chosen number of iterations. Furthermore, the initial model parameters for each dataset were standardized using the same initial values derived from a

normal distribution, thereby ensuring that any observed differences in algorithm performance were attributable to the algorithms themselves and not to initial conditions.

A fixed step size rule for the learning rate was applied across all experiments. Since the goal was comparing the given algorithms, other optimization methods regrading to stopping condition and step size have been skipped in this study. This approach simplifies the comparison by removing dynamic adjustments to the learning rate that could potentially obscure the intrinsic differences between Gradient Descent (GD) and Block Coordinate Gradient Descent with the Gauss-Southwell rule (BCGD). The use of a fixed learning rate provides a clear basis for evaluating the efficiency and effectiveness of each optimization strategy under controlled conditions. However, to facilitate a more comprehensive comparison, different learning rates were also tested in some cases, providing additional insights into the algorithms' behavior under varied conditions.

4.1 Preprocessing

1. Synthetic Data: For the synthetic dataset, the preprocessing step primarily involved the generation and partitioning of the data into training and testing sets. This dataset was constructed to provide a controlled environment for testing the optimization algorithms. Using a seed for random number generation ensures consistency in the data across different runs.

The feature matrix A and parameters x were generated from a normal distribution N(0,1). Noise e, also drawn from a normal distribution N(0,1), was added to introduce variability and simulate real-world data conditions. Class labels b were determined based on the highest scores from the linear combination of A and x, plus noise, which is typical in multi-class classification tasks.

Train-Test Split: The dataset was partitioned into an 80-20 train-test split, aligning with common practices to validate models on unseen data effectively.

2. Fashion-MNIST Data: for FMNIST dataset, preprocessing steps were crucial in preparing the image data for effective model training:

Flattening and Normalization: Each image in the dataset, originally structured in a 28x28 pixel format, was flattened into a 784-dimensional vector to simplify processing. The pixel values were then normalized by dividing by 255, transforming them into a scale between 0 and 1. This normalization is essential for smoothing the learning process, as it ensures that all input features contribute equally to the model's predictions.

Data Reduction: Due to computational constraints, a subset of the data was used—10,000 samples for training and 2,000 for testing. This reduction maintains a manageable workload while still providing sufficient data variability for robust algorithm testing.

Label Processing: The target labels, representing different clothing items, were directly utilized from the dataset without further modification, as they were already in a suitable format for multi-class classification tasks.

3. Iris Data: For the Iris dataset, the preprocessing steps are straightforward and tailored to prepare the data for effective model training:

Feature Standardization: The features of the dataset are standardized using *StandardScaler*. This step normalizes the feature set to have zero mean and unit variance. Standardization is crucial for many machine learning algorithms, including logistic regression, as it aligns the scales of input features, leading to improved convergence during optimization.

Train-Test Split: The standardized dataset is split into training and testing sets with a ratio of 80% for training and 20% for testing. This split is executed, ensuring a random selection of instances for each subset, controlled by setting the random state to 42 for reproducibility. The training set includes 120 samples, while the testing set comprises 30 samples, maintaining a balanced approach to both learning the model parameters and validating the model's performance on unseen data.

4.2 Gradient Descent

Monitoring and Logging: At regular intervals (every 10 iterations) and at the final iteration, the algorithm logs the current state of the model. Metrics such as training and testing accuracy and loss are computed to monitor the progress of the optimization. These metrics are crucial for understanding the model's performance and determining whether the model is improving and converging towards a minimum.

Convergence Check: The convergence of GD can be assessed by observing changes in the loss function. If changes become negligible between iterations, it suggests that the model is close to finding a minimum, assuming the learning rate is adequately tuned.

Output: Upon completing all iterations, the function outputs the optimized parameters and a detailed log of the training process, including accuracies and losses at specified intervals, providing insights into the model's learning trajectory and overall effectiveness.

4.3 BCGM Gauss-Southwell

Iteration: The algorithm iteratively updates one block of parameters at a time based on the Gauss-Southwell selection criterion.

Specifically for the Fashion-MNIST dataset, once the block gradient is computed, it undergoes "clipping" to limit its magnitude. This prevents excessively large updates that can lead to unstable training dynamics, particularly useful in deep learning applications where exploding gradients can be a significant issue. The choice to use gradient clipping for the Fashion-MNIST dataset underscores the adaptability of BCGD to different data characteristics. By adjusting the algorithm's behavior based on the dataset, it is possible to achieve better optimization results and ensure that the model remains robust across various types of data.

Convergence Monitoring: At specified intervals (every 10 iterations), the algorithm evaluates the model's performance by calculating accuracy and loss both on the training set and the test set. This frequent monitoring allows for the tracking of progress and the determination of convergence.

Output: Upon completion, the algorithm returns the optimized parameters, along with metrics like accuracy and loss for both the training and test datasets, and the computational time for each recorded iteration.

5 Results and Comparisons

This section presents the results obtained from the application of Gradient Descent (GD) and Block Coordinate Gradient Descent (BCGD) across the three datasets: synthetic, Fashion-MNIST, and Iris. Detailed visualizations of the training and testing performance for each dataset are provided in the following subsections. These plots illustrate the effectiveness of the optimization algorithms in terms of accuracy and loss metrics throughout the training iterations, offering insights into their convergence behaviors and overall efficiency.

5.1 Synthetic Data

For the synthetic dataset, the optimization algorithms were applied with uniform settings to maintain consistency in the comparative analysis. The dimensions were set as m, d, k=1000,1000,50 with each algorithm initialized at the same starting point x derived from a standard normal distribution. A fixed learning rate of 0.001 and number of iterations 300 was utilized for both methods.

As depicted in Figure 1, Gradient Descent demonstrates robust performance with this learning rate, converging effectively towards a minimum. In contrast, the results for Block Coordinate Gradient Descent indicate that a higher learning rate might be necessary to achieve comparable outcomes. The slower convergence observed in BCGD underlines its sensitivity to the choice of learning rate in scenarios where the parameter space and dataset characteristics are complex. This suggests that while GD is stable across a range of learning rates, BCGD may require careful tuning of the learning rate to optimize performance effectively.

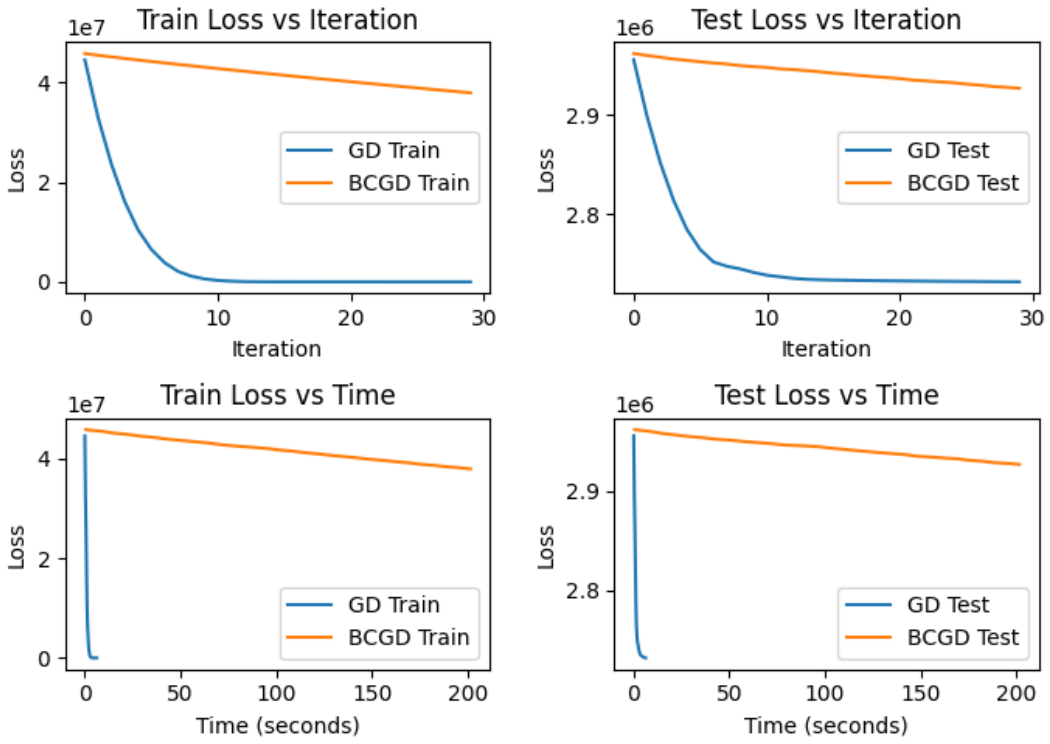

Figure 1. Results for loss with learning rate 0.001

In light of the initial findings, the learning rate for Block Coordinate Gradient Descent (BCGD) was adjusted to 0.1, while maintaining the learning rate for Gradient Descent (GD) at 0.001. This decision was based on observations that a higher learning rate for GD led to overfitting, as indicated by poor generalization on the test set.

Figures 2 and 3 illustrate the outcomes with these adjusted learning rates. The increase in the learning rate for BCGD resulted in significant improvements in convergence speed and overall performance. The plots clearly show that BCGD with a higher learning rate aligns more closely with the effectiveness of GD at its optimal learning rate of 0.001. This adjustment not only enhanced the convergence of BCGD but also balanced the comparative analysis between the two methods, highlighting the necessity of tuning the learning rates differently for each optimization technique to achieve optimal results in multi-class logistic regression tasks.

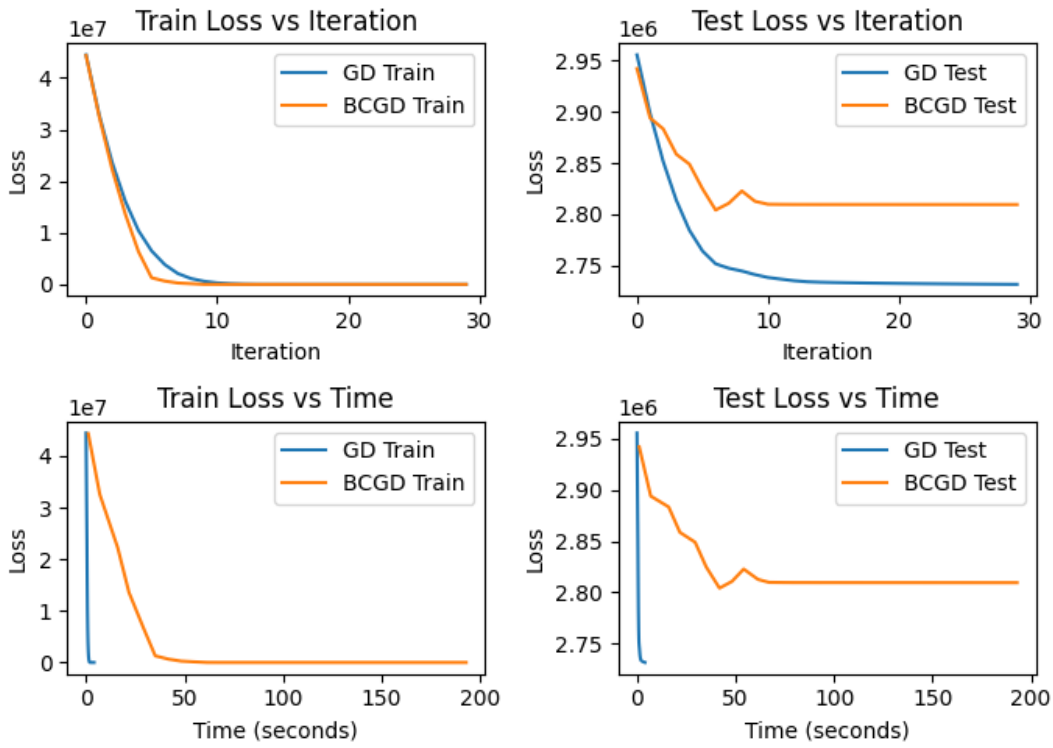

Figure 2. loss with learning rate 0.001 for GD and 0.1 for BCGD

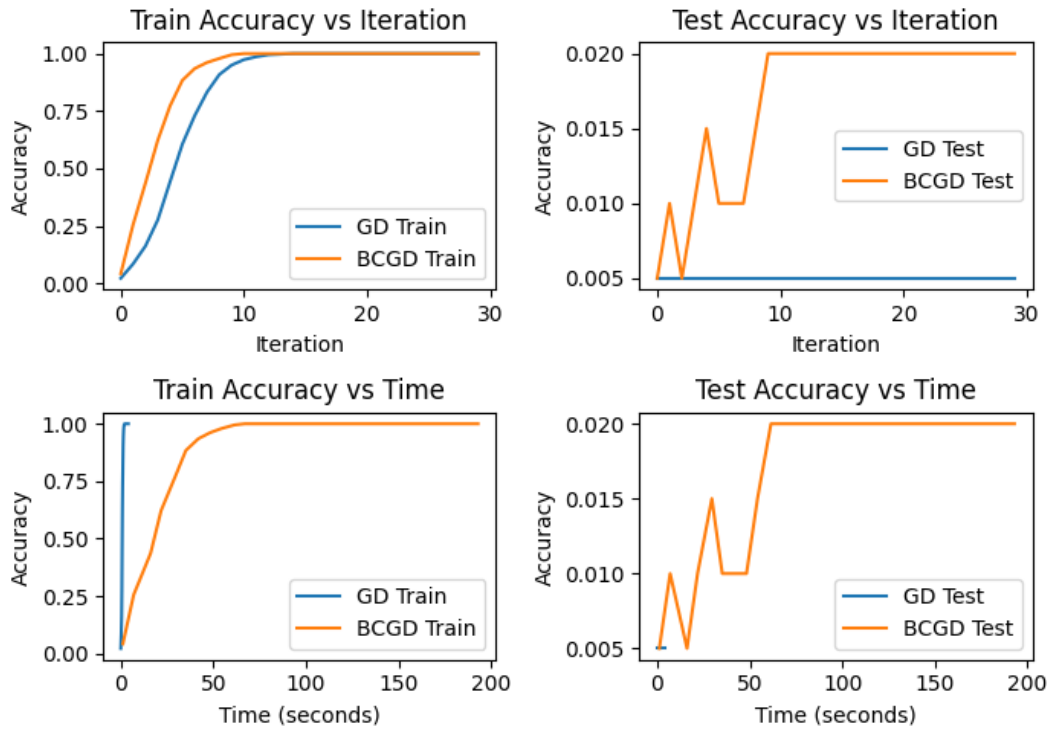

Figure 3. Accuracy with learning rate 0.001 for GD and 0.1 for BCGD

5.2 FMNIST data

In the FMNIST dataset analysis, the same experimental setup was applied as in the synthetic data analysis, with 300 iterations and standardized initial parameters k=10 for the number of classes and d=784 for the number of features, which correspond to the flattened image dimensions in the FMNIST dataset. The initial weights for the logistic regression model were drawn from a normal distribution, consistent across both Gradient Descent and Block Coordinate Gradient Descent.

Observations from the FMNIST data mirrored those from the synthetic dataset. Given that BCGD showed slower convergence compared to GD, a higher learning rate of 0.1 was utilized for BCGD to accelerate its optimization process and improve convergence speed. For GD, a learning rate of 0.001 was used to maintain steady progress without risking overfitting. These configurations led to improved model performance, as demonstrated in figures 4 and 5.

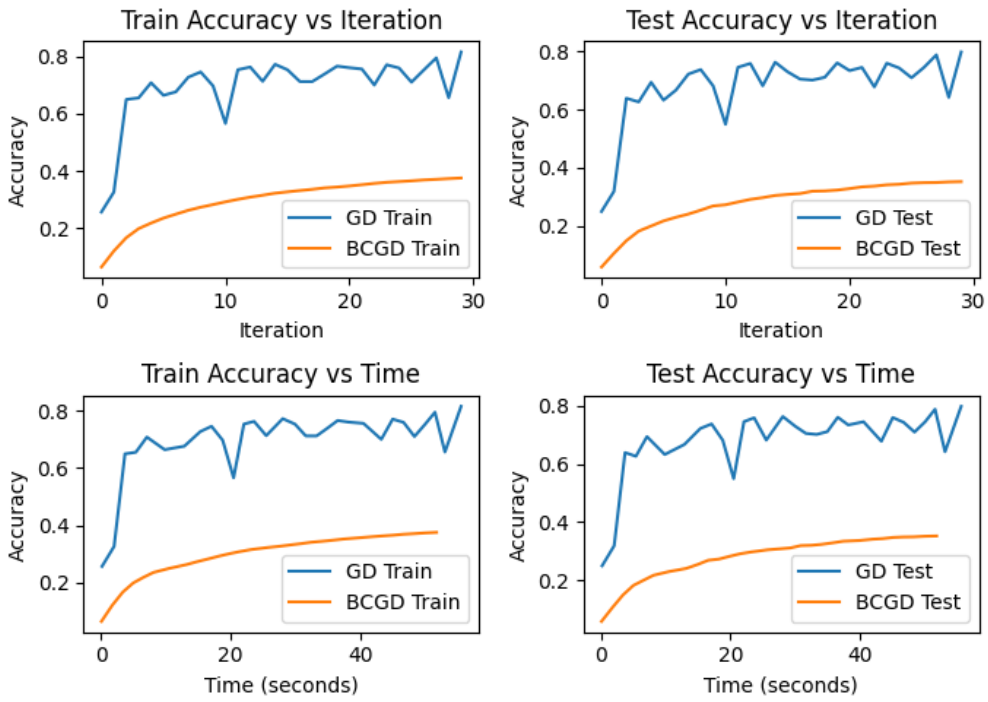

Figure 4. Accuracy with learning rate 0.001 for GD and 0.1 for BCGD

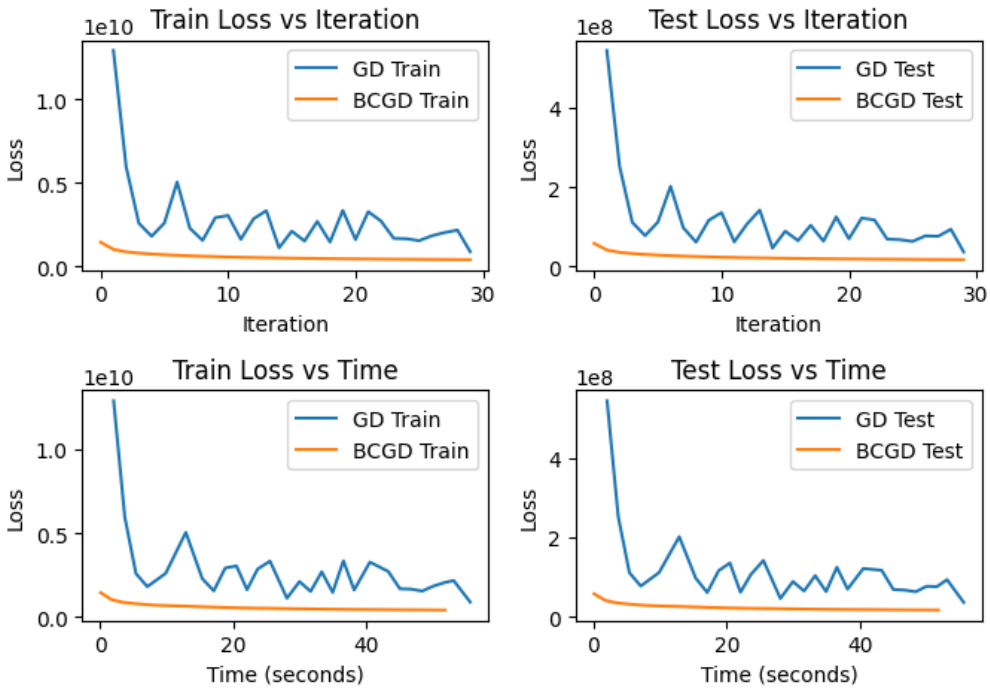

Figure 5. Loss with learning rate 0.001 for GD and 0.1 for BCGD

5.3 IRIS data

For the analysis of the Iris dataset, the setup was consistent with previous datasets to ensure comparability. The Iris dataset features three classes (k=3) and four features (d=4), and the logistic regression model was initialized with weights drawn from a normal distribution. This standardization was crucial for maintaining consistent conditions across all experiments.

Given the simpler nature of the Iris dataset, a learning rate of 0.001 was selected for both Gradient Descent and Block Coordinate Gradient Descent. This rate was determined to be optimal based on previous results which indicated good performance without overfitting on other datasets. This consistent learning rate across different algorithms and datasets facilitated a more straightforward comparison of their effectiveness and efficiency under similar conditions.

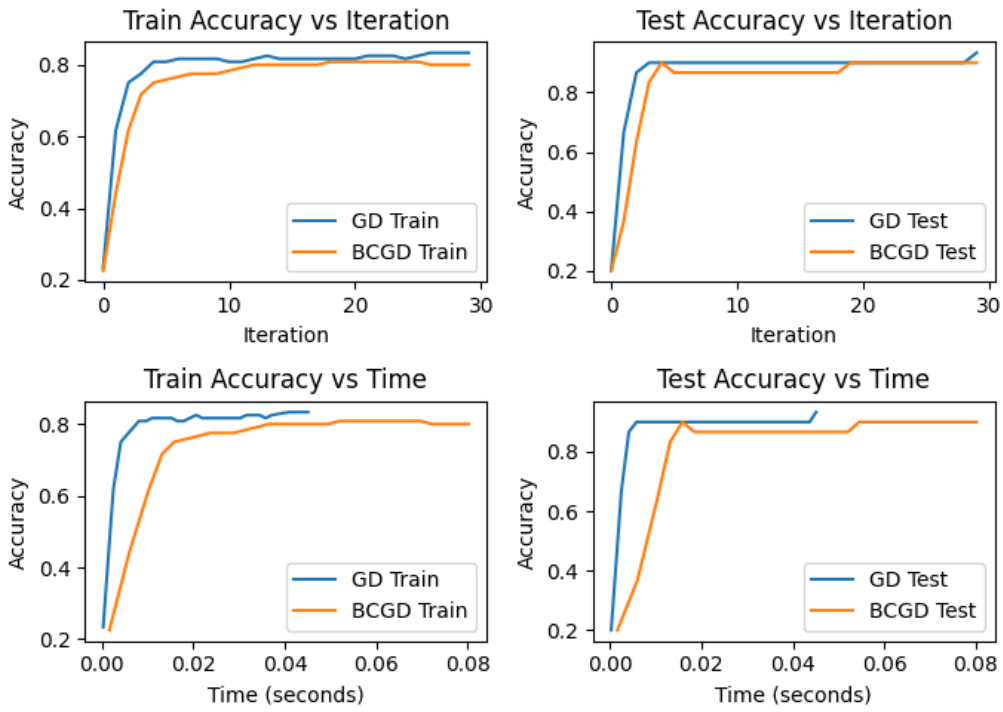

Figure 6. Accuracy with learning rate 0.001 for GD and BCGD

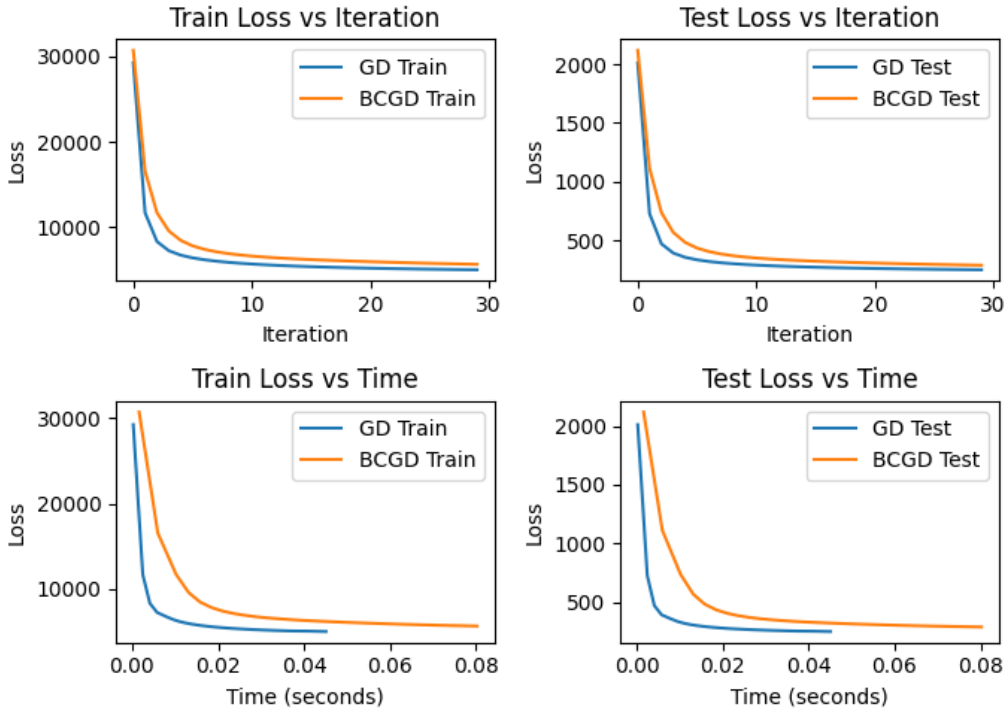

Figure 7. Loss with learning rate 0.001 for GD and BCGD

6 Conclusion

In this study, three distinct datasets of varying complexities (synthetic, Fashion-MNIST, and Iris) were utilized to conduct a comprehensive comparison of two optimization algorithms: Gradient Descent and Block Coordinate Gradient Descent with the Gauss-Southwell rule. This selection enabled the assessment of these algorithms across a spectrum of data characteristics, from highly synthetic and structured to real-world and noisy images, as well as simpler botanical measurements. The summary of results is shown in Table 1.

Data	Method	Learning Rate	Final Train Accuracy	Final Test Accuracy	Final Train Loss	Final Test Loss	Total CPU Time (Seconds)
Synthetic	GD	1e-3	1.00	0.005	2038.25	2731356.2	4.22
	BCGD	1e-3	0.0488	0.005	37867577.47	2926862.68	207.38
		1e-1	1.00	0.02	22.02	2809371.38	199.77
FMNIST	GD	1e-3	0.8154	0.798	890397580.47	36872536.87	57.89
	BCGD	1e-1	0.3735	0.3525	412981566.12	17346651.29	54.1
Iris	GD	1e-3	0.8333	0.9333	5044.44	247.67	0.05
	BCGD	1e-3	0.8	0.9	5683.11	286.46	0.08

Table 1. Algorithm Performance Summary

The synthetic dataset, with its high dimensionality and numerous classes, represented a challenging computational scenario, particularly for BCGD, which initially struggled with lower learning rates. When the learning rate for BCGD was increased to 0.1, both training efficiency and accuracy improved significantly, demonstrating the sensitivity of BCGD to appropriate parameter tuning in complex scenarios.

Fashion-MNIST, as a real-world dataset with substantial variability and noise, provided a robust platform for testing the generalizability of the algorithms. GD performed admirably with a high accuracy of 0.8154 on the training set and 0.798 on the test set, indicating strong fitting and generalization capabilities under a fixed learning rate of 0.001. In contrast, BCGD, even with a high learning rate of 0.1 and the implementation of gradient clipping for the Fashion-MNIST dataset, showed limited success. This highlights its potential limitations in handling complex patterns and noise effectively without additional optimizations.

The Iris dataset, being the simplest among the three, showcased high accuracies and low losses with both algorithms under the same learning rate conditions, affirming the effectiveness of both GD and BCGD in scenarios with fewer classes and lower feature dimensions. This dataset underscored the algorithms' capabilities in straightforward contexts where overfitting and computational complexity are less of concern.

This study reaffirms the necessity of careful algorithm selection and parameter tuning in machine learning. While the findings contribute to an academic understanding of how GD and BCGD perform under varied conditions, they also serve as a practical guide for similar educational

projects, emphasizing the practical aspects of learning algorithm behavior through empirical testing. This project thus provides a foundational step towards more nuanced applications and investigations in the field of machine learning optimization.