Optimization for Data Science Homework

Multi Class Logistic Regression Classification: Gradient Descent and Block Coordinate Gradient Descent (GS) methods analysis

Author: Shabnam Sattar

1 Introduction

Multi-class logistic regression is an extension of the binary logistic regression model used for predicting outcomes where the dependent variable can take on more than two classes. Unlike binary logistic regression, which estimates the probability of an instance belonging to one of two classes, multi-class logistic regression involves estimating probabilities for each class in a scenario where there are three or more outcomes. This is typically accomplished using a one-vs-rest (OvR) or a one-vs-one (OvO) approach, or by using the SoftMax function in a multinomial logistic regression framework. The SoftMax function converts the raw scores, or logits, for each class into probabilities that sum to one, providing a more comprehensive prediction model suitable for multiple categories. This method is particularly useful in various fields such as natural language processing, image classification, and medical diagnosis, where outcomes are inherently multi-class.

In practice, multi-class logistic regression works by generalizing the logistic function to handle multiple classes. The model parameters are estimated using maximum likelihood estimation, where the likelihood function is derived from the SoftMax function. The parameters include the weights for each feature and each class, and the training process involves optimizing these weights to minimize cross-entropy loss, a common objective function for classification tasks. While this model is powerful and widely used, it assumes linear separability in the feature space, which might not always hold true. Therefore, it is often combined with other techniques such as feature engineering, regularization, or kernel methods to improve performance and handle more complex datasets. Despite its assumptions and limitations, multi-class logistic regression remains a fundamental and highly interpretable method in the machine learning toolkit.

2 Problem Statement

2-1 Loss Function

The loss function in logistic regression, specifically the negative log-likelihood, quantifies the discrepancy between the predicted probabilities and the actual class labels. Lower values of this function indicate superior model performance. The loss function is mathematically expressed as follows:

$$\min \sum_{i=1}^{m} \left[-X_{b_i}^T a_i + \log\left(\sum_{c=1}^{k} \exp(X_c^T a_i) \right) \right]$$

where m represents the number of independent identically distributed training samples, k denotes the number of classes. X_c is column c of matrix parameter with dimensions $d \times k$, where d is the number of features, and label is $b_i \in \{1,2,\dots,k\}$

2-2 Generating Synthetic Data

In this section, we generate the matrix $A \in \mathbb{R}^{m \times d}$ with entries drawn from a standard normal distribution as an input matrix. To produce the labels, we construct two matrices, X and E from normal distribution, with dimensions $d \times k$ and $m \times k$, respectively. We then compute the expression $AX + E$. Finally, for each row in the resulting matrix, the index of the maximum value is assigned as the corresponding label. In our experiment, $m = d = 1000$ and $k = 50$.

2-3 Gradient

With respect to X_{Jc} the gradient of the function is:

$$\frac{\partial f(X)}{\partial X_{JC}} = -\sum_{i=1}^{m} a_{iJ}\left[I(b_i = c) - \frac{\exp(X_c^T a_i)}{\sum_{c'=1}^{k}\exp(X_{c'}^T a_i)}\right]$$

The identification variable *I* is defined such that it equals one if the label corresponding to input sample *i* is *c*, and zero otherwise.

To expedite the computation of the gradient in the code, we can fix certain constants that remain consistent throughout the entire computation:

$$AX = A \times X$$

$$exp.AX = e^{AX}$$

$$exp.sum = \sum_{c'=1}^{k}\exp(X_{c'}^T a_i) \quad \forall\, i$$

2-3 Step size

Throughout the analysis, we utilize a fixed step size. To ensure an appropriate step size, we initially calculate the Lipschitz constant, denoted as *L*. To compute the Lipschitz constant for the step size in our multi-class logistic regression problem, which remains consistent throughout the process, we need to derive it from the gradient of the loss function.

The Lipschitz constant *L* of the gradient $\nabla f(X)$ in logistic regression can be computed as follows:

1. **Hessian Matrix:** The Lipschitz constant is related to the Hessian matrix of the loss function. For logistic regression, the Hessian matrix is bounded by the maximum eigenvalue of $A^T A$
2. **Maximum Eigenvalue:** The Lipschitz constant *L* is the largest eigenvalue of $A^T A$
3. **Spectral Norm:** This is also known as the spectral norm (or the 2-norm) of the matrix *A.* It is the square of the largest singular value of A.

3 Definition of the Algorithms

The analysis incorporates two optimization algorithms: Gradient Descent and Block Coordinate Gradient Descent with Gauss-Southwell. For synthetic data, 100 iterations are executed for each algorithm, whereas for the real dataset, 500 iterations are performed.

The early stopping condition, applied before reaching the maximum number of iterations, is defined as $||\nabla f(X)|| < epsilon$ with $epsilon = 10^{-10}$

3-1 Gradient descent

The standard Gradient Descent algorithm computes the partial derivatives for all points simultaneously at each iteration and adjusts their labels by moving along the descent direction with an appropriate step size. This iterative process continues until a predefined stopping condition is met or the maximum number of iterations is reached.

$$X_K = X_{K-1} - \frac{1}{L}\nabla f(X_{K-1})$$

3-2 Block Coordinate Gradient Descent

Block Coordinate Gradient Descent (BCGD) is an optimization algorithm that extends traditional gradient descent by updating variables in blocks rather than individually. In each iteration, BCGD focuses on optimizing a subset of the variables while keeping the others fixed, which can simplify the optimization problem and improve computational efficiency, especially for high-dimensional data. This method divides the parameter space into blocks, and at each step, it performs a gradient descent update on one block while maintaining the

remaining parameters constant. The process iterates over all blocks cyclically or selects them according to a predefined scheme, ensuring convergence to a local minimum under certain conditions. BCGD is particularly useful in large-scale machine learning problems and scenarios where the objective function exhibits a natural block structure, as it can lead to faster convergence and more manageable computations compared to standard gradient descent.

Block Coordinate Gradient Descent encompasses various methods, primarily distinguished by the strategies they employ to select the index of the gradient element to update. Among these, the Gauss-Southwell BCGD method stands out by selecting the index corresponding to the element with the maximum norm of the gradient. This approach prioritizes updating the component of the gradient that exhibits the greatest magnitude, potentially leading to more efficient convergence by addressing the most significant gradient elements first.

$$i = Arg \max_{j\in\{1,..,b\}} \left|\left|\nabla_j f(X_k)\right|\right|$$

$$X_{k+1} = X_k - \frac{1}{L} U_{ik} \nabla_{ik} f(X_k)$$

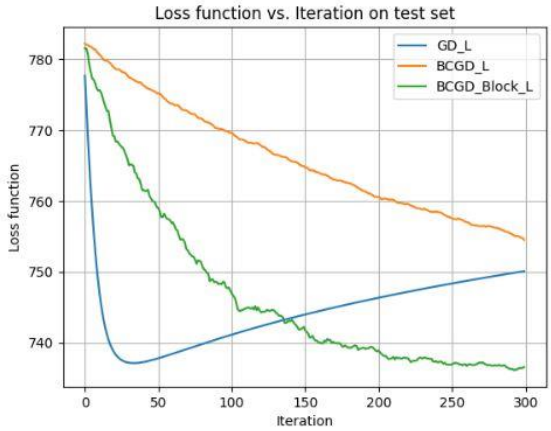
In our experiment, we designated each column of X as a single block, resulting in the number of blocks being equal to the number of labels in the dataset.

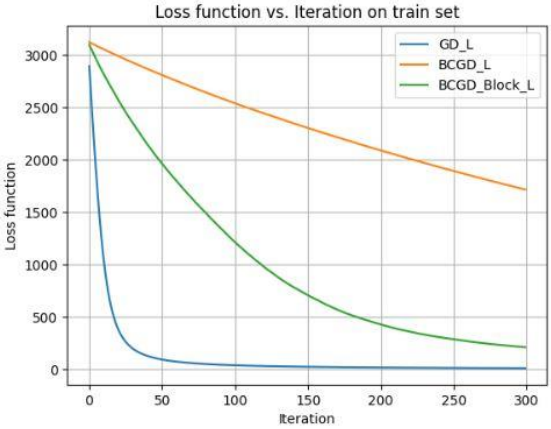
4 Results

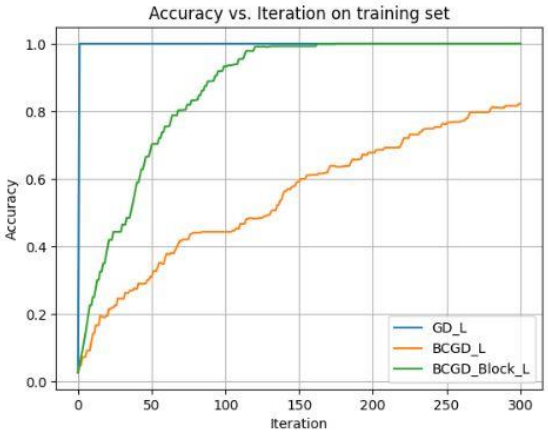
4-1 Synthetic Dataset

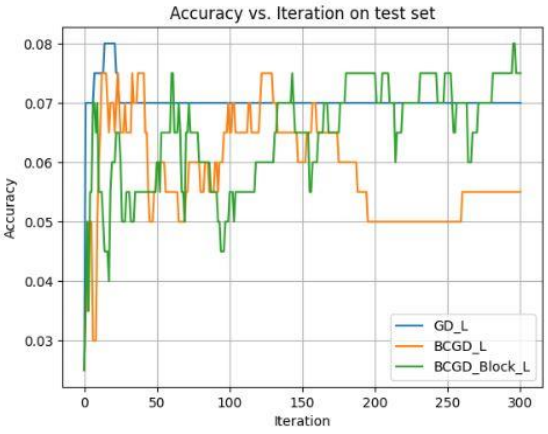
In this section, I implement Gradient Descent (GD) and Block Coordinate Gradient Descent (BCGD), and we analyze the results of the loss function and accuracy versus iterations and CPU time. For BCGD, I employ two methods for determining the step size. The first method uses a fixed L for each block, while the second method employs a block-wise Lipschitz step size, which assigns a different L to each block.

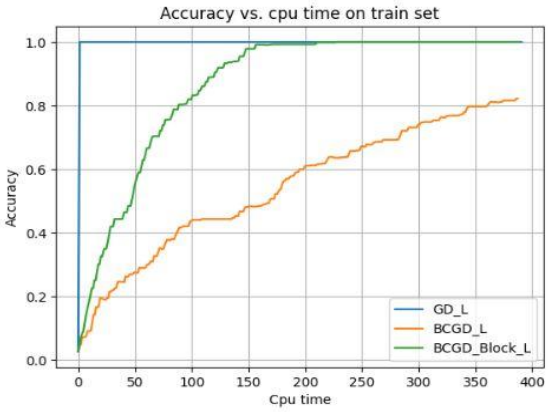
As we can observe in the plot related to the loss function and accuracy on the test set, the GD method starts to show signs of overfitting after iteration 25, as evidenced by the increasing loss and decreasing accuracy. Furthermore, the GD method performs better overall compared to the two BCGD methods. Additionally, among the BCGD methods, the one with varying L demonstrates better performance compared to the one with a fixed L.









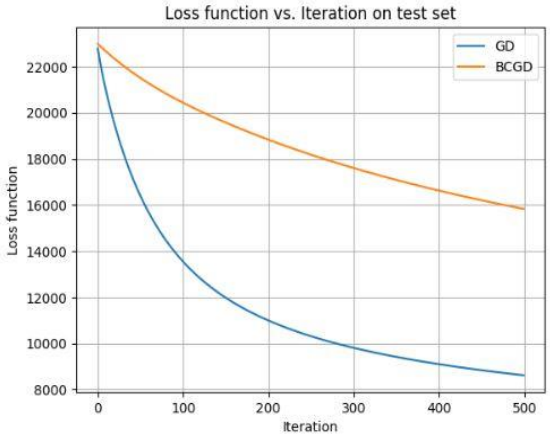


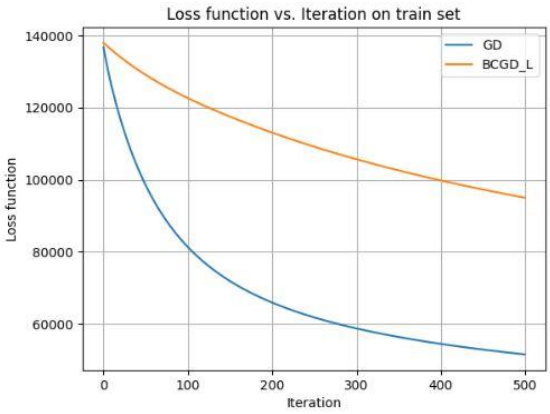
4-2 Real Dataset

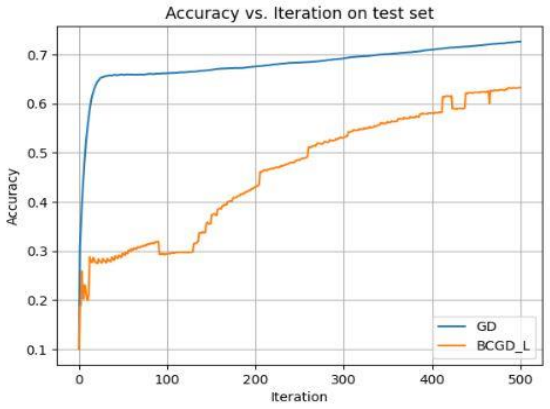
We also evaluate our model on a real dataset, specifically the Fashion MNIST dataset derived from Kaggle web site.

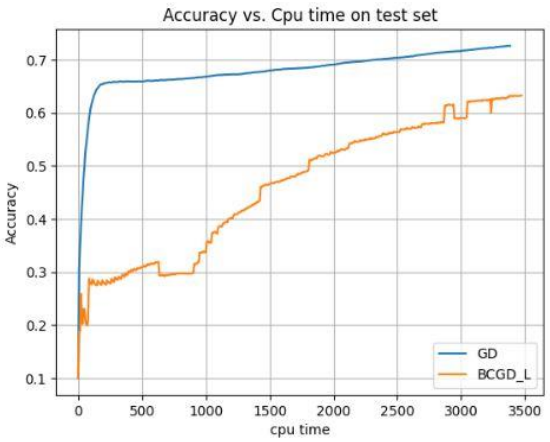
The Fashion MNIST dataset is a popular benchmark dataset in machine learning, consisting of 70,000 grayscale images of 10 different types of clothing items, such as T-shirts, trousers, and dresses. Each image is 28×28 pixels, providing 784 features per image. The dataset is split into 60,000 training images and 10,000 test images, and each image is associated with a label from one of the 10 classes.

Unlike our synthetic data, the implementation of the block Lipschitz step size on the real dataset was unsuccessful, yielding illogical results. Therefore, I decided to proceed with Gradient Descent (GD) and Block Coordinate Gradient Descent (BCGD) using a fixed Lipschitz constant step size. The results of these methods are presented below.









The Gradient Descent (GD) method consistently outperforms the Block Coordinate Gradient Descent (BCGD) method with a constant step size on the real dataset, highlighting the necessity of choosing an optimal optimization approach. While BCGD offers computational efficiency and scalability benefits, its performance may lag traditional gradient-based methods like GD. Further exploration of alternative optimization techniques is recommended to bridge this performance gap and improve the effectiveness of optimization algorithms in real-world scenarios.

5 Conclusion

This experiment's main objective was to explore optimization methods that outperform the Gradient Descent (GD) approach. While investigating the Block Coordinate Gradient Descent (BCGD) method with a fixed step size, it became evident that it did not yield superior performance. As a direction for future research, it is conjectured that methods employing varying step sizes or step sizes determined through line search techniques may offer improved results. Therefore, future investigations will focus on exploring these alternative optimization strategies to enhance convergence rates and achieve better overall performance in optimization tasks.