optimization for Data Science Homework

Sadaf Jamali,Bahador Mirzazadeh,
Mohammad Matin Parvanian, Seyedeh Moones Sheibani

May 2023

1 Introduction

In the field of optimization, one of the most important tasks is to improve the computational efficiency of algorithms while maintaining or improving their accuracy. Optimizing the update rule of gradient-based methods in specific Machine Learning problems is useful because it can significantly improve the computational efficiency of the ML algorithm. The update rule determines how the algorithm adjusts its parameters to minimize the cost function.
By optimizing this update rule, we can improve the algorithm's performance in terms of both speed and accuracy. In particular, optimizing the update rule with respect to the boundaries can lead to better convergence and more accurate results, which is especially important in cases where the boundaries are complex or poorly defined.
The primary objective of this assignment is to address a semi-supervised learning classification problem utilizing three unique methods: Gradient Descent, Randomized Block Coordinate Gradient Descent (RBCGD), and Block Coordinate Gradient Descent with Gaussian Southwell (BCGDGS). We aim to assess the effectiveness of each method by comparing their performances across a randomly chosen dataset. Following this, we will apply the methods on a specifically chosen dataset to determine which method works best for a given scenario.

2 Generating Random Data Samples

First of all, We generate 1000 data using make blobs from scikit-learn library. with 2 features and 2 centers.After this we split the generated dataset into two sets, a labeled set (X_labeled and y_labeled) and an unlabeled set (X_unlabeled and y_unlabeled). The labeled set is created by retaining only 50 instances and randomly assigning labels to this subset of the data. We considered the remaining 950 instances for Unlabled set.Then, the labels in y_labeled are adjusted by scaling and shifting them to be in the range of 1 and -1.

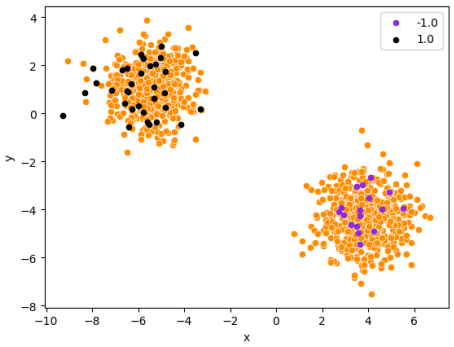

Figure 1: unlabeled data are shown with orange color while labeled data are shown in black and Violet colors

3 Model processing

In this section, we will define our problem and discuss different approaches in which we can address it.

3.1 Similarity Function

The similarity measure function is given below. For this task, we consider different similarity functions. Among all of them, we chose The Euclidean distance and The Manhattan distance. They are both good functions but there is a difference between them which made us choose The Euclidean distance. The Euclidean distance between two points is calculated as the straight-line distance between two sets of points. On the other hand, The Manhattan distance is measured by moving along the edges of a city block.

Euclidean Distance:

$$d_E(\vec{x}, \vec{y}) = e^{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}}$$

So the similarity matrix is defined as below:

$$\omega_{ij} = e^{\sqrt{(x_{1i} - x_{1j})^2 + (x_{2i} - x_{2j})^2}}$$

x_1 refer to the first dimension and x_2 is the second dimension in 2D.
i, j are two indexes for labeled and unlabeled data.

Therefore the matrix is as below:

$$\omega_{ij} = \begin{bmatrix} e^{\sqrt{(x_{11}-x_{11})^2+(x_{21}-x_{21})^2}} & e^{\sqrt{(x_{11}-x_{12})^2+(x_{21}-x_{22})^2}} & \dots \\ e^{\sqrt{(x_{12}-x_{11})^2+(x_{22}-x_{21})^2}} & \ddots & \dots \\ \vdots & \dots & \dots \end{bmatrix}$$

3.2 Terminology

Let's introduce our cost function first. As it is shown in the function below the cost function is consist of two terms. The first term is a weighted sum of the squared differences between y and a vector of means $\bar{y}^i$ for each group i. Besides, The second term is also a weighted sum of squared differences, but this time between each pair of y values.

Cost function:

$$\min_{y \in R^u} \sum_{i=1}^{l} \sum_{j=1}^{u} \omega_{ij}(y^j - \bar{y}^i)^2 + \sum_{i=1}^{u} \sum_{j=1}^{u} \bar{\omega}_{ij}(y^j - y^i)^2$$

Also, the gradient with respect to y^j is presented below. The first term is the weighted sum of the differences between y^j and a fixed value $\bar{y}^i$ for all i such that ω_{ij} is nonzero. Also, The second term is the weighted sum of the differences between y^j and y^i for all i such that $\bar{\omega}ij$ is nonzero. The weight of each difference is given by $\bar{\omega}ij$.

Gradient with respect to y^j:

$$\nabla_{y^j} f(y) = 2 \sum_{i=1}^{l} \omega_{ij}(y^j - \bar{y}^i) + 2 \sum_{i=1}^{u} \bar{\omega}_{ij}(y^j - y^i)$$

where

u: unlabeled data, l: labeled data, y: predicted label (parameters), $\bar{y}$: determined label, ω_{ij}: similarity measure matrix between labeled and unlabeled data, $\bar{\omega}_{ij}$: similarity measure matrix between unlabeled data.

4 Implementation

In this section, we will implement three different optimization algorithms on our dataset: Gradient Descent, Randomized Block Coordinate Gradient Descent (RBCGD), and Block Coordinate Gradient Descent with Gaussian-Southwell (BCGDGS). The goal is to determine which algorithm most suits our generated dataset and produces the best results. In order to perform these algorithms, we

defined the cost function first in Python. Next, We defined our cost function (dev_cost_function).
If the method is 'GD', the function performs gradient descent on the given parameters to calculate the development cost. If the method is 'RBCGD', the function performs Randomized Block Coordinate Gradient Descent (RBCGD) to calculate the development cost. If the method is 'BCGDGS', the function performs Block Coordinate Gradient Descent with Gaussian-Southwell (BCGDGS) to calculate the development cost. If the provided method is not one of the expected ones, the function raises a value error. The development cost is calculated based on the input parameters and the chosen method.

Besides, we defined the armijo_search function. This function takes in several parameters including the current weights, the labeled and unlabeled data, the derivative of the cost function with respect to the weights, an initial learning rate, a scalar c, and a maximum number of iterations for the search. The function calculates the cost function and its derivative for the current weights and then iteratively adjusts the learning rate until the new cost function is less than or equal to the old cost function minus a scaled multiple of the derivative of the cost function with respect to the learning rate. The function returns the updated learning rate and the new unlabeled data.

Finally, we have defined suitable functions for our optimization algorithms, which are Gradient Descent, Randomized Block Coordinate Gradient Descent, and Block Coordinate Gradient Descent with Gaussian Southwell. We have named these functions gd_method, rbcgd_method, and bcgdgs_method, respectively.

4.1 Gradient Descent

Letś use the Gradient Descent algorithm first. We consider 100 iterations with a learning rate of 0.0003. As you can see below the loss function is decreasing with each iteration and the accuracy function is increasing with each iteration so we can say that the Gradient Descent working properly for our generated dataset. we can improve these results by generating more data and applying more iterations, but we need to consider the matter of cost and time.

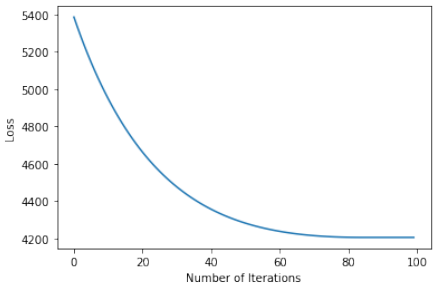

Figure 2: Loss vs Number of Iterations for GD

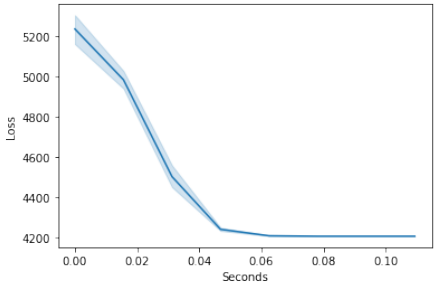

Figure 3: Loss vs Seconds for GD

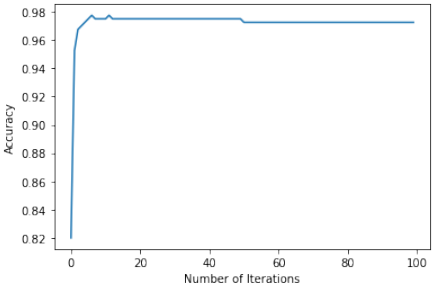

Figure 4: Accuracy vs Number of Iterations for GD

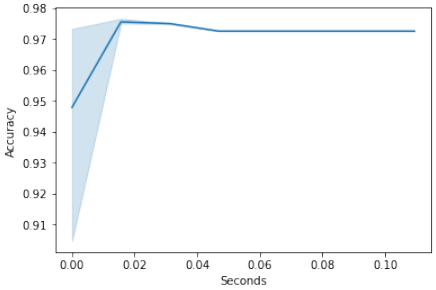

Figure 5: Accuracy vs Seconds for GD

.

4.2 Randomized Block Coordinate Gradient Decent

For this algorithm, we considered 100 iterations with a learning rate of 0.0001. As you can see the loss has decreased and the accuracy has increased. We could reach better accuracy by using more iterations but it took a lot of time. By the way, we could obtain some good results.

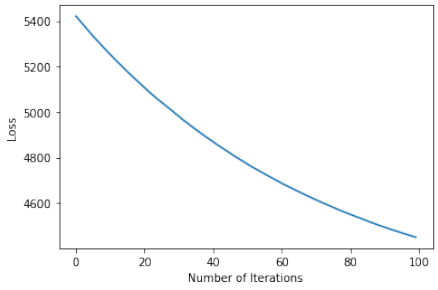

Figure 6: Loss vs Number of Iterations for RBCGD

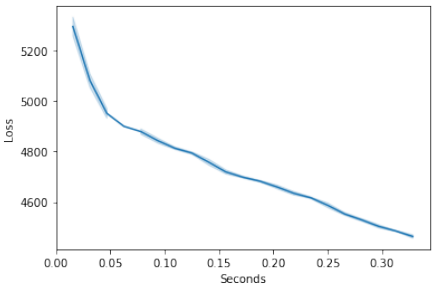

Figure 7: Loss vs Seconds for RBCGD

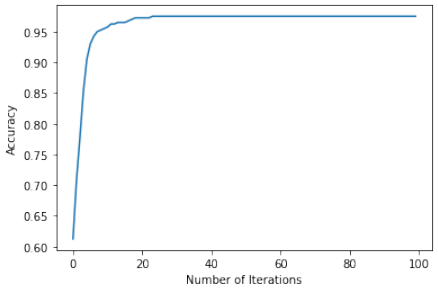

Figure 8: Accuracy vs Number of Iterations for RBCGD

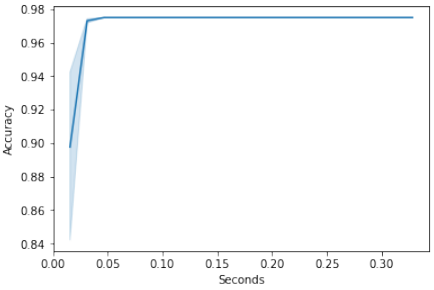

Figure 9: Accuracy vs Seconds for RBCGD

.

4.3 Block Coordinate Gradient Descent with Gaussian Southwell

For this algorithm, we considered 100 iterations with a learning rate of 0.0001. The loss function and accuracy function performance are shown below. Again we have to mention that the performance could be better with more iterations.

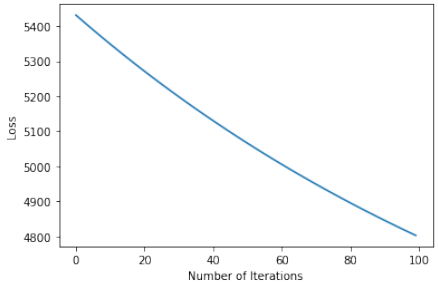

Figure 10: Loss vs Number of Iterations for BCGDGS

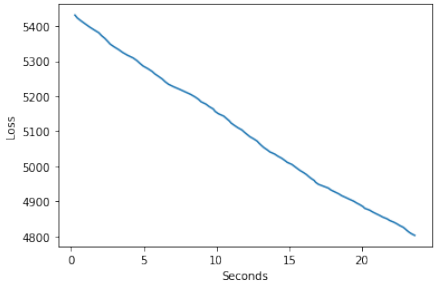

Figure 11: Loss vs Seconds for BCGDGS

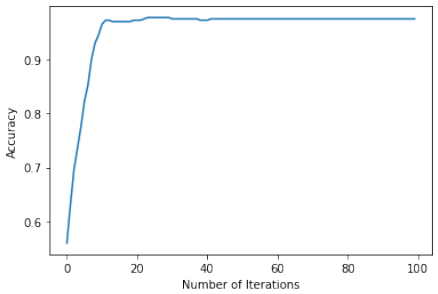

Figure 12: Accuracy vs Number of Iterations for BCGDGS

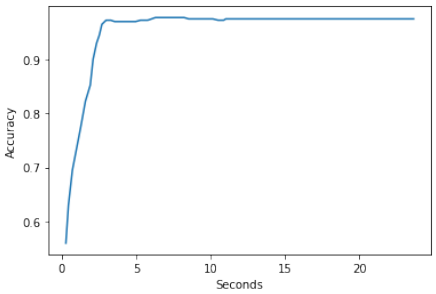

Figure 13: Accuracy vs Seconds for BCGDGS

.

4.4 Assessment of Model Performance

As you can see, The cost function performed better when we apply the Gradient descent method in comparison to the other two algorithms. Also, it is obvious that the Gradient descent method needs less time to achieve the highest possible accuracy in comparison to RBCGD and BCGDGS.

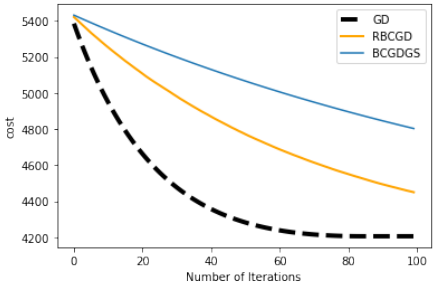

Figure 14: Loss vs Number of Iterations

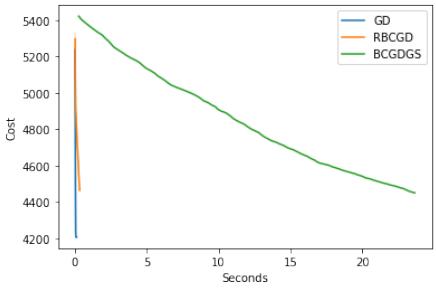

Figure 15: Loss vs Seconds

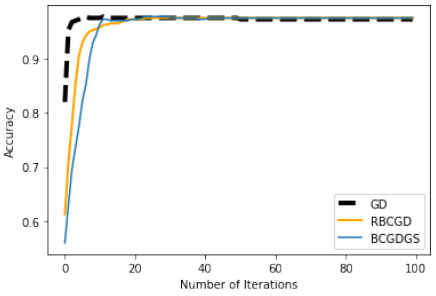

Figure 16: Accuracy vs Number of Iterations

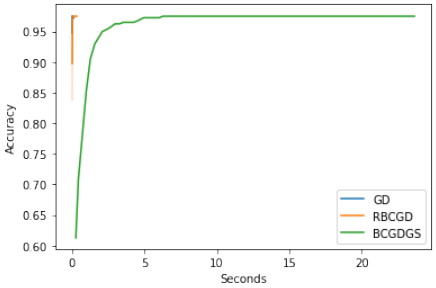

Figure 17: Accuracy vs Seconds

5 Spambase dataset

After testing out algorithms on our generated dataset, now it's time to check them for the outsourced dataset. In order to check this, we used the Spambase dataset from UCI Machine Learning Repository. The dataset includes instances of two class emails: spam and not spam. The number of features is 57 and the number of instances: is 4601 but for the matter of time, we considered 350 instances since after more than 5 hours the BCGDGS didn't finish 300 iterations. So we considered 2000 iterations for 350 instances.

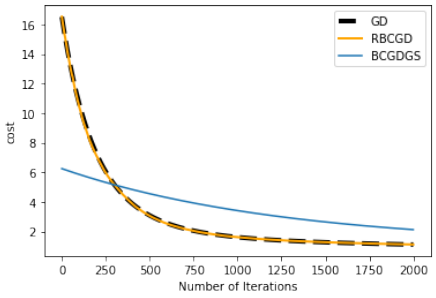

Figure 18: Loss vs Number of Iterations

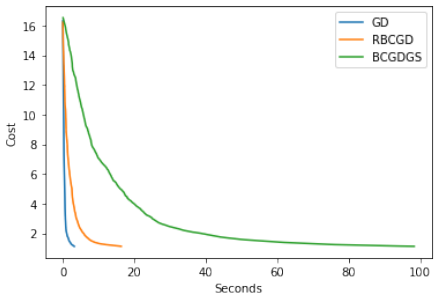

Figure 19: Loss vs Seconds

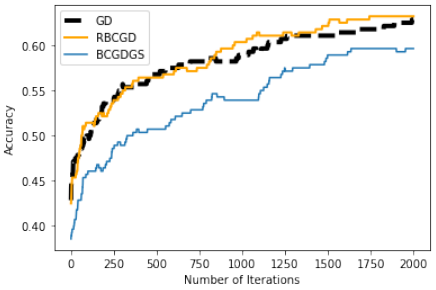

Figure 20: Accuracy vs Number of Iterations

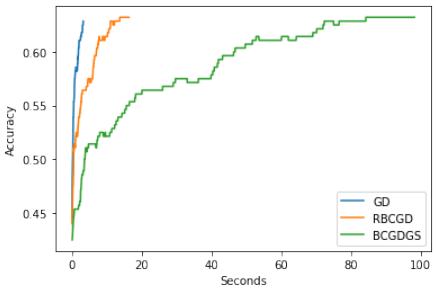

Figure 21: Accuracy vs Seconds

We can again see that the GD method has performed a better approach using less CPU time in comparison to the two other approaches. Besides, the GD method reached the highest possible accuracy in almost 4 seconds.