Exploring Gradient Descent and BCGD Models: A Comparative Analysis

Group:
Fairouz Baz Radwan, fairouz.bazradwan@studenti.unipd.it
ID: 2071913
Nour Al Housseini, nour.alhousseini@studenti.unipd.it
ID: 2081230
Lorenzo Ausilio, lorenzo.ausilio@studenti.unipd.it
ID: 2046831
Sofia Pope Trogu, sofiapopetrogu@studenti.unipd.it
ID: 2072117

Date: May 15, 2023

Table of Contents:

1. Introduction

Given the vast expanse of accessible data, we note that parameter tuning, feature selection, and classification impose a large challenge for modern day algorithms. In this report, we apply a sequence of unconstrained analytical algorithms to a classic semi-supervised binary classification problem. The paradigm includes a limited number of labeled data samples versus a large number of unlabeled data samples, in which the objective of the learning task is to identify the unlabeled data points. This approach involves employing a gradient descent method, the heavy ball method, as well as two different block coordinate gradient descent (BCGD) methods which all seek to optimize the minimization of the loss function.

Semi-supervised learning emerged as a revolutionary mechanism to transform the way machines and algorithms interpret learning problems. This learning method is a powerful approach that combines the strength of both supervised and unsupervised learning methods. In conventional supervised learning, models are trained using finite labeled input and output data, which impose certain constraints like time consumption, impracticality, and cost. On the other hand, unsupervised learning, such as clustering methods involve labeling training and test data based on various metrics, such as similarity measures, without ai priori knowledge of labels. Semi-supervised provides a bridge between these two learning methodologies. Not only has semi-supervised enhanced the field of artificial intelligence, but it has also addressed data scarcity concerns across many fields, like healthcare, finance, and census data collection. Semi-supervised learning methodologies have overall aided machines in creating an underlying framework for data distribution, hence enhancing machines' robustness and prediction abilities.

As we progressed in our analysis, we tested our methodologies on a real dataset in which the semi-supervised learning converged towards an unconstrained paradigm. This progressive, hybrid approach encouraged open-ended data exploration and novel analysis. However, we ensured that the flexibility of the problem was balanced with appropriate evaluation and analytical techniques.

2. Definitions & Notations

2.1. Loss Function

$$f(y) = \sum_{i=0}^{l}\sum_{j=0}^{u} w_{ij}\left(y_j - \overline{y}_i\right)^2 + \frac{1}{2}\sum_{i=0}^{u}\sum_{j=0}^{u} \overline{w}_{ij}\left(y_j - y_i\right)^2$$

In the loss function mentioned above, the following terms are:

- y_j: label assigned to the jth unlabeled element.
- $\overline{y}_j$: label assigned to the jth labeled element.
- w_{ij}: weight associated between the ith labeled element & jth unlabeled element
- $\overline{w}_{ij}$: weight associated between the ith unlabeled element & jth unlabeled element
- $l = |labeled\ data|$
- $u = |unlabeled\ data|$

The loss function expresses the main paradigm, in which "similar features = similar labels", and works on minimizing the overall disparity between our predicted and actual values.

2.2. Similarity Measure: Gaussian Weights

Among the many similarity measures available, we have chosen the Gaussian Kernel

$$K(x_1, x_2) = \exp\left(-\frac{\left||x_1 - x_2|\right|^2}{2\sigma^2}\right)$$

which is a non-linear function of the Euclidean distance between two vectors x_1 and x_2. The hyperparameter σ must be chosen wisely; if it is too large, then the Gaussian Kernel will approach 1 for all vectors, meaning that all vectors will be considered similar, and if it is very small, then the similarity will approach 0 for all vectors that are not "very close" to each other. We have tuned it for both of our datasets by trial and error.

2.3. Gradient

Given the loss function $f(y)$, we can compute the gradient vector with respect to each of the unlabeled output y_j , where $j \in \{1, \dots, |unlabeled\ data|\}$, in the following way:

$$\nabla_{y_j} f(y) = 2\sum_{i=0}^{l} w_{ij}(y_j - \overline{y}_i) + 2\sum_{i=0}^{u} \overline{w}_{ij}(y_j - y_i)$$

$$= 2\left[\left(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij}\right) y_j - \sum_{i=0}^{l} w_{ij}\ \overline{y}_i - \sum_{i=0}^{u} \overline{w}_{ij}\ y_i\right]$$

In the above equation, we notice that the first coefficient along with the 2nd term are independent of the unlabeled output and so we only need to compute them once as shown below. On the other hand, the third term $\sum_{i=0}^{u} \overline{w}_{ij}\ y_i$ depends on the entire unlabeled outcomes vector.

```
sumbycol_label_unlabel = np.sum(w_labeled_unlabeled, axis=0).reshape((-1,1))
sumbycol_unlabel_unlabel = np.sum(w_unlabeled_unlabeled, axis=0).reshape((-1,1))
grad_first_term = 2 * (sumbycol_label_unlabel + sumbycol_unlabel_unlabel)
grad_second_term = 2 * np.multiply(y_labeled.reshape((-1,1)), w_labeled_unlabeled).sum(0).reshape((-1,1))
```

2.4. Hessian & Lipschitz

To determine which step size to use in our algorithms, one common approach is to calculate the hessian matrix of our function and then compute the maximum eigenvalue which is an upper bound of the Lipschitz constant L. We then set our step size $\alpha = \frac{1}{L}$.

For the diagonal elements of the hessian matrix where k = j:

$$\nabla^2_{y_j y_j} f(y) \;=\; 2\left(\sum_{i=0}^{l} w_{ij} \;+\; \sum_{i=0}^{u} \overline{w}_{ij}\right) \;-\; 2\,\overline{w}_{jj}$$

For the non-diagonal elements of the Hessian matrix where $k \in \{1, \dots, u\} \setminus \{j\}$:

$$\nabla^2_{y_j y_k} f(y) \;=\; -\,2\,\overline{w}_{jk}$$

$$Hessian_{u\times u} \;=$$

$$\begin{bmatrix} \sum_{i=0}^{l} w_{i1} + \sum_{i=0}^{u} \overline{w}_{i1} - \overline{w}_{11} & -\overline{w}_{12} & \cdots & -\overline{w}_{1u} \\ -\overline{w}_{21} & \sum_{i=0}^{l} w_{i2} + \sum_{i=0}^{u} \overline{w}_{i2} - \overline{w}_{22} & \cdots & -\overline{w}_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ -\overline{w}_{u1} & -\overline{w}_{u2} & \cdots & \sum_{i=0}^{l} w_{iu} + \sum_{i=0}^{u} \overline{w}_{iu} - \overline{w}_{uu} \end{bmatrix}$$

2.5. Performance Metrics

We assess and compare the performance of the algorithms across the datasets by generating plots of their loss values, considering both the number of iterations and the computational time elapsed. While we provide accuracy plots to measure the performance of these algorithms, it's important to note that we haven't partitioned the dataset into typical train/validation/test splits, as is common in traditional machine learning scenarios. Our primary focus here is not to evaluate the algorithms' ability to generalize on unseen data, instead our main interest lies in comparing these algorithms in terms of their convergence behavior. We aim to analyze and compare how quickly and efficiently each algorithm converges to the solution, rather than their predictive performance on new data, and as such the overfitting scenario is not much of an issue in our case.

3. Datasets

3.1. Data Preprocessing

In real-world problems, data is abundant and unconstrained across many dimensions. However, visualizing data while attributing labels poses a serious challenge. This challenge is reflected in our report's mission to visualize and adapt unconstrained algorithms to constrained synthetic data. Raw data in its natural form can be messy, so some preprocessing techniques must be implemented. First, the data will be normalized using z-score standardization, which serves as a proxy for faster convergence during the initial learning process as well as reducing error prone measurement. This allows the algorithms'

assumptions to be authenticated, leading to better compatibility and improved performance. Formula for z-score standardization: $z = \frac{x - \mu}{\sigma}$.
After data standardization, we split the data into two main clusters, and randomly define the labeled and unlabeled data points following the classification {-1,1}.
The analytical algorithms will be conducted on the following datasets:

3.2. Synthetic Data

We generate a mock two-dimensional dataset algorithmically (two classes and clusters) given as two globule scatter plots. From a sample of 10,000 elements generated, only 10% are labeled. The remaining 90% will be initialized to 0, while saving the removed labels and indices for later use. This process signifies the essence of our semi-supervised algorithms. The saved labels will act as core reference points to compare our unlabeled data with. Depending on how well each algorithm converges towards our target labels, we will determine the most efficient solution for our optimization problem.

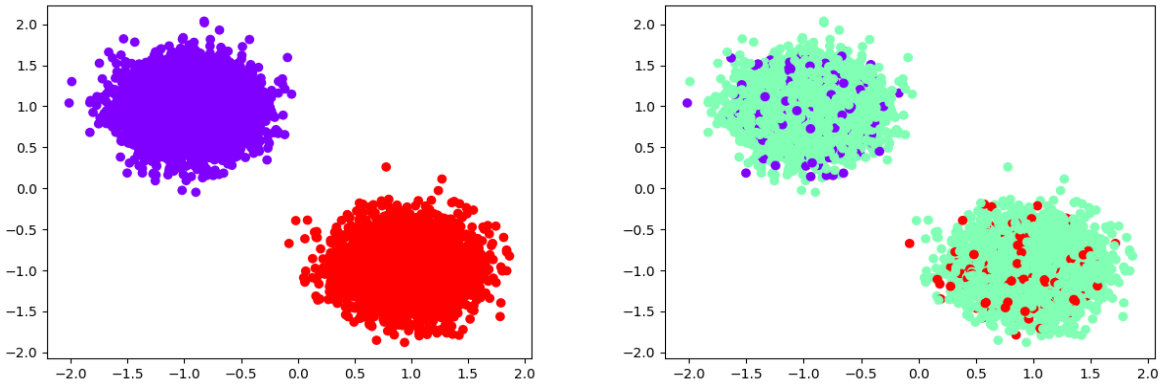

Figure 1. Synthetic data points before and after removing the labels

3.3. Real-World Data

Our choice of real-world data is a publicly available dataset titled the Skin Segmentation for Melanoma dataset. This dataset has a binary classification framework that focuses on detecting melanoma in skin segmentations through a positive and negative two-class mechanism. In brief, melanoma is a type of skin cancer that originates in melanocytes that have potential to spread to other parts of the body. The set contains a total of 245,057 occurrences, with approximately 6,000 instances labeled as positive (as in melanoma positive) and the remaining instances labeled as negative (non-melanoma). However, we subset the dataset to 10,000 balanced data points for our experiments. Each occurrence in the dataset consists of three continuous-valued features, which are derived from the RGB color space of the skin segment. Since the original dataset was 3-dimensional we used PCA to reduce the dimensionality to 2. The dataset was created by the Laboratory of Artificial and Natural Intelligence at the University of Porto and is available as part of the UCI Machine Learning Repository. It was originally used in their paper "Human skin color clustering for face detection",'https://archive.ics.uci.edu/ml/machine-learning-databases/00229/Skin_NonSkin.txt'.

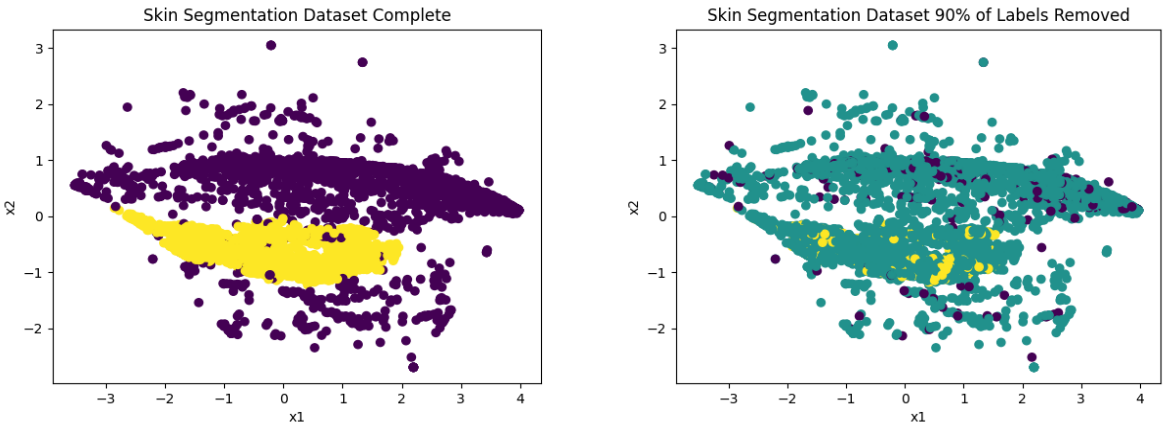

Figure 2. Skin-Segmentation data points before and after removing labels

4. Algorithms

4.1. Gradient descent

This algorithm's importance lies in its ability to identify discrepancies between our predicted output compared to our original output. The main benchmark of its role is to minimize the loss function to find the optimal values (refer to section 2.1). We start by initializing the method's parameters and hyperparameters, which we will discuss further in detail in the sections below. The gradient descent methodology updates the parameters as it calculates the search direction at each iteration. The search direction is affected by other hyperparameters we quantify, such as gradient step size. This process is repeated iteratively within a maximum number of iterations until convergence is met or the loss function is minimized.

The general formula for gradient descent is as follows: $x^{k+1} = x^k - \alpha_k \nabla f(x^k)$

4.1.1. Improved Gradient Descent

This is considered an important step for modifications to obtain a faster convergence. Several strategies exist to improve the gradient descent method, including a fitted step size of $\alpha = \frac{1}{L}$ (refer to section 2.4).

4.1.2. Momentum-based Gradient Descent: Heavy Ball Method

The Heavy Ball method is an updated gradient descent

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k) + \beta^k (x^k - x^{k-1})$$

which adds a momentum term $x^k - x^{k-1}$ responsible for nudging the current update in the direction of the previous update. It was introduced by Polyak and, intuitively, a heavier ball will bounce less and move faster through regions of low curvature than a lighter ball due to momentum. The hyperparameter β^k thus controls the amount of momentum from the previous step that is applied to the current step, with a higher value leading to a larger influence from the previous step.
For our analysis instead of keeping β^k fixed, we have chosen it as

$$\beta^k = \frac{\lambda^{k-1} - 1}{\lambda^k}$$

where λ^k is defined as follows $\lambda^k = \frac{1+\sqrt{1+4(\lambda^k)^2-1^2}}{2}$ and λ^0 is initialized at 0.

4.2. Block Coordinate Gradient Descent

Block coordinate gradient descent (BCGD) is an optimization algorithm that aims to minimize a function by updating subsets of variables (blocks) in each iteration.

4.2.1. BCGD randomized

Randomized BCGD is a variant of BCGD where the order of updating the blocks is random. In our study, we consider a gaussian distribution to randomly generate our blocks taking the improved probability measure $p[i_k = i] = \frac{L_i}{\sum_{i=0}^{u} L_i}$ instead of using a uniform distribution with a fixed probability $\frac{1}{u}$.

The update is then applied on 1 single index in each iteration as follows: $y_j^{k+1} = y_j^k - \frac{1}{L} \nabla_{y_j^k} f(y^k)$

4.2.2. BCGM Gauss-Southwell

Unlike the previous algorithm, which randomly selects the blocks at each iteration, the BCGD Gauss-Southwell algorithm chooses the indices based on a deterministic strategy known as the Gauss-Southwell rule. In our code, we rely on choosing the maximum absolute value of the gradient array as our index in each iteration. And so, it is important to us to compute the updated full gradient array at each of the iterations. To do that, we use our loss function and the gradient derived with respect to each unlabeled point to compute the update of the gradient descent.

$$\min_{y \in \mathbb{R}^u} \left(\sum_{i}^{l} \sum_{j}^{u} w_{ij} (y_j - \overline{y}_i)^2 + \frac{1}{2} \sum_{i}^{u} \sum_{j}^{u} \overline{w}_{ij} (y_j - y_i)^2 \right)$$

- First, we suppose that we are deriving with respect to the unlabeled point y_j:

$\nabla_{y_j} f(y) = 2\sum_{i=0}^{l} w_{ij}(y_j - \overline{y}_i) + 2\sum_{i=0}^{u} \overline{w}_{ij}(y_j - y_i)$, where j= 1,...,u

Which we can rewrite as:

$$\nabla_{y_j} f(y) = 2\left[\left(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij}\right) y_j - \sum_{i=0}^{l} w_{ij}\ \overline{y}_i - \sum_{i=0}^{u} \overline{w}_{ij}\ y_i\right]$$

- Assuming we're updating the j[th] unlabeled point at iteration k+1 in our algorithm, we'll have:

$$\nabla_{y_j^{k+1}} f(y^{k+1}) = 2\left[\left(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij}\right) y_j^{k+1} - \sum_{i=0}^{l} w_{ij}\ \overline{y}_i - \sum_{i=0}^{u} \overline{w}_{ij}\ y_i^{k+1}\right]$$

For simplicity denote $A_j = \sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij}$ and $b_j = \sum_{i=0}^{l} w_{ij}\ \overline{y}_i$

Substituting the above terms and expanding the last sum, we obtain:

$$\nabla_{y_j^{k+1}} f(y^{k+1}) = 2\left[A_j\, y_j^{k+1} - b_j - \left(\overline{w}_{0j}\, y_0^{k+1} + \ldots + \overline{w}_{jj}\, y_j^{k+1} + \cdots + \overline{w}_{uj}\, y_u^{k+1}\right)\right]$$

- Now we note that since we're only updating the j[th] unlabeled point at the iteration k+1, we know then that all the other unlabeled outputs are unchanged.

In other words: $y_i^{k+1} = y_i^k \ \forall\, i \in \{1, \ldots, u\} \setminus \{j\}$ and $y_j^{k+1} = y_j^k - \frac{1}{L}\nabla_{y_j} f(y^k)$

And so, substituting in our equation:

$$\nabla_{y_j^{k+1}} f(y^{k+1}) = 2\left[A_j\left(y_j^k - \frac{1}{L}\nabla_{y_j^k} f(y^k)\right) - b_j - \left(\overline{w}_{0j}\, y_0^k + \ldots + \overline{w}_{jj}\left(y_j^k - \frac{1}{L}\nabla_{y_j^k} f(y^k)\right) + \cdots + \overline{w}_{uj}\, y_u^k\right)\right]$$

Taking the notation $step = \frac{1}{L}\nabla_{y_j^k} f(y^k)$ and rearranging our equation, we get:

$$\begin{aligned}\nabla_{y_j^{k+1}} f(y^{k+1}) &= 2\left(A_j\, y_j^k - b_j - \sum_{i=0}^{u} \overline{w}_{ij}\ y_i^k\right) + 2(A_j - \overline{w}_{jj})(-step) \\ &= \nabla_{y_j^k} f(y^k) + 2(A_j - \overline{w}_{jj})(-step)\end{aligned}$$

- Now, computing the gradient with respect to the other unlabeled outputs $y_z\ where\ z \in \{1, \ldots, u\} \setminus \{j\}$, we have:

$$\begin{aligned}\nabla_{y_z^{k+1}} f(y^{k+1}) &= 2\left[\left(\sum_{i=0}^{l} w_{iz} + \sum_{i=0}^{u} \overline{w}_{iz}\right) y_z^{k+1} - \sum_{i=0}^{l} w_{iz}\ \overline{y}_i - \sum_{i=0}^{u} \overline{w}_{iz}\ y_i^{k+1}\right] \\ &= 2\left[A_z\, y_z^{k+1} - b_z - \left(\overline{w}_{0z}\, y_0^{k+1} + \ldots + \overline{w}_{jz}\, y_j^{k+1} + \cdots + \overline{w}_{uz}\, y_u^{k+1}\right)\right]\end{aligned}$$

Similarly, to the first case, since we're only updating the j[th] unlabeled point at the iteration k+1, we have:

$$y_z^{k+1} = y_z^k \;\; \forall\, z \in \{1, \dots, u\} \setminus \{j\} \quad \text{and} \quad y_j^{k+1} = y_j^k - \frac{1}{L}\nabla_{y_j} f\left(y^k\right)$$

In this case we have only 1 term that depends on y_j^{k+1} and so our equation becomes:

$$\begin{aligned}\nabla_{y_z^{k+1}} f\left(y^{k+1}\right) &= 2\left[A_j\, y_z^k - b_j - \left(\overline{w}_{0z}\, y_0^k + \dots + \overline{w}_{jz}\left(y_j^k - \frac{1}{L}\nabla_{y_j^k} f\left(y^k\right)\right) + \cdots + \overline{w}_{uz}\, y_u^k\right)\right] \\ &= 2\left(A_j\, y_z^k - b_j - \sum_{i=0}^{u} \overline{w}_{iz}\; y_i^k\right) + 2\,\overline{w}_{jz}\,(step) \\ &= \nabla_{y_z^k} f\left(y^k\right) + 2\;\overline{w}_{jz}\,(step)\end{aligned}$$

- And so, we update our gradient array at the iteration k+1:

$$\nabla_{y_j^{k+1}} f\left(y^{k+1}\right) = \nabla_{y_j^k} f\left(y^k\right) + 2\left(\overline{w}_{jj} - A_j\right)\left(\frac{1}{L}\,\nabla_{y_j^k} f\left(y^k\right)\right)$$

$$\nabla_{y_z^{k+1}} f\left(y^{k+1}\right) = \nabla_{y_z^k} f\left(y^k\right) + 2\,\overline{w}_{jz}\left(\frac{1}{L}\nabla_{y_z^k} f\left(y^k\right)\right)\;,\; \forall\, z \in \{1, \dots, u\} \setminus \{j\}$$

We notice that we're adding the term $2\,\overline{w}_{ji}\,\frac{1}{L}\nabla_{y_i^k} f\left(y^k\right)$ to each component of the gradient array, and we implement that in our code by adding the j[th] row of the w_unlabeled_unlabeled multiplied by 2*step.

Then we only update the j[th] component of the gradient by subtracting the term: $2\left(A_j\right)\left(\frac{1}{L}\,\nabla_{y_j^k} f\left(y^k\right)\right)$

```
def bcgd_gs_gradient(previous_grad, index, Li):
    update_step = 1/Li[index] * previous_grad[index]
    bcgd_gs_grad = np.copy(previous_grad)
    bcgd_gs_grad += 2 * w_unlabeled_unlabeled[index] * update_step
    bcgd_gs_grad[index] -= grad_first_term[index] * update_step
return update_step, bcgd_gs_grad
```

Finally, at each iteration we select a singleton block using the $j = Arg \max_{i \in \{1,..,u\}} \left|\left|\nabla_{y_i} f(y)\right|\right|$ and then update the k[th] unlabeled output as follows: $y_j^{k+1} = y_j^k - step$

5. Results

5.1. GD Comparisons: Improved GD vs Heavy Ball

5.1.1. Synthetic Dataset

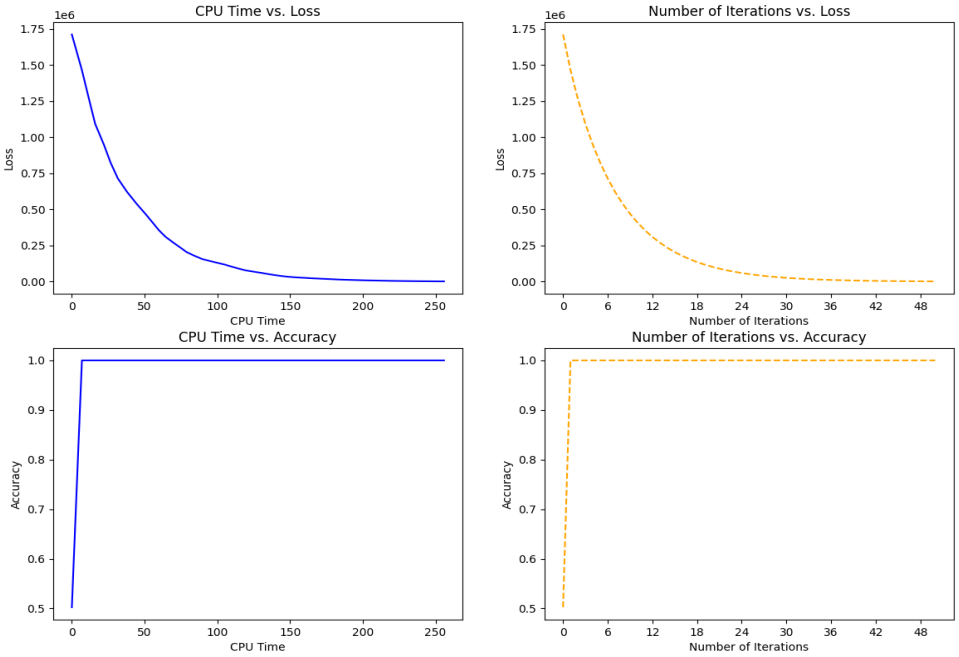

Figure 3: Synthetic dataset: Loss and Accuracy metrics of Improved Gradient Descent

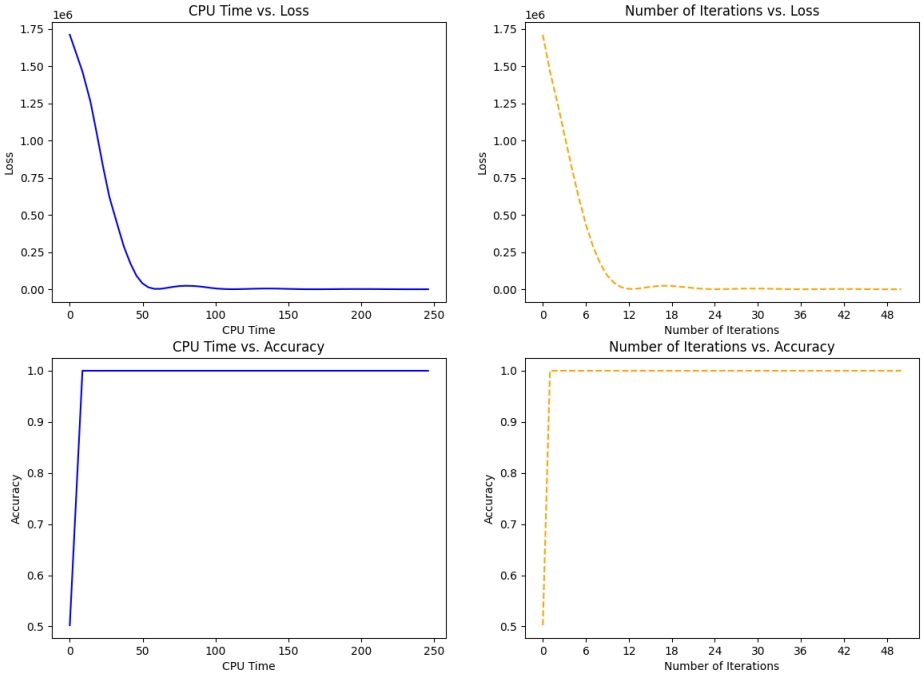

Figure 4: Synthetic dataset: Loss and Accuracy metrics of Heavy Ball Gradient Descent

5.1.2. Skin Cancer Dataset

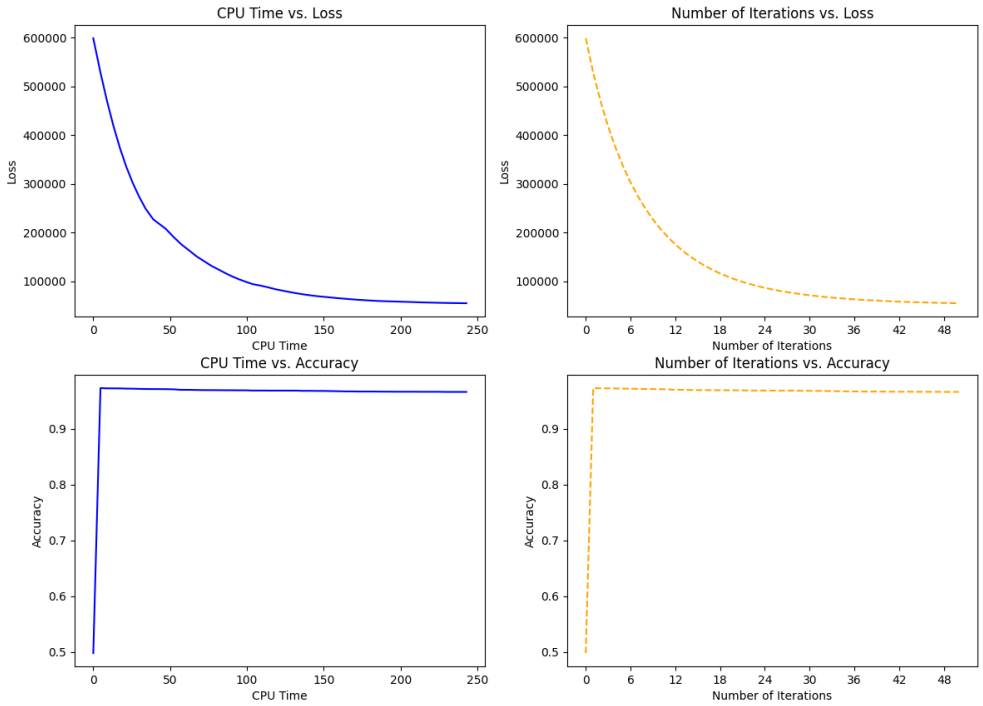

Figure 5: Skin-Cancer dataset: Loss and Accuracy metrics of Improved Gradient Descent

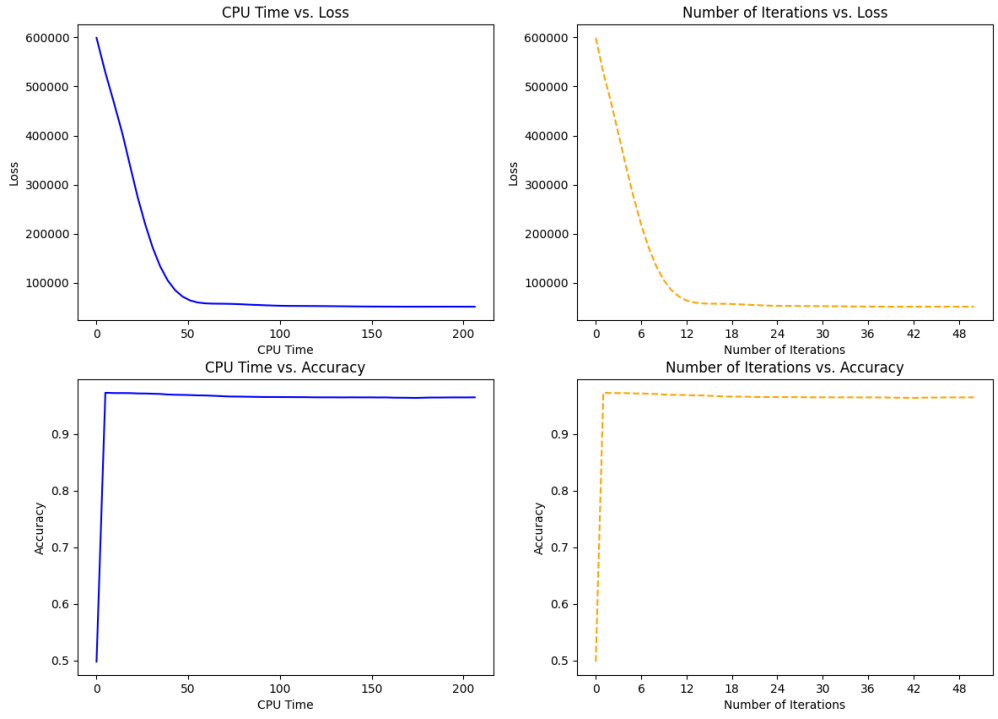

Figure 6: Skin-Cancer dataset: Loss and Accuracy metrics of Heavy Ball Gradient Descent

5.2. BCGD Comparisons: Randomized vs Gauss-Southwell

5.2.1. Synthetic Dataset

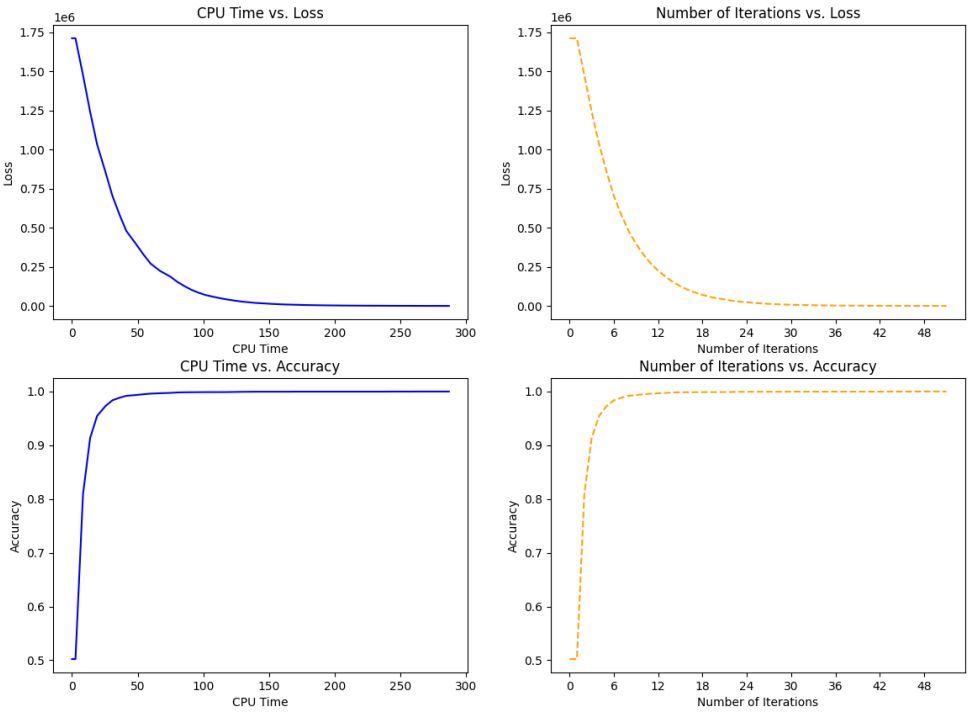

Figure 7. Synthetic Dataset: Loss and Accuracy metrics of BCGD: Randomized method

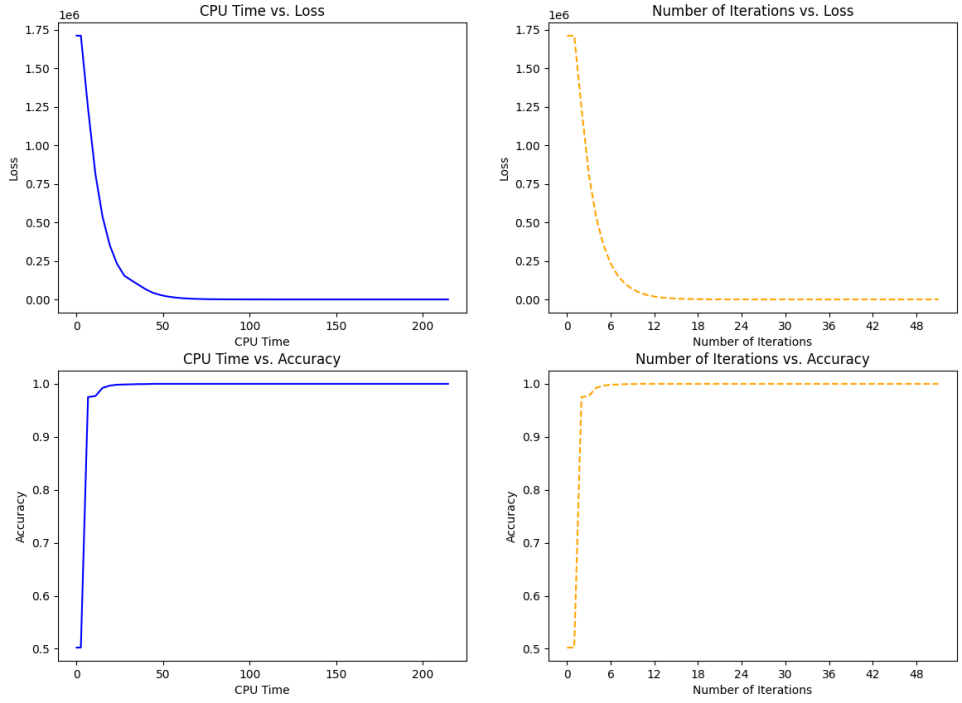

Figure 8. Synthetic Dataset: Loss and Accuracy metrics of BCGD: Gauss Southwell method

5.2.2. Skin Cancer Dataset

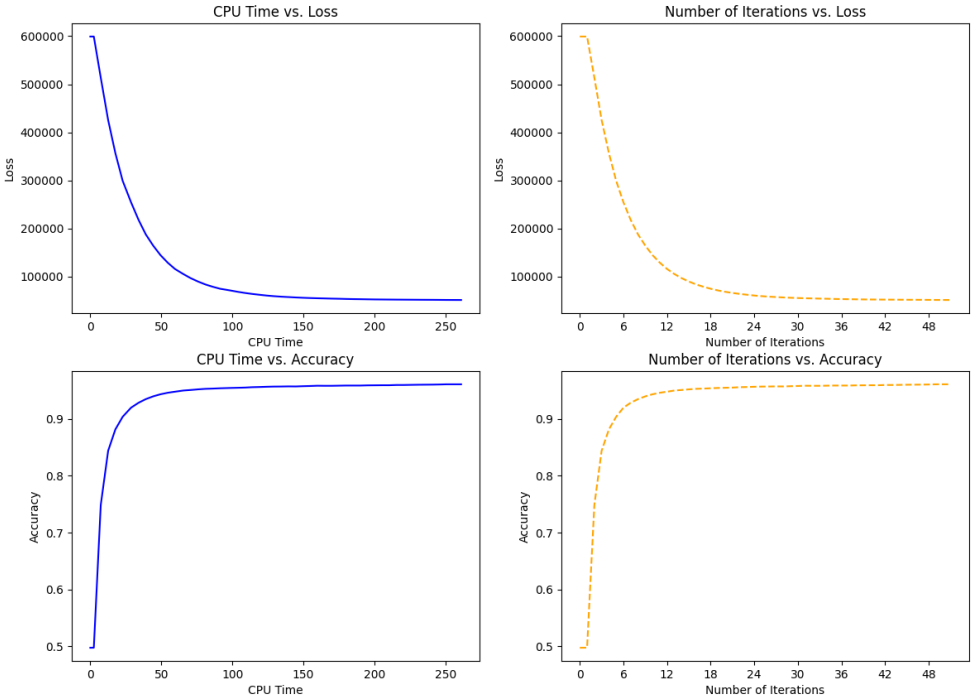

Figure 9. Skin-Cancer Dataset: Loss and Accuracy metrics of BCGD: Randomized method

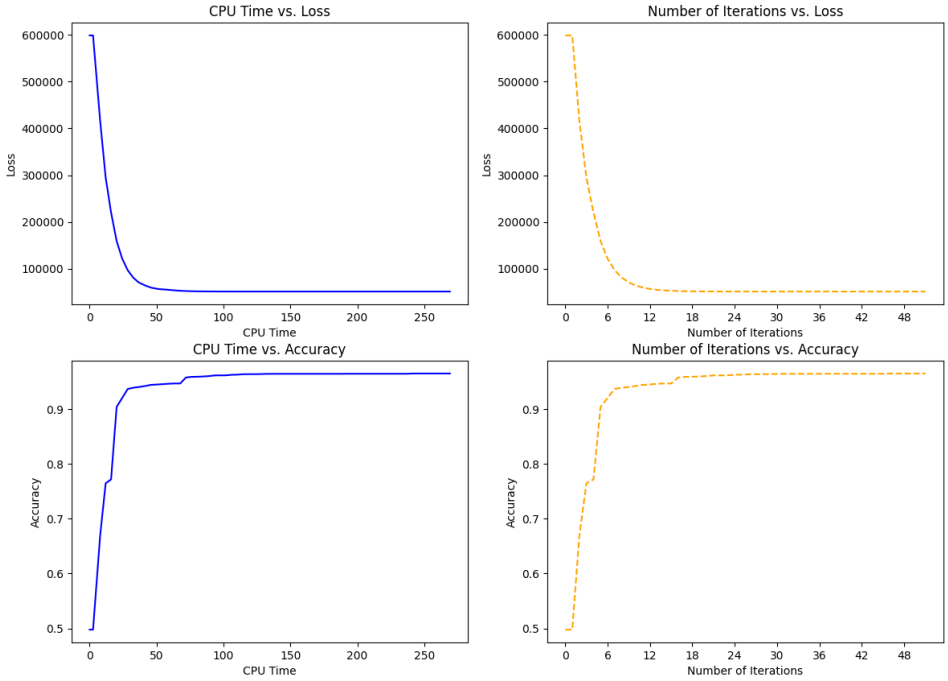

Figure 10: Skin-Cancer Dataset: Loss and Accuracy metrics of BCGD: Gauss Southwell method

5.3. Overall Comparisons

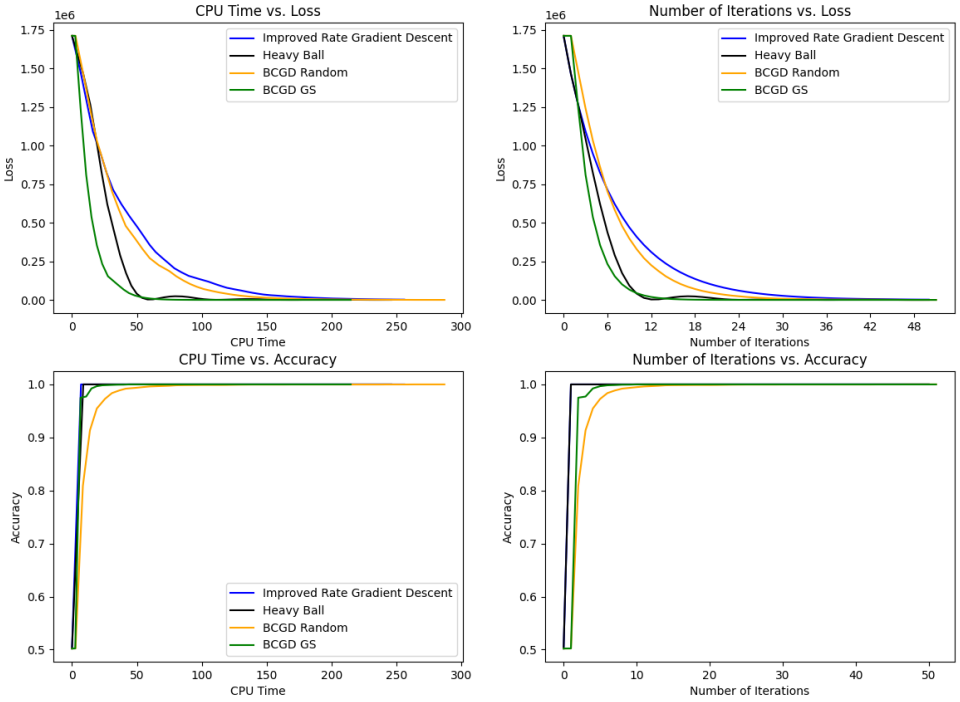

Figure 11: Synthetic Dataset: Comparison of all optimization models

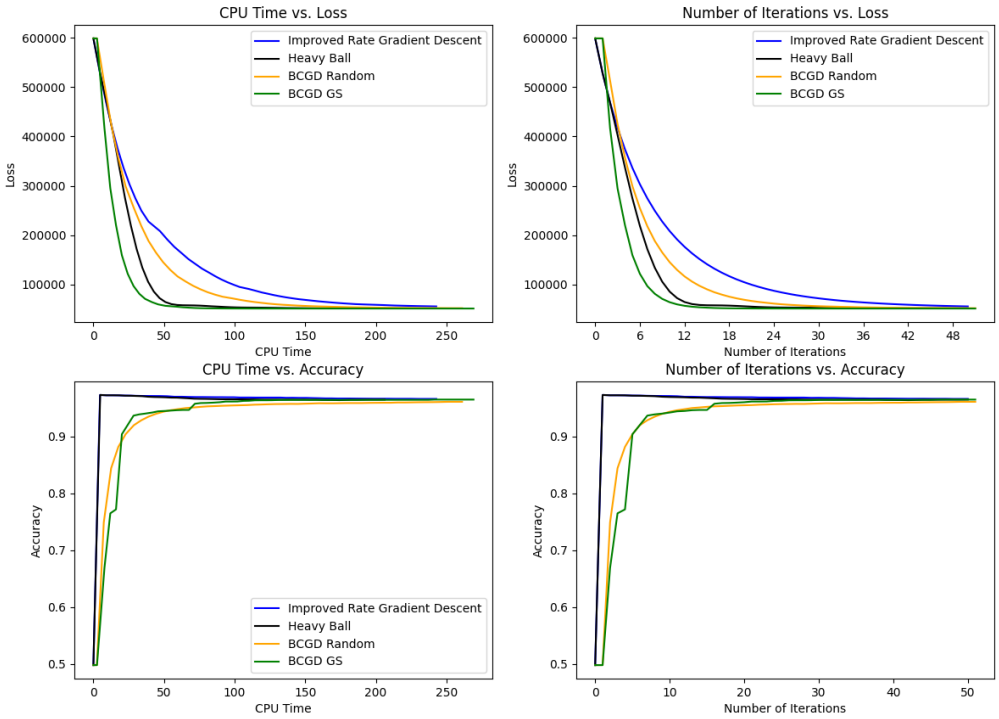

Figure 12: Skin-Cancer Dataset: Comparison of all optimization models

	Algorithm	*Accuracy score (%)*	*F1 score*
Synthetic Dataset	*Gradient Descent with Improved Rate*	100	1.00
	Heavy Ball Gradient Descent	100	1.00
	BCGD Randomized	100	1.00
	BCGD Gauss-Southwell	100	1.00
Skin Cancer Dataset	*Gradient Descent with Improved Rate*	96.6	0.97
	Heavy Ball Gradient Descent	96.8	0.97
	BCGD Randomized	96.1	0.96
	BCGD Gauss-Southwell	96.5	0.97

Table 1: Performance comparisons between algorithms: Accuracy and F1 score

6. Discussion

We initially implement our algorithms on synthetic data that we have randomly generated, and we take note of the good performance of said algorithms expressed in our loss and accuracy metrics. Moreover, we can further prove the efficacy of our algorithms when applied to real-life data.

After visualizing our results, we observe that the Block Coordinate Gradient Descent: Gauss-Southwell algorithm performs the best overall amongst all the tested algorithms in both synthetic and skin segmentation datasets. This high performance can be detected in the faster convergence of the loss function per unit time and per unit of iterations. However, when it comes to accuracy score and F1 score, we notice that both Gradient Descent algorithms outperform the BCGD algorithms.

When reviewing the comparison plots for the loss of all the models more closely in the Skin Segmentation dataset, we find the following ascending performance across both CPU time and iterations, where improved gradient descent performs the worst, followed by BCGD Random, heavy ball, and finally BCGD Gauss-Southwell. Meanwhile, we note the following ascending performance of accuracy: BCGD Random performs the worst, followed by BCGD Gauss-Southwell, and finally, both gradient methods perform equally well at the top.

In conclusion, our work can be expanded by considering alternative values for our hyperparameters such as the learning rate or the standard deviation σ in the similarity matrices' computation. Additionally, we can implement other variations of the algorithms we have tested, such as the BCGD Cyclic or the Nesterov Accelerated Gradient Descent.