Onur Alp Güvercin
11.05.2022

LABELING WITH OPTIMIZATION ALGORITHMS FOR SEMISUPERVISED LEARNING

Abstract:
This report focuses on the implementation and evaluation of three optimization algorithms, namely Standard Gradient Descent (GD), Randomized Gradient Descent (RBCGD), and Gauss-Southwell Gradient Descent (Gauss-Southwell GD), in the context of a semi-supervised learning task. The algorithms were applied to two datasets: a toy dataset and the widely used MNIST dataset from Kaggle. The evaluation in this report is on the toy dataset for simplicity.

The initial implementation and evaluation were conducted on the toy dataset to ensure the correctness and convergence of the algorithms. Promising results were obtained, with Standard GD achieving an accuracy of 0.998 and a loss of 0.173. Randomized BCGD exhibited even higher accuracy at 0.998 with a loss of 0.570, while Gauss-Southwell BCGD demonstrated exceptional accuracy, achieving a perfect score of 1.000 with a loss of 2.916.

To assess the scalability and performance on a larger and more complex dataset, the algorithms were further evaluated on the MNIST dataset. This analysis provided insights into their effectiveness and efficiency in the context of semi-supervised learning.

Additionally, the report presents the computational time required by each algorithm, with GD, RBCGD, and Gauss-Southwell GD taking 296.9 seconds, 56.8 seconds, and 89.2 seconds, respectively.

1. Introduction:

This project focuses on the implementation and evaluation of a semi-supervised learning approach for accurate data labeling. The initial stage involved the creation of a toy dataset consisting of 2000 data points with two class labels. Following the development of the core code, a flexible function was introduced to allow the selection of two datasets: the toy dataset and the widely used MNIST dataset from Kaggle.

For MNIST dataset, preprocessing steps were applied to prepare the data for analysis. The MNIST dataset was flattened, normalized, and reduced to a 2-dimensional representation using Principal Component Analysis (PCA). Moreover, the classification labels were reduced to two classes: if an image represented the digit 0, the label was set to 0; otherwise, the label was set to 1.

In order to simulate a semi-supervised learning scenario, a significant portion of the data points (90%) was randomly unlabeled. This unlabeled subset was chosen for further processing, aiming to assign accurate labels by minimizing a loss function.

2. Explanations of Some Important Functions

2.1 Loss function:

This loss function quantifies the dissimilarity between the labeled and unlabeled data points, taking into account the respective weights. The goal is to minimize this loss function to accurately label the unlabeled data points in the semi-supervised learning task.

The loss is calculated in two parts:
The first double sum calculates the squared difference between unlabeled data points (Y_unlabeled) and labeled data points (Y_labeled), weighted by w_labeled_unlabeled.

The second double sum calculates the squared difference between pairs of unlabeled data points (Y_unlabeled), weighted by w_unlabeled_unlabeled.

The two parts are then summed together, with the second part divided by 2, resulting in the final loss value.

$$f(y) = \sum_{i=0}^{l} \sum_{j=0}^{u} w_{ij}(y^j - \bar{y}^i)^2 + \frac{1}{2} \sum_{i=0}^{u} \sum_{j=0}^{u} \bar{w}_{ij}(y^j - y^i)^2$$

2.2 Weight Matrix Functions:

Euclidean distances are computed between the data points. These distances are then transformed using an exponential decay formula, where the squared distances are multiplied by -10 and passed through the exponential function. The resulting values represent the weights assigned to each data point pair.

The weights obtained from these functions are important for the loss function used in the semi-supervised learning task. They determine the impact of each data point pair on the overall loss calculation, helping to guide the optimization process.

2.3 Gradient functions:

These functions play a key role in determining the gradients needed for optimizing the parameters. By considering the weighted differences between data points, they capture the relevant information required for guiding the optimization process towards a better solution. Here 2 different functions are created, one (non-iterative one) is used in GD and other (iterative one) is used in RBCGD and GS-BCGD.

$$\nabla_{y^i} f(y) = 2 \sum_{i=0}^{l} w_{ij}(y^j - \bar{y}^i) + \sum_{i=0}^{u} \bar{w}_{ij}(y^j - y^i)$$

2.4 Hessian matrix and Lipschitz Constant:

These functions estimate the Lipschitz constant, which determines the step size or learning rate during optimization. By computing the Lipschitz constant, an appropriate step size can be chosen to ensure convergence and stability during optimization.

hessian_matrix (): This function computes the Hessian matrix used in determining the Lipschitz constant. It initializes the Hessian matrix by copying the negative values of w_unlabeled_unlabeled. Then, for each unlabeled index, it updates the diagonal element of the Hessian matrix with a combination of weighted sums. The resulting Hessian matrix is returned. Here only diagonal elements are updated because after some calculations I realized that the only different entries in Hessian from negative w_unlabeled_unlabeled matrix is the diagonal elements.

calculate_lipschitz_constant(hessian_matrix): This function calculates the Lipschitz constant, which represents the maximum eigenvalue of the provided Hessian matrix. It utilizes the eigh() function to compute the eigenvalues of the Hessian matrix. The Lipschitz constant is obtained by finding the maximum eigenvalue. Finally, the Lipschitz constant is returned.

$$\frac{\partial}{\partial y_1}(\nabla_{y^l} f(y))\$ = 2\sum_{i=0}^{l} w_{i1} + \sum_{i=0}^{u} \bar{w}_{i1} - \bar{w}_{i1}$$

$$\frac{\partial}{\partial y_2}(\nabla_{y^l} f(y)) = -\bar{w}_{21}$$

3. Optimization Algorithms:

3.1 Gradient Descent (GD):

In this project phase, the primary objective was to minimize the loss function and improve the accuracy of the labeled data through the implementation of the gradient descent (GD) algorithm. The GD algorithm was employed to iteratively update the model parameters in order to minimize the loss and enhance the quality of the labeled labels.

To achieve this objective, the Armijo line search method was utilized to determine the appropriate step size for each iteration of the GD algorithm. By dynamically adjusting the step size, the algorithm aimed to efficiently navigate the loss landscape and converge towards the optimal solution.

The evaluation of the algorithm's performance included monitoring the accuracy achieved and the computational time required. The accuracy metric provided a measure of the algorithm's success in accurately labeling the data points. Meanwhile, the CPU time measurement reflected the computational cost associated with the loss calculation and step size determination performed in each iteration.

It is worth noting that the observed CPU time was slightly longer than anticipated. This can be attributed to two main factors. Firstly, the algorithm calculates the loss function at each iteration, which involves significant computational effort. Additionally, the step size is recalculated using the Armijo line search method in every iteration, further contributing to the overall computational time.

CPU times: user 4min 15s, sys: 41.7 s, total: 4min 56s Wall time: 5min 17s

```
iteration: 0 --- accuracy: 0.504 ---- loss: 45099.387
iteration: 20 --- accuracy: 0.993 ---- loss: 30.626
iteration: 40 --- accuracy: 0.998 ---- loss: 4.631
iteration: 60 --- accuracy: 0.998 ---- loss: 1.489
iteration: 80 --- accuracy: 0.998 ---- loss: 0.637
iteration: 100 --- accuracy: 0.998 ---- loss: 0.370
iteration: 120 --- accuracy: 0.998 ---- loss: 0.278
iteration: 140 --- accuracy: 0.998 ---- loss: 0.236
iteration: 160 --- accuracy: 0.998 ---- loss: 0.210
iteration: 180 --- accuracy: 0.998 ---- loss: 0.189
iteration: 200 --- accuracy: 0.998 ---- loss: 0.173
CPU times: user 4min 15s, sys: 41.7 s, total: 4min 56s
Wall time: 5min 17s
```

3.2 Randomized BCGD (RBCGD):

In this part of the project, the focus was on implementing the Randomized Block Coordinate Gradient Descent (RBCGD) algorithm to update the labels. The RBCGD algorithm follows an iterative approach where one random block is selected at a time. The gradient is calculated for that block, and the corresponding values are updated using an appropriate step size (1/L, where L is the Lipschitz constant). The Lipschitz constant was determined by computing the Hessian matrix and extracting the maximum eigenvalue.

The iterative nature of the algorithm allows for efficient updates of the labels by focusing on specific blocks at each iteration. This approach balances computational efficiency and optimization accuracy.

It is important to note that due to the iterative nature of the algorithm and the additional computational steps involved, the CPU time required for RBCGD was slightly longer than expected. However, this trade-off is necessary to accurately optimize the labels and improve the performance of the semi supervised learning task.

CPU times: user 51.5 s, sys: 5.35 s, total: 56.8 s Wall time: 57.1 s

```
iteration: 0 --- accuracy: 0.998 ---- loss: 0.173
iteration: 2 --- accuracy: 0.617 ---- loss: 21496.419
iteration: 20 --- accuracy: 0.984 ---- loss: 253.069
iteration: 40 --- accuracy: 0.992 ---- loss: 36.781
iteration: 60 --- accuracy: 0.994 ---- loss: 11.647
iteration: 80 --- accuracy: 0.998 ---- loss: 5.661
iteration: 100 --- accuracy: 0.998 ---- loss: 3.149
iteration: 120 --- accuracy: 0.998 ---- loss: 1.970
iteration: 140 --- accuracy: 0.998 ---- loss: 1.382
iteration: 160 --- accuracy: 0.998 ---- loss: 0.995
iteration: 180 --- accuracy: 0.998 ---- loss: 0.727
iteration: 200 --- accuracy: 0.998 ---- loss: 0.570
CPU times: user 51.5 s, sys: 5.35 s, total: 56.8 s
Wall time: 57.1 s
```

3.3 Gauss-Southwell BCGD:

In the final part of the project, the Gauss-Southwell Block Coordinate Gradient Descent (Gauss-Southwell BCGD) approach was implemented to update the labels. Unlike the Randomized BCGD algorithm, Gauss-Southwell BCGD considers the gradients of all points, and identifies the dimension with the steepest descent, then it updates the corresponding point using an appropriate step size derived from the Lipschitz constant. The Lipschitz constant was determined by computing the Hessian matrix and extracting the maximum eigenvalue.

This approach ensures that the updates are made in the direction that leads to the most significant improvement in the loss function. While the Gauss-Southwell BCGD algorithm offers enhanced accuracy and convergence properties, it requires more computational time compared to other methods. The additional computational time can be attributed to the calculation of gradients for all points and the careful selection of the optimal direction for each update.

CPU times: user 1min 23s, sys: 5.31 s, total: 1min 29s Wall time: 1min 29s

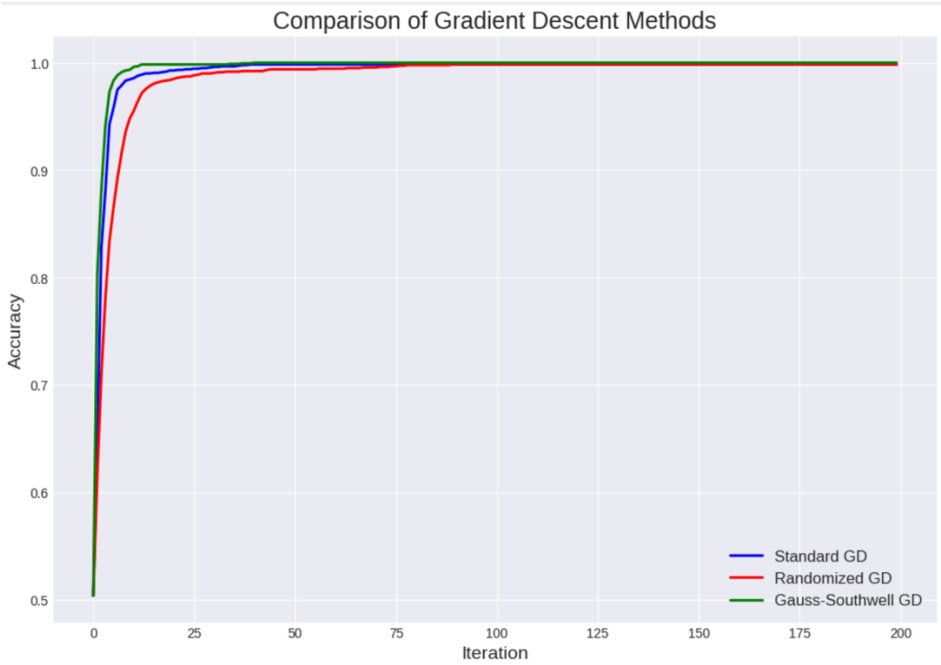
```
iteration: 0 --- accuracy: 0.998 ---- loss: 0.173
iteration: 2 --- accuracy: 0.803 ---- loss: 8836.722
iteration: 20 --- accuracy: 0.998 ---- loss: 140.720
iteration: 40 --- accuracy: 0.999 ---- loss: 57.047
iteration: 60 --- accuracy: 1.000 ---- loss: 31.514
iteration: 80 --- accuracy: 1.000 ---- loss: 21.451
iteration: 100 --- accuracy: 1.000 ---- loss: 13.457
iteration: 120 --- accuracy: 1.000 ---- loss: 7.412
iteration: 140 --- accuracy: 1.000 ---- loss: 5.497
iteration: 160 --- accuracy: 1.000 ---- loss: 4.164
iteration: 180 --- accuracy: 1.000 ---- loss: 3.615
iteration: 200 --- accuracy: 1.000 ---- loss: 2.916
CPU times: user 1min 23s, sys: 5.31 s, total: 1min 29s
Wall time: 1min 29s
```

4. Conclusion:

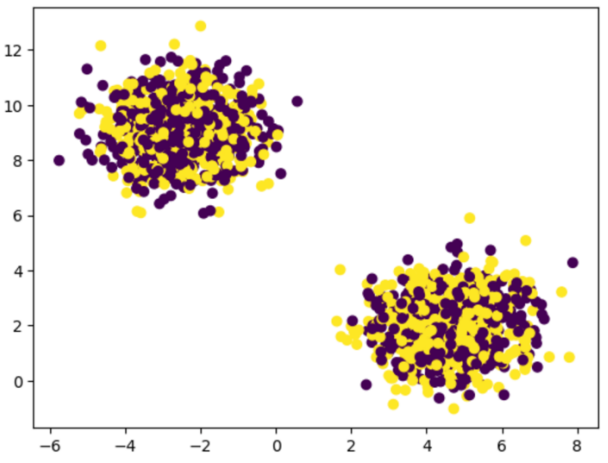
In conclusion, this project aimed to improve label accuracy in a semi supervised learning task using optimization algorithms. The implemented algorithms, including Standard Gradient Descent (GD), Randomized Gradient Descent (RBCGD), and Gauss-Southwell Gradient Descent (Gauss-Southwell GD), demonstrated their effectiveness in achieving high accuracy.

The results showed that all three algorithms produced impressive accuracy values, with Standard GD achieving 0.998, RBCGD achieving 0.998, and Gauss-Southwell GD achieving a perfect score of 1.000. The corresponding loss values were 0.173, 0.570, and 2.916, respectively. However, it is important to note that the computational time varied among the algorithms. RBCGD had a CPU time of 56.8 seconds, Gauss-Southwell GD had a CPU time of 89.2 seconds, while Standard GD had the longest CPU time of 296.9 seconds.

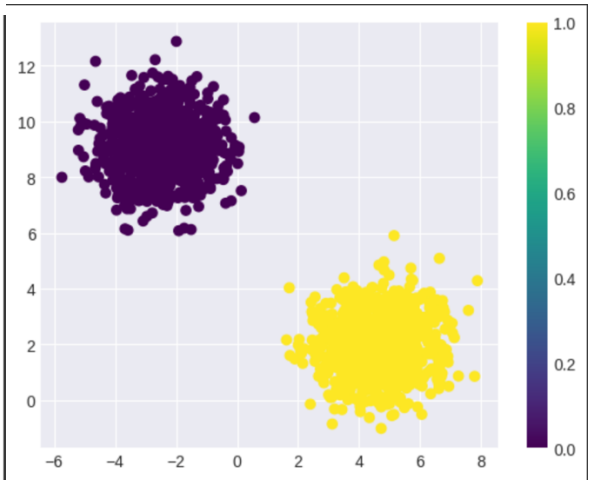
The increased computational time can be attributed to the iterative nature of the algorithms, which involve loss calculation and step size determination in each iteration. In summary, this project successfully demonstrated the effectiveness of optimization algorithms in improving label accuracy in a semi supervised learning task.



---------- Comparison of loss functions ----------



---------- Before the algorithm ----------



---------- After the algorithm ----------