Homework Report for Optimization for Data Science Course

Gradient Based Unconstrained Optimization Methods for Semi Supervised Learning

Anna Badalyan

14/05/2023

Contents

1 Introduction

The aim of this report is to implement the unconstrained optimization algorithms namely Gradient Descent, Block Coordinate Gradient Descent (BCGD) with randomized rule and BCGD with Gauss-Southwell rule. The work is based on the last year implementation done by Abhishek Varma Dasaraju, Anna Badalyan, Brenda Eloısa Tellez Juarez and Rebecca Di Francesco and aims at improving the performance of implemented algorithms. The major improvement was achieved for the randomized BCGD method by correcting the mistake in the gradient calculation which caused an increase in the values of objective function. The usage of the stepsize based on the Lipschitz constant L_i for each coordinate also improved the performance on the randomized BCGD method. In addition, we also implemented Nesterov's sampling for the randomized BCGD method. The stopping condition as the product of the gradient with its direction was implemented. As a result, this work significantly improved the performance of the BCGD methods on a real dataset compared to last year's implementation.

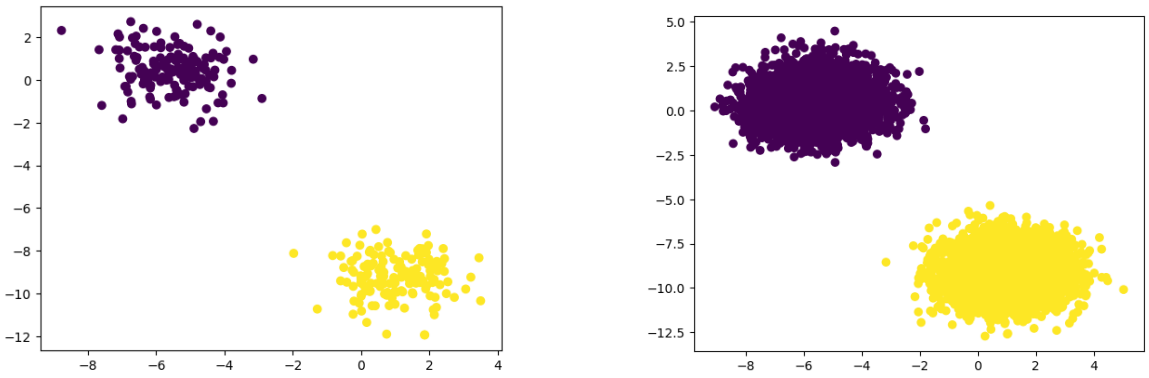
2 Task

The task is to classify unlabelled data points into 2 classes. For this task we generated 10,000 data points clearly separable in 2 clusters labelled as -1, 1. We kept labels for the 3% of the data points, while the rest 97% remained unlabelled.

To classify the unlabelled points we need to solve the following minimization problem:

$$\min_{y \in \mathbb{R}^n} \sum_{i=1}^{l} \sum_{j=1}^{u} w_{ij}(y^j - \overline{y}^i)^2 + \sum_{i=1}^{u} \sum_{j=1}^{u} \overline{w}_{ij}(y^i - y^j)^2$$

where $\overline{y}$ is the vector of labels for labelled points, w is the similarity matrix between the points i and j and y is the vector of labels for unlabelled points that we need to find.



(a) Labelled data points

(b) Unlabelled data points

Figure 1: Generated blobs of data

3 Implementation

3.1 Similarity function

We used Gaussain weights as a similarity measure where the similarity between 2 points i and j is defined as follows:

$$w_{ij} = exp(\frac{-d(x_i, x_j)^2}{\epsilon^2})$$

where d is the euclidean distance between points defined as:

$$d(a,b) = \sqrt{\sum_{i}^{n}(a_i - b_i)^2}$$

The value ϵ was chosen at 0.5.

3.2 Stepsize

Similarly to the last year implementation, we choose a fixed stepsize for all three algorithms. To chose the value of the stepsize we estimated the Lipschitz constant of the gradient. As the function $f(y)$ is convex and twice differentiable, we can bound the Lipschitz constant as follows:

$$\nabla^2 f(y) \preceq LI$$

As the Hessian matrix is positive semi-definite, its norm is equal to the largest eigenvalue

$$\|\nabla^2 f(y)\| = \lambda_{max} \leq L$$

Thus, we can lower bound the Lipschitz constant with the largest eigenvalue of the Hessian matrix. In practice we increased this value to obtain faster convergence.

The Hessian matrix is defined as follows for the diagonal entries:

$$\frac{\partial^2 f(y)}{\partial(y_j)(y_j)} = 2\sum_{i=1}^{l} w_{ij} + 2\sum_{i=1}^{n} \overline{w}_{ij}$$

for the non-diagonal entries:

$$\frac{\partial^2 f(y)}{\partial(y_j)(y_i)} = -2\overline{w}_{ij}$$

3.2.1 Lipschitz constant for coordinates

We calculated the Lipschitz constant for each coordinate as follows. By definition, the the Lipschitz continuous gradient with constant L_k is defined as:

$$\|\nabla f(y + U_k h_k) + \nabla f(y)\| \leq L_k \|h_k\|, \forall h_k \in \mathbb{R}^{n_k} \forall y \in \mathbb{R}^n$$

The derivative $\nabla_{y^j} f(y)$ is given as:

$$\nabla_{y^j} f(y) = 2\sum_{i=1}^{l} w_{ij}(y^j - \overline{y}^i) + 2\sum_{i=1}^{n} \overline{w}_{ij}(y^j - y^i)$$

When we add h_k to the coordinate k the derivative will be as follows for the coordinate $j \neq k$

$$\nabla_{y^j, j\neq k} f(y + U_k h_k) = 2\sum_{i=1}^{l} w_{ij}(y^j - \overline{y}^i) + 2\sum_{i=1, i\neq k}^{n} \overline{w}_{ij}(y^j - y^i) + 2\overline{w}_{kj}(y^j - y^k - h_k) \quad (1)$$

$$= 2\sum_{i=1}^{l} w_{ij}(y^j - \overline{y}^i) + 2\sum_{i=1}^{n} \overline{w}_{ij}(y^j - y^i) - 2\overline{w}_{kj} h_k \quad (2)$$

While for the coordinate k the derivative is

$$\nabla_{y^k} f(y + U_k h_k) = 2\sum_{i=1}^{l} w_{ik}(y^k + h_k - \overline{y}^i) + 2\sum_{i=1}^{n} \overline{w}_{ij}(y^k + h_k - y^i) \tag{3}$$

$$= 2\sum_{i=1}^{l} w_{ik}(y^k - \overline{y}^i) + 2\sum_{i=1}^{n} \overline{w}_{ik}(y^k - y^i) + 2\sum_{i=1}^{l} w_{ik}h_k + 2\sum_{i=1}^{n} \overline{w}_{ik}h_k \tag{4}$$

Thus, we can see that the j-th entry of the vector $\nabla f(y + U_k h_k) + \nabla f(y)$ is given as

$$\begin{cases} -2\overline{w}_{kj}h_k & \text{for} \quad j \neq k \\ 2\sum_{i=1}^{l} w_{ik}h_k + 2\sum_{i=1}^{n} \overline{w}_{ik}h_k & \text{for} \quad k = j \end{cases}$$

In our case, h_k is a one dimensional vector, and it's norm can be treated as scalar. So the vector $\nabla f(y + U_k h_k) + \nabla f(y)$ can be written as $h_k v_k$, where v_k is defined as:

$$\begin{cases} -2\overline{w}_{kj} & \text{for} \quad j \neq k \\ 2\sum_{i=1}^{l} w_{ik} + 2\sum_{i=1}^{n} \overline{w}_{ik} & \text{for} \quad k = j \end{cases}$$

We have that

$$h_k\|v_k\| \leq L_k h_k$$
$$\|v_k\| \leq L_k$$

So the Lipschitz constant L_k is bounded by the norm of the vector v_k which we can notice is the k-th column of the Hessian matrix.

In practice, we obtained better convergence by increasing the values of L_k which are smaller than a certain threshold.

3.3 Stopping criteria

We used the product of the gradient with the descent direction as a stopping criteria. For the Gradient Descent algorithms we divided the dot product between the 2 vectors by the number of observations. For the BCGD with randomized rule we summed the products of each n iterations and divided them by n as without averaging the random updates, it happened that a small update was encountered and the algorithms stopped.

We tried to implement the stopping criteria based on the objective function value, however, this approach was expensive for methods based on BCGD. While evaluating the value of the objective function was too costly, we calculated the updates that could be done after the update of each coordinate, however, the numpy implementation of thousands of these updates was numerically unstable and produced incorrect results when the algorithm converged. Thus, we continued using only the product of the gradient with the descent direction to stop the algorithms.

4 Algorithms and experiments

4.1 Gradient Descent

To efficiently implement the Gradient method we implemented functions to compute the objective function and the full gradient using matrix multiplications in numpy. This significantly improved the convergence time of out algorithms compared to the implementation using for loops.

The convergence of the Gradient method could be improved by increasing the stepsize almost twice from $\frac{1}{L}$ to 0.00039897.

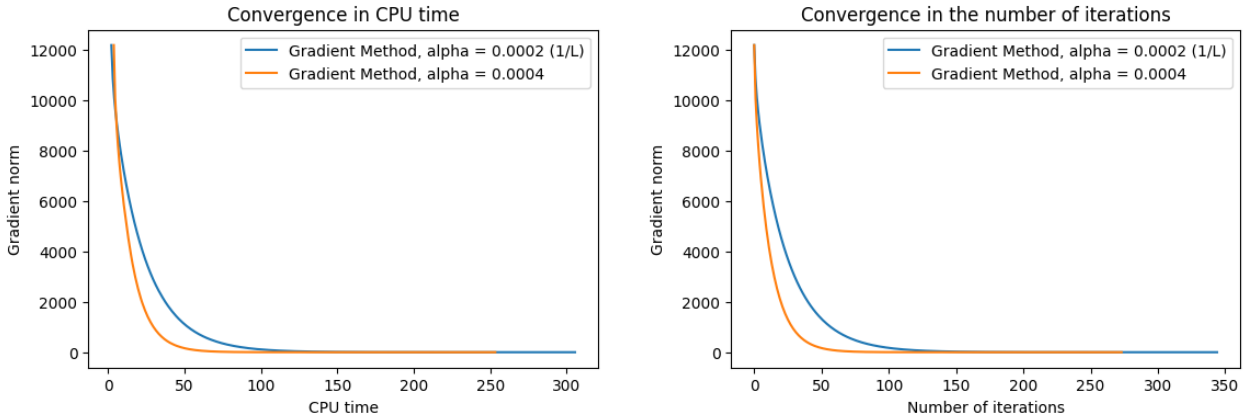


Figure 2: Gradient method convergence with different stepsizes

4.2 BCGD with randomized rule

We implemented 2 versions of the randomized BCGD algorithms. For the first implementation, we sampled the coordinates from the discrete uniform distribution, while for the second one we used non-uniform sampling according the Nesterov's suggestion choosing the coordinate with the largest block more often. The probability of choosing the i-th block is defined as follows:

$$P(i_k = i) = \frac{L_i}{\sum_{i=1}^{b} L_i}$$

Randomized BCGD with the stepsize $\alpha_k = \frac{1}{L_k}$ showed the best results in terms of both CPU time and the number of iterations needed to converge. Fixed stepsize for all coordinates equal to 0.00033 showed similar convergence results. Meanwhile, the stepsize of $\frac{1}{L}$ required the most iterations to convergence.

Nesterov's sampling produced similar results to the uniform sampling with $\alpha_k = \frac{1}{L_k}$ in terms of the number of iteration required to converge, however, sampling with a certain probability distribution took twice more time than uniform sampling.

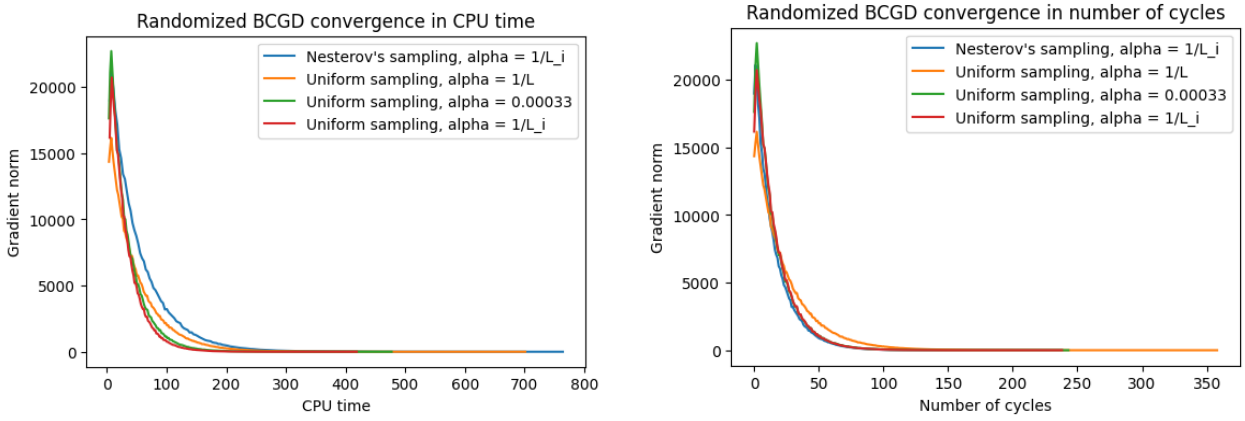


Figure 3: Randomized BCGD method convergence with different stepsizes

4.3 BCGD with Gauss-Southwell rule

To choose the coordinate with the largest value of the gradient we need to have the information about the gradient at each iteration of the algorithms. However, it is very costly to compute the full gradient

at each iteration. That's why, we computed the gradient only in the beginning and used the change derived in equation 1 to update other value of the gradient after the coordinate k was changed. In the end we recomputed the value of the gradient with respect to the chosen coordinate.

We can see in Figure 4 that increasing the stepsize helps to significantly improve the performace of the BCGD with Gaus-Soouthwell rule. The best performance was achieved with the $\alpha_k = \frac{1}{L_k}$ with the minimum α_k set to 0.00039897.

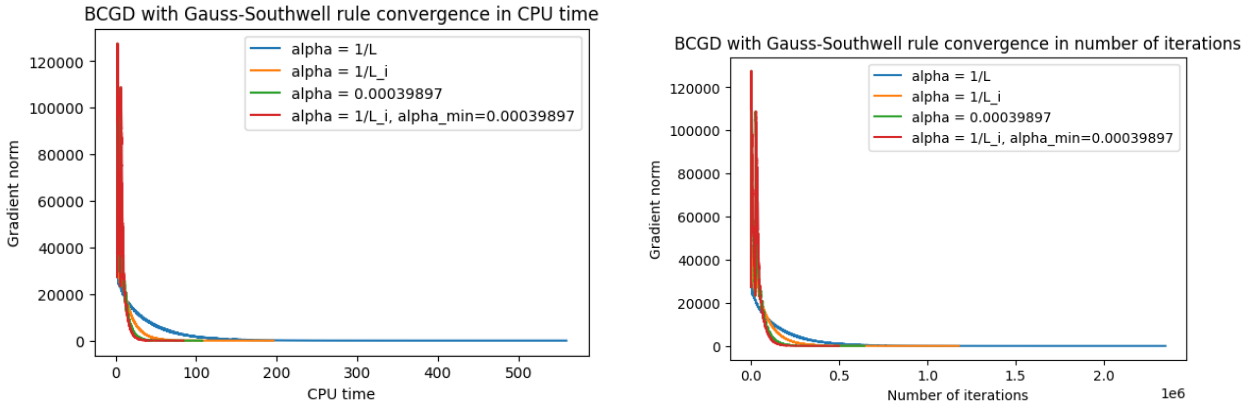
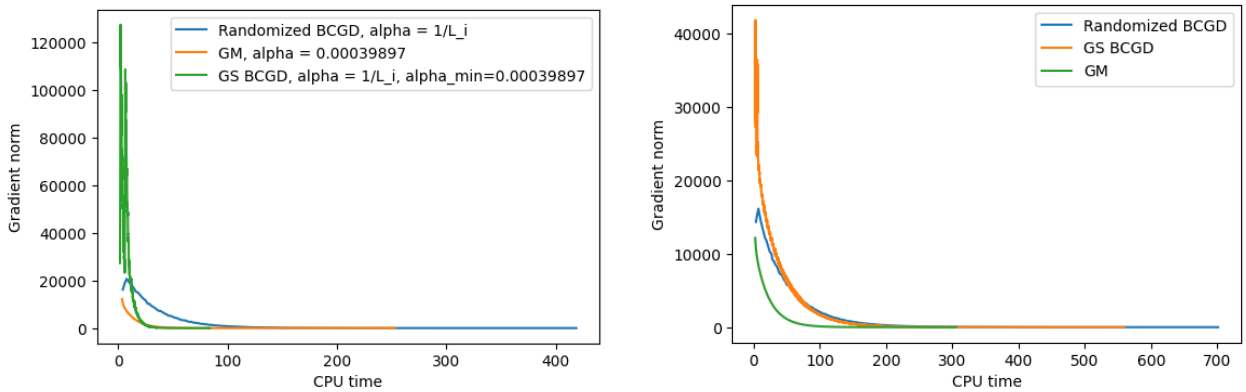


Figure 4: BCGD with Gauss Southwell rule convergence with different stepsizes

4.4 Method comparison

We can see the performance comparison of various methods in Figure 5. As expected, the BCGD method with Gauss-Southwell rule outperforms other algorithms when the stepsize is well chosen. It's more than 4 times faster than randomized BCGD and more than 2 times than Gradient method in reaching convergence. During the first iterations the gradient norm is particularly large which allows the algorithm to reach to the minimum faster.

With the same stepsize equal to $\frac{1}{L}$, Gradient method outpereforms other methods in terms of CPU time.



(a) Best method convergence in CPU time

(b) Algorithms convergence with alpha = 1/L

Figure 5: BCGD with Gauss Southwell rule convergence with different stepsizes

All methods produced 100% accuracy classifying the unlabelled points. The value of the objective function was decreased to almost 0 with all algorithms.

5 Real dataset

The implemented methods were further tested on the citrus dataset which contained 10,000 samples divided in 2 classes, oranges and grapefruits. Each observation contained 5 features namely diameter, weight, red, green, blue.

Initially, the data was normalized and scaled within the range of -100 to 100, in order to increase the sparsity of the points. Since the points were not well separable, the selection of a suitable similarity function was deemed crucial. A series of experiments were conducted in order to identify the optimal value for gamma, with values of 1, 0.2, 0.02, 0.01, 0.005 and 0.001 being tested. The rbf kernel with a gamma = 0.001 produced better outcomes.

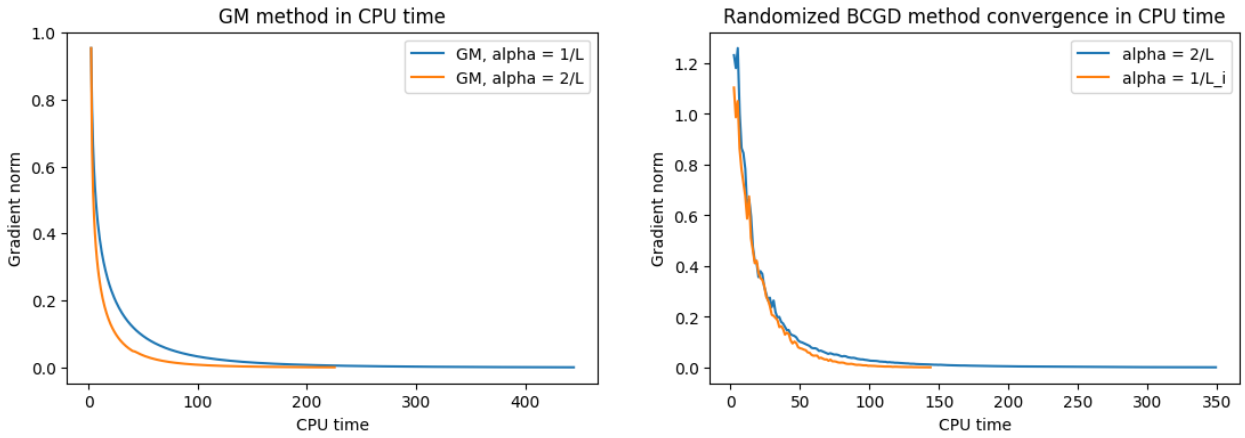


Figure 6: GM and randomized BCGD convergence in CPU time

Similarly to the generated dataset, the Gradient method performs better with the stepsize that is about twice as large as the $\frac{1}{L}$. The randomized BCGD method performs better with $\alpha_k = \frac{1}{L_k}$ than with $\alpha = \frac{1}{L}$. Unlike the generated dataset, the randomized BCGD performs better than the Gradient method. Similarly to the generated dataset, BBCGD with Gauth-Southwell rule reaches the best performance among all algorithms.

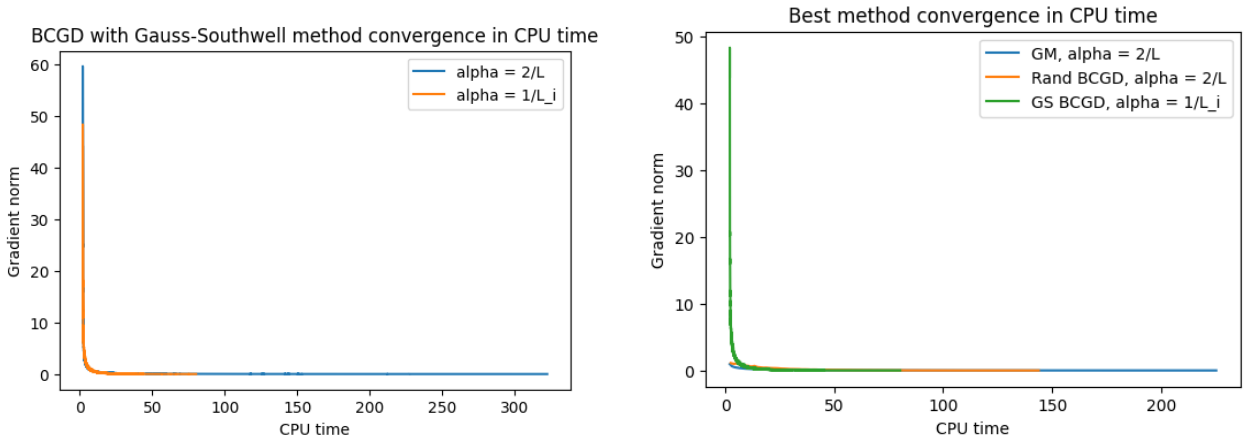


Figure 7: Best method convergence in CPU time

The best algorithms shown in 7 reach the accuracy 92.09%, 92.19% and 91.05% (GM, randomized BCGD, GS BCGD respectively).

6 Conclusions

On both generated and real dataset the BCGD with Gauss-Southwell method showed the best performance when a proper step size strategy is chosen. Randomized BCGD outperformed GM on a real dataset, but showed worse performance on the geneerated dataset. This might be due to the fact that when the point are easily separable, GM performs better, however, in a real world scenario and when the number of variables is large, randomized BCGD performs better.