Homework Optimization for Data Science

Caregari Alberto Mincato Emanuele Schiavo Leonardo

Abstract

We are dealing with a Semisupervised Learning problem. We simulate a scenario where we have high number of data where only few of them are labelled. The goal is to assign to all the points the right label by some optimization algorithm.

1. Introduction

We simulated this case generating two clouds of points on the plane and randomly choosing few of them to create the labelled set of points. Then we used methods such as Gradient Descent, Randomized Block Coordinate Gradient Descent (BCGD) and Cyclic BCGD with blocks of dimension 1. The goal is to properly classify all the points between the two sets and test the accuracy of the implemented methods. After this first section we tested the methods on a real dataset for a problem of gender classification.

2. Toy Dataset

The Toy dataset is composed of 10000 points. With a probability of 50% a point is sampled by a normal distribution $N((0,0),1)$ and after it is translated by a positional vector $\mu = (-0.37, 2.64)$, or $-\mu$, that we chose arbitrarily. We decided to normalize the dataset with the min-max normalizing function. In practice this creates two clouds of points with roughly 5000 points each. We now assume to know only 1.5% of the total labels. Below it is shown the initial situation of the toy dataset created with this approach, where light blue and green represents the labelled points of the two different sets.

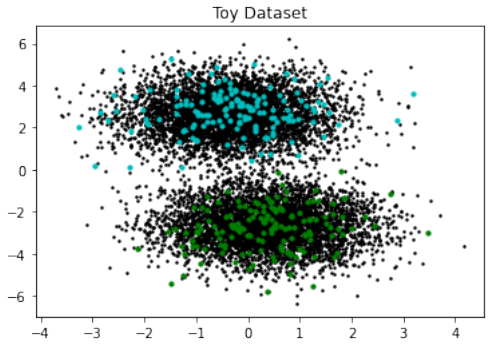

Figure 1. Initial situation of the Toy dataset

3. Real Dataset

The real Dataset we choose for our analysis is "Gender Classification Dataset" [1], provided by Kaggle. This dataset contains 7 features and one label for a sample of 5000 observations. A brief description of the features:

1. long hair: this column contains 0 and 1 where 1 is "long hair" and 0 is "not long hair";
2. forehead width: this is the width of the forehead measured in cm;
3. forehead height: this is the height of the forehead and it is in cm;
4. nose wide: this column contains 0 and 1 where 1 is "wide nose" and 0 is "not wide nose";
5. nose long: this column contains 0 and 1 where 1 is "Long nose" and 0 is "not long nose";
6. lips thin: this column contains 0 and 1 where 1 represents the "thin lips" while 0 is "not thin lips";
7. distance nose to lip long: this column contains 0 and 1 where 1 represents the "long distance between nose and lips" while 0 is "short distance between nose and lips".

The label column represents the target gender of the observation and could be either "Male" or "Female". As done previously we normalized the data and we assumed to know only 1.5% of all the target labels and use the others for accuracy purposes.

4. Similarity Measure and Loss Function

Given $x, y \in \mathbb{R}^n$ we defined the similarity measure between x and y as

$$K(x,y) = e^{-\gamma||x-y||^2}$$

where γ is proportional to the inverse of the square of the variance (of our kernel) and it is an hyperparameter. We have set this value to one, both in the case we measure the weight between unlabeled points and in the case between labeled points and unlabeled points. We used this particular similarity measure, an exponential decreasing function, so that as the distance between x and y increases, the similarity approaches 0, and on the other hand, as the distance decreases, the similarity approaches 1. As opposed to the reciprocal of the Euclidean distance, this measure has the property of being bounded everywhere, even between overlapping points. We now generated a symmetric weighed matrix W where every entry is the weight between two points. We defined the loss function as

$$\mathcal{L}(y) = \sum_{i=1}^{\ell}\sum_{j=1}^{u} w_{ij}\left(y^j - \bar{y}^i\right)^2 + \frac{1}{2}\sum_{i=1}^{u}\sum_{j=1}^{u} \bar{w}_{ij}\left(y^j - y^i\right)^2$$

where y is the vector of the predicted labels and $\bar{y}$ contains the already existing labels. The goal is to minimize the loss function, i.e. to find

$$\min_{y\in\mathbb{R}^u} \mathcal{L}(y)$$

In order to achieve this result we had plenty of possible options. In this work we focused on first order methods like gradients methods, taking advantage from the smoothness of the objective function. Another significant decision we have to take is the size of the learning step. We used a fixed stepsize method, empirically choosing the step as $\alpha = 0.0001$. Then we implemented Armijo Rule and Exact line search and, as we will see further, using these stepsize methods we obtain better accuracy in less epochs but they are more expensive from an computational point if view. For what concern the Random BCGD Method we computed the loss function every 10000 epochs. We had to do that because computing the loss function for every single update of the gradient is too computational heavy and so we decided to calculate it every 10000 updates as in the Gradient Descent Methods and Cyclic BCGD Method, where the gradient is updated for the whole sample at the same time.

5. Results on the Toy Dataset

We reported the results at the end of the report([6][7][8][9][10]). We plotted loss function over time and accuracy over time, where the red line indicates reaching a 99% accuracy level. In the right side there are a plotted labelled points of the two sets, the red points are misclassified. With all methods we reached an high accuracy in relatively few epochs and reasonable CPU time, as we can see from the table below[6]. Using the Gradient Descent with fixed step-size we need more epochs (34) but less CPU time (10.6sec) compare to GD with Armijo Rule and Exact line search that needs less epochs but more CPU time (5epcs 16.4sec and 6epcs 31.7sec). Even though this two methods are implemented for a better efficiency, when we face some high-dimensional problems the best step-size method is the simplest from a computational point of view, in this case it is the fixed step-size. The performance obtained using Random BCGD and Cyclic BCGD are worse than the original GD, reaching 99% accuracy in way more CPU time (52sec for cyclic and 76sec random).

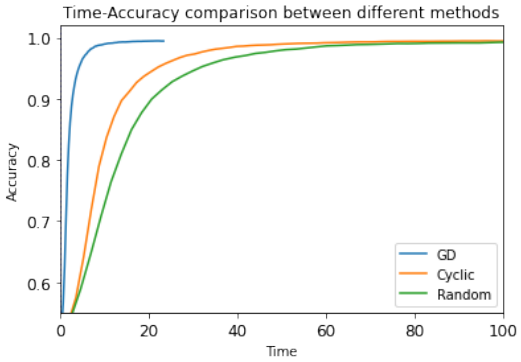

Figure 2. Toy dataset: accuracy time comparison for different methods

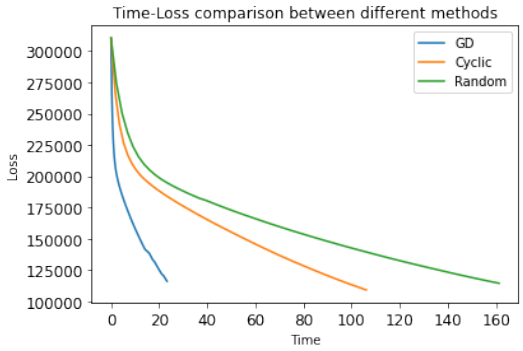

Figure 3. Toy dataset: time loss comparison between different methods

6. Results on the Real Dataset

We observed that the loss function is smooth and convex, this ensures the convergence of the gradient method. We set different epochs threshold according to the convergence rate of these methods, observed in various test. The results are shown at the end of the report, the red line indicates reaching a 95% accuracy level[15]. For this specific problem we need to take into account how expensive are our algorithms, since we are dealing with high dimensional variables. In practise, we found results similar to the toy dataset ones: GD with Armijo Rule and Exact line search after less than 10sec reach more than 95% accuracy and GD with fixed step-size takes only 2.7sec of CPU time to reach the same amount of accuracy. Cyclic and Randomize Gra-

dient Methods perform better in these scenario, but still not getting better results compared to classic Gradient Descent Method.

	Toy (99%)		Gender (95%)	
Alg.	Time	Epochs	Time	Epochs
GD Fix	10.617	34	2.687	31
GD Arm	16.469	5	8.883	10
GD Exa	31.675	6	9.1	4
Random	76.184	33 k	10.512	15 k
Cyclic	52.083	32	5.317	31

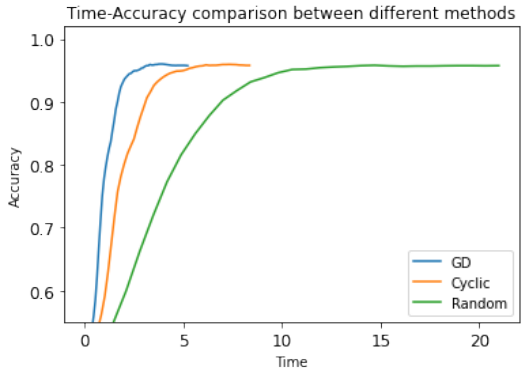

Figure 4. Real dataset: accuracy time comparison between different methods

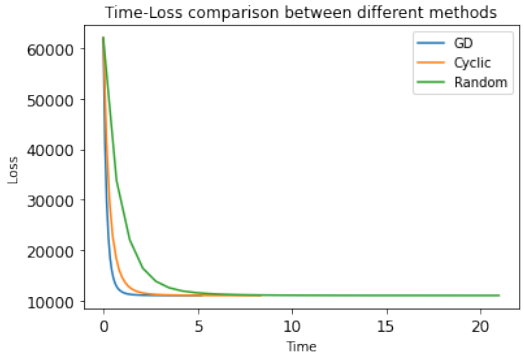

Figure 5. Real dataset: loss time comparison between different methods

7. Conclusions

By observing the plots in Figure [2] and [4] we can state that the classical Gradient Descent is the most efficient methods analyzed by us, both in the case of Toy dataset and Real dataset. We thought that the hardest points to classify in the Toy dataset are the ones in the border between the clouds of points, in particular where the clouds are close. Thus, we are expecting that most of misclassified points are in this area. This is what happened using GD with Armijo Rule and Exact line search. Surprisingly using the Fixed stepsize rule there are mislassified point also in the edges of the clouds, which are far away from the other set so they should be easy to classify correctly. We think that this error is due to the choice of the initial point for our optimization problem: choosing a better initial point or running for more epochs the algorithm should solve the problem. Further studies could be choosing bigger datasets to see better the differences between the algorithms in CPU time and accuracy. Moreover we could find the Lipschitz constant for different loss function to choose a better stepsize.

References

[1] https://www.kaggle.com/datasets/elakiricoder/gender-classification-dataset.

[2] Scikit Organization. User guide scikit-learn.

Toy Dataset Plots

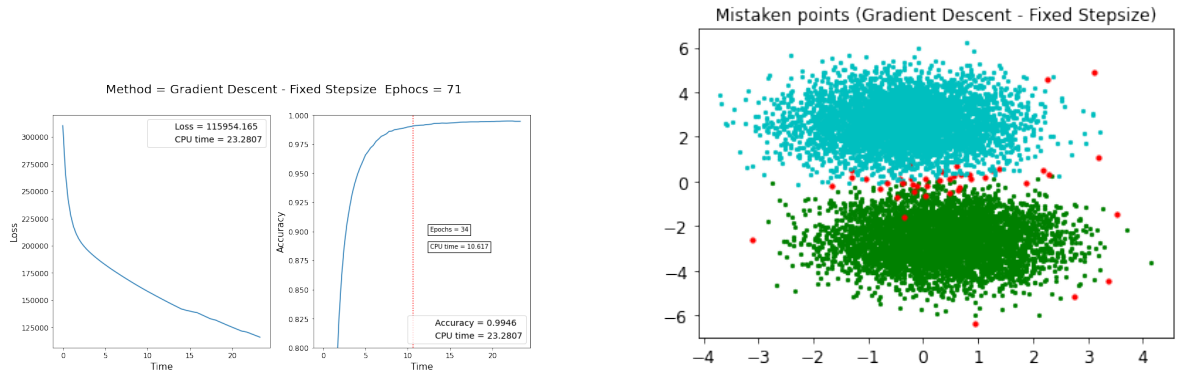

Figure 6. Gradien Descent Method with fixed stepsize

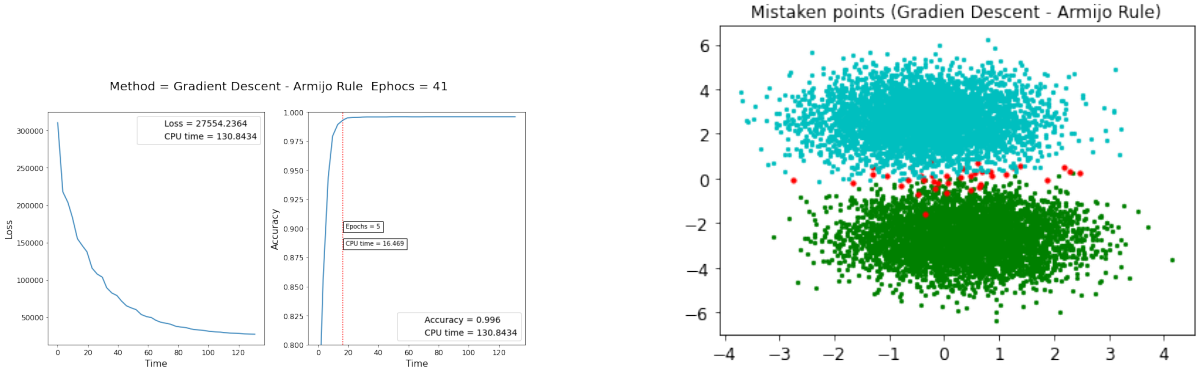

Figure 7. Gradien Descent Method with Armijo Rule

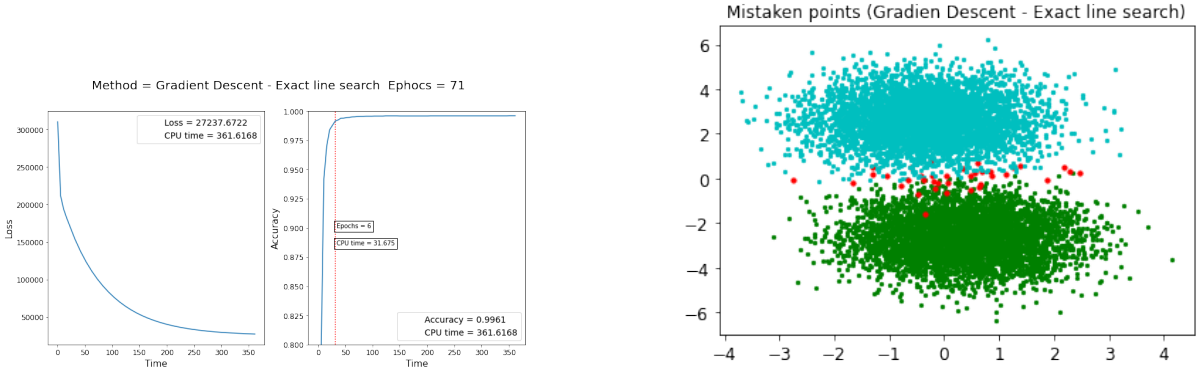

Figure 8. Gradient Descent Method with Exact line search

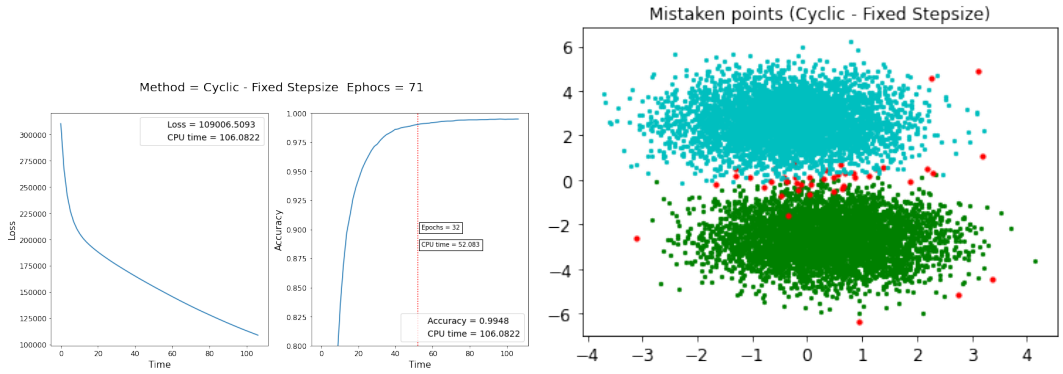

Figure 9. Cyclic Method with Fixed Stepsize

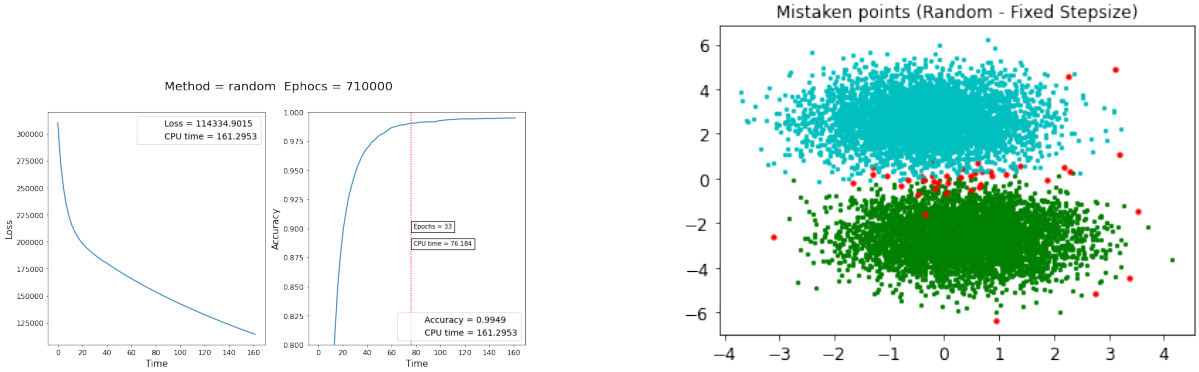

Figure 10. Random Method with Fixed Stepsize

Real Dataset Plots

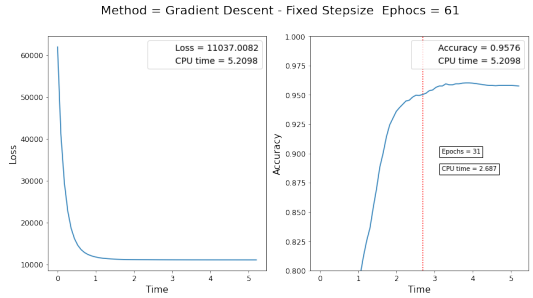

Figure 11. Gradient Descent Method with fixed stepsize

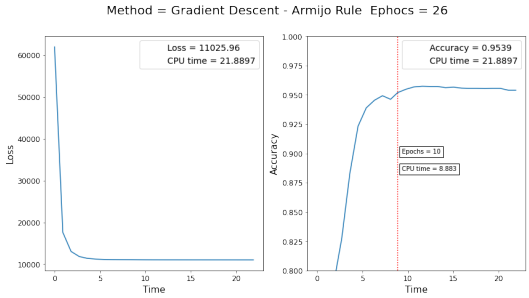

Figure 12. Gradient Descent Method with Armijo Rule

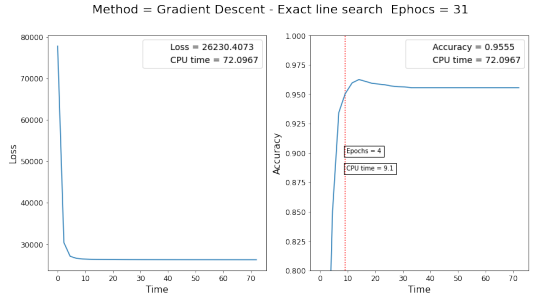

Figure 13. Gradient Descent Method with Exact line search

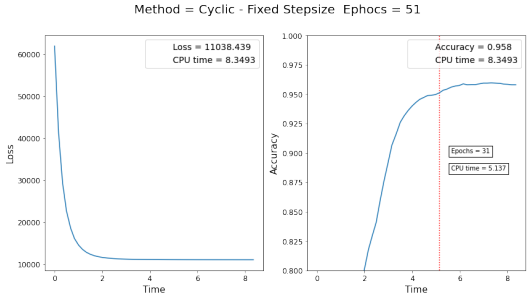

Figure 14. Cyclic Method with Fixed stepsize

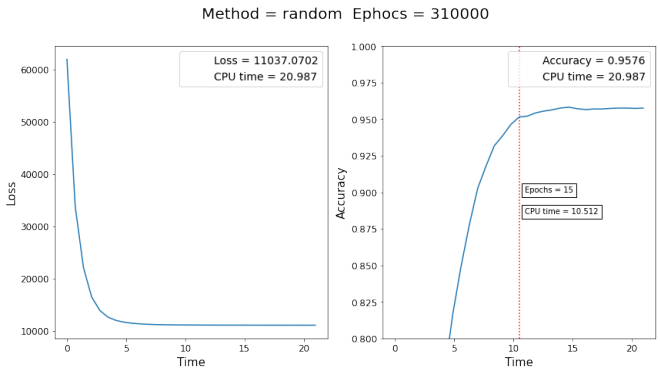

Figure 15. Random Method with Fixed stepsize