Optimization for Data Science - Homework 1

Chiara Bigarella Student nr. 2004248
Silvia Poletti Student nr. 1239133
Gurjeet Singh Student nr. 2004251
Francesca Zen Student nr. 2010640

1 Introduction

The goal of this report is to explore and compare the performances of the Gradient Descent method and the Block Coordinate Gradient Descent methods with randomized and cyclic rules, in Semi-Supervised Learning. For this purpose, we will first consider a convex function minimization problem with respect to randomly generated points in the 2D space. Secondly, we will approach the same problem on a real-world dataset. In particular, the classification task is carried out by examining the feature similarities of the data.

2 Materials

The preliminary part of the analysis focuses on the binary classification of random points in the 2D space. In the second part of our work, we consider a balanced subset of the HTRU2 dataset[1], containing a sample of Pulsar candidates, labeled as Pulsar or Non-Pulsar stars. Each Pulsar candidate is described by some continuous statistic measures (mean, standard deviation, kurtosis and skewness) of the integrated profile and the DM-SRN Curve, for a total of eight features. Data is pre-processed with standardization in order to reduce the variability of the features.

The binary classification task is carried out in Semi-Supervised Learning on a small number of labeled examples $(\bar{x}^i, \bar{y}^i)$ for $i = 1, ..., l$, with labels in $\{-1, 1\}$, and a vast number of unlabeled examples x^j for $j = 1, ..., u$. The goal is to find the optimal labels y^j associated to x^j such that:

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

In particular, $W = [w_{ij}]_{\substack{i=1,...,l \\ j=1,...,u}}$ and $\overline{W} = [\overline{w}_{ij}]_{\substack{i=1,...,u \\ j=1,...,u}}$ are the similarity matrices whose components can be respectively interpreted as follows:

- w_{ij} expresses the similarity between the labeled example $\bar{x}^i$ and the unlabeled example x^j;
- $\overline{w}_{ij}$ expresses the similarity between the unlabeled examples x^i and x^j.

The components of matrix W (and similarly for matrix $\overline{W}$) can be defined by using one of the following Euclidean distance-based similarity measures:

(I) $w_{ij} = \dfrac{1}{1 + \|x^j - \bar{x}^i\|_2}$, where adding 1 to the Euclidean distance at the denominator imposes an upper bound of 1 to the value of w_{ij};

(II) $w_{ij} = \exp\left(\dfrac{-\|x^j - \bar{x}^i\|_2^2}{2\sigma^2}\right)$, where the smaller σ^2 the faster the decreasing of w_{ij} for increasing values of the Euclidean distance.

It's important to notice that these measures take values in the range of $[0, 1]$ and higher values occur when the data samples are more alike, i.e. when their Euclidean distance is lower. Therefore, problem (1) gives higher priority to the minimization of the terms associated to the higher similarity weights w_{ij} and $\overline{w}_{ij}$: if two points are assigned with a high similarity value, then their labels will be forced to be equal, with a higher priority with respect to couples of points associated to low similarity values.
Moreover, the similarity matrices are made sparse by applying a threshold on the components: in this way we avoid the exploding gradient problem (overflow) because similarity values that are not close enough to 1 are set to 0.

For simplicity, we can rewrite the objective function of problem (1) by exploiting the fact that $\overline{w}_{ij} = \overline{w}_{ji}$:

$$\begin{aligned} f(y) &= \sum_{i=1}^{l} \sum_{j=1}^{u} w_{ij}(y^j - \bar{y}^i)^2 + \frac{1}{2} \sum_{i=1}^{u} \sum_{j=1}^{u} \overline{w}_{ij}(y^i - y^j)^2 \\ &= \sum_{i=1}^{l} \sum_{j=1}^{u} w_{ij}(y^j - \bar{y}^i)^2 + \sum_{i=1}^{u} \sum_{j=i}^{u} \overline{w}_{ij}(y^i - y^j)^2. \end{aligned} \quad (2)$$

[1] https://archive.ics.uci.edu/ml/datasets/HTRU2

Then, the j-th component of the gradient results to be:

$$\nabla_{y^j} f(y) = 2\sum_{i=1}^{l} w_{ij}(y^j - \bar{y}^i) - 2\sum_{i=1}^{u} \overline{w}_{ij}(y^i - y^j)$$
$$= 2\sum_{i=1}^{l} w_{ij}(y^j - \bar{y}^i) + 2\sum_{i=1}^{u} \overline{w}_{ij}(y^j - y^i). \quad (3)$$

3 Methods

The classic Gradient Descent (GD) method calculates each iterate as

$$y_{k+1} = y_k - \alpha_k \nabla f(y_k) \quad (4)$$

with stepsize $\alpha_k > 0$.
The convergence rate of the GD method is dimension-free, but the computation of the full gradient can be very costly when dealing with high-dimensional data.

On the other hand, the idea underlying the Block Coordinate Gradient Descent (BCGD) methods is to split the optimization problem into smaller optimization processes by assuming that the variables are partitioned in b blocks of dimension n_i, for $i = 1, ..., b$, such that $u = n_1 + \cdots + n_b$, where u is the dimension of our problem. In our case: $n_i = 1$ and $b = u$.
In particular, the Cyclic BCGD method updates each iterate only after computing

$$y_i = y_{i-1} - \alpha_i U_i \nabla_i f(y_{i-1}) \quad (5)$$

for each block i, where $\mathbb{I} = [U_1|\dots|U_b]$ is the identity matrix in $\mathbb{R}^{u\times u}$ and $U_i \in \mathbb{R}^{u\times n_i}$.
Instead, the Randomized BCGD method calculates each iterate as

$$y_{k+1} = y_k - \alpha_k U_{i_k} \nabla_{i_k} f(y_k) \quad (6)$$

where block i_k is randomly chosen at each iteration, by sampling from the uniform distribution, i.e. $P(i_k = i) = \frac{1}{u} \quad \forall\, i = 1...u$.

All the methods have been implemented by setting a fixed stepsize, since the complexity of Exact Line Search or Armijo Rule results to be prohibitive in practice: our objective function is neither cheap nor well structured, because it can't be fully rewritten in matrix form.

For what concerns the stopping criterion, all the methods are forced to not exceed the number of maximum iterations and GD and Cyclic BCGD are also forced to stop when the norm of the gradient gets lower than a certain tolerance. This last condition is not applied to Randomized BCGD because we noticed that the quantity $|u \cdot \nabla^2_{i_k} f(y_k)|$ is not enough representative of the full gradient norm. Moreover, for computational complexity reasons, we excluded the stopping criterion based on the value of the loss function.

Since the optimization problem (1) is defined for $y \in \mathbb{R}^u$, we also define the iterates of all the methods as real-valued vectors and the resulting gradient norm at the optimum is very close to 0.
Then, to assess the classification task for $y \in \{-1, 1\}^u$ we discretize the optimum by considering $sign(y)$ and compute again the gradient on this discrete vector. The norm of the new gradient is reasonably higher than the one of the gradient computed on the real-valued optimum. Indeed if the components of y assume only values equal to -1 or 1, then the terms $(y^i - \bar{y}^i)$ and $(y^i - y^j)$ can be only equal to -2, 0 or 2, resulting in a high absolute value of the final sum. On the contrary, if y is continuous, the previous terms assume values in a range of $[-2, 2]$, thus the final sum is reasonably closer to 0.
In particular, a smaller value of the gradient computed on $sign(y)$ always corresponds to a better accuracy, while this is not always the case for the gradient computed on y.

4 Results

4.1 Toy Example

First, we consider a balanced toy example of 400 labeled and 1000 unlabeled datapoints, generated from two Gaussian distributions with same variance but different means, as represented in Figure 1.

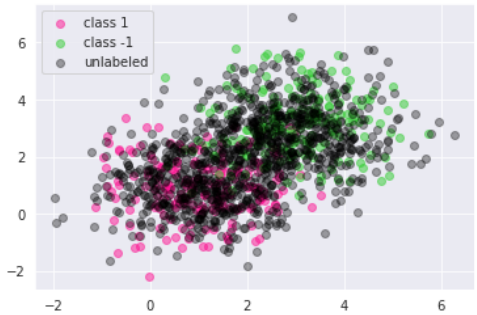

Figure 1. Toy example 2D plot.

The Gaussian Radial Basis function (similarity measure II) with σ^2 equal to the variance of the two Gaussian distributions, empirically shows to work well.

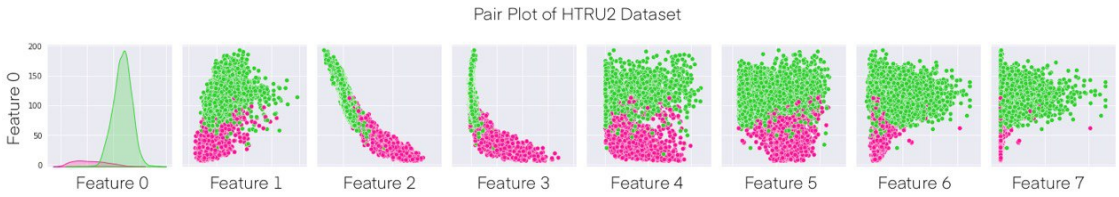

Figure 3. HTRU2 Features Pair Plots
This figure shows just the first row of the Pairs Plot of HTRU2 Dataset, representing the first feature in relation to all the other seven features. We can find the density curves on the diagonal, representing the distribution of a single variable, and the scatter-plots on the upper and lower triangles, showing the relationship between two variables.
Class 1 (Pulsar) is colored in pink and Class -1 (Non-Pulsar) in green.

Figure 2 shows almost the same smooth gradient decreasing for the GD and the Cyclic BCGD, while the Randomized BCGD produces a much more irregular decreasing, since it modifies just one component of y at each iteration.

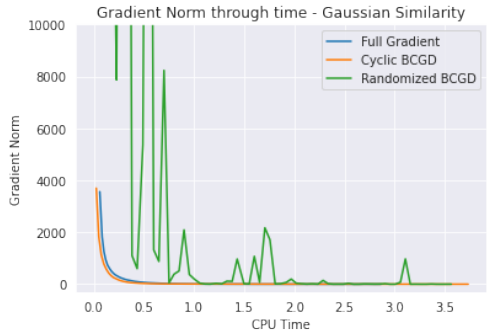

Figure 2. Gradient norm vs CPU time (cropped plot).

As shown in Table 1, the Randomized BCGD is the fastest method because the calculation of the partial derivatives is much cheaper than computing the whole gradient, but, for the same reason, it requires much more iterations to converge: in order to pass through all of the u blocks at least one time, it needs much more then u random attempts.

	GD	Cyclic	Randomized
Gradient on y	0.05012	0.04985	0.82172
Gradient on sign(y)	660.94980	660.92631	660.92631
Iterations	636	626	150000
CPU time	11.85532	15.42657	9.74568
Accuracy	91.90	92.00	92.00

Table 1. Toy example Summary.

Moreover, Cyclic BCGD computes u partial gradients at each iteration, while GD directly computes the full gradient of length u; since the computational complexity of BCGD methods depends on the number of blocks (and here we consider $b = u$) then Cyclic BCGD needs slightly fewer iterations but higher CPU time than GD to converge.
In conclusion, all the three methods are able to reach an accuracy of almost 92%.

4.2 HTRU2 Dataset

Since the HTRU2 dataset is strongly unbalanced, we selected a balanced subset composed by 800 labeled and 1500 unlabeled datapoints.
This dataset is particularly suitable for our application because the features of the datapoints are almost linearly separable, as shown in Figure 3. Therefore Euclidean distances are well representative of the feature similarities. Instead, in case of a dataset in which points from different classes widely overlap, we may have many points that are close to each other but are not similar.

The simple Similarity measure (I) empirically works better in this case.

Results in Table 2 are coherent with the theory: random sampling improves the convergence rate expectation, but results to be highly time-consuming in large-scale problems. Indeed, here we're considering a bigger dataset than the previous toy example.

	GD	Cyclic	Randomized
Gradient on y	0.50122	0.49961	3.13508
Gradient on sign(y)	238.53269	238.32253	238.83703
Iterations	582	578	250000
CPU time	20.23947	26.04186	20.92415
Accuracy	89.40	89.47	88.40

Table 2. HTRU2 Summary.

In Figure 4 we can see the accuracy of the methods through the time needed by each method to reach the minimum. It is interesting to notice that GD and Cyclic BCGD methods reach a good accuracy in very few steps while Randomized BCGD method needs some extra time.

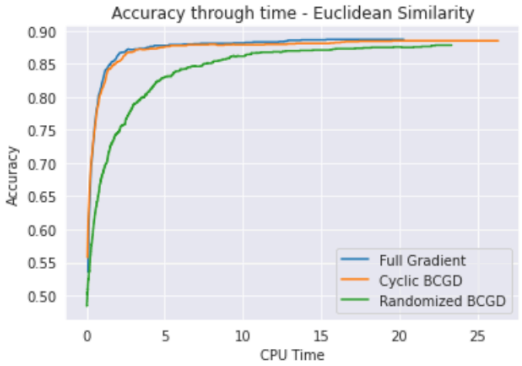

Figure 4. Accuracy vs CPU time.

5 Conclusions

Moving from the toy example to the real-world dataset, we can see that the required CPU time for the three methods to converge increases with respect to the complexity and the amount of data, leading to little drops in accuracy.

In particular, our empirical results confirm the fact that the Cyclic BCGD iteration cost is $\mathcal{O}(b)$ times larger than Randomized BCGD.

While the Randomized BCGD seems the best choice for small-dimensional problems, it is sligthly slower compared to the classic Gradient Descent method in larger-scale problems.

In general, the BCGD methods would perform better when considering bigger blocks, a non-fixed stepsize and a strongly convex function. In our work, each block has dimension 1, thus we are not exploiting any structure and the stepsize is fixed, so Cyclic BCGD takes more or less the same iterations as GD, but more CPU time, to converge. Moreover, Randomized BCGD should improve a lot when considering non-uniform sampling.