Homework 1

Optimization for Data Science (A.Y. 2020/21)

Alvise Dei Rossi (ID: 2004250) - Lorenzo Corrado (ID: 2020623) - Riccardo Vinco (ID: 2005800)

1 Introduction

Semi-supervised learning is a machine learning approach that represents the meeting point between supervised and unsupervised learning. During the analysis, semi-supervised learning uses a small portion of labeled data and a larger portion of unlabeled data. The aim of this work is to correctly assign labels to all observations in our dataset. This is a very important problem nowadays as it is relatively easy to have a large amount of data but it is often much more difficult to have correctly labeled data; it is therefore important to develop criteria that are able to assign the correct labels to the observations within our dataset if they are missing. More specifically, each observation of our dataset represents a point in two dimensions, which can be represented on a cartesian plane. We know the label of only a small portion l of these observations, denoted by $\bar{y}^i \in \{-1, 1\}, \forall i = 1, ..., l$; the other u observations are unlabeled. The basic paradigm for assigning unlabeled observations to one of the two classes is that, having a measure of similarity between observations, observations that have high similarity with each other must belong to the same class.

2 Problem

In order to solve this problem we have a function that we have to optimize:

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

Where w_{ij} represents the measure of similarity between labeled observations and unlabeled observations, while $\bar{w}_{ij}$ represents the measure of similarity between unlabeled observations[1].

2.1 Similarity measures

In order to assign each observation to the correct class we need to define a similarity criterion between the observations. In our work we decided to define the distance between two points using the euclidean distance[2] d. Finally, to obtain a similarity measure we decided to test two different criteria; the first criterion chosen is:

$$w_{ij} = \frac{1}{1 + d(\bar{y}^i, y^j)} \in [0, 1], \qquad \bar{w}_{ij} = \frac{1}{1 + d(y^i, y^j)} \in [0, 1] \tag{2}$$

The second chosen is based on the kernel of a normal distribution:

$$w_{ij} = exp\Big(-\frac{1}{2\sigma^2} d(\bar{y}^i, y^j)\Big) \in [0, 1], \qquad \bar{w}_{ij} = exp\Big(-\frac{1}{2\sigma^2} d(y^i, y^j)\Big) \in [0, 1] \tag{3}$$

In this case, the idea was to assign a similarity between the observations that also took into account the probability that two observations having a certain distance could belong to the same class. This criterion, with respect to the previous one, tends to exponentially increase the similarity values between nearby observations and, on the contrary, tends to exponentially penalize more distant observations. The idea was to elaborate a similarity criterion that would highlight a clearer behavior among the observations. The σ parameter will be chosen according to the different characteristics of the datasets, this allows a certain flexibility of our procedure to different types of data.

[1] Note that $\bar{w}_{ij} = \bar{w}_{ji} \quad \forall i, j = 1, ..., u$.

[2] The euclidean distance is defined as $d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}, \quad \forall p, q$.

2.2 Convexity of the function and further considerations

In order to effectively apply a specific class of optimization algorithms, it is necessary to study the characteristics of the objective function. In particular, the gradient of equation (1) with respect to the single unlabeled observation y^j is:

$$\nabla_{y^j} f(y) = 2\sum_{i=1}^{l} w_{ij}(y^j - \bar{y}^i) + 2\sum_{i=1}^{u} \bar{w}_{ij}(y^j - y^i) \tag{4}$$

$\forall j = 1, ..., u$. To prove the convexity of the function it is possible to consider the partial derivative of the function with respect to the fixed component $j = j^*$ and deriving again we obtain:

$$\frac{\partial^2 f}{\partial (y^{j^*})^2} = 2\sum_{i=1}^{l} w_{ij^*} + 2\sum_{i \neq j*}^{u} \bar{w}_{ij^*} \geq 0, \quad \forall j^* = 1, ..., u$$

Since we proved that the second derivative is non-negative for every single component j we can conclude that $f(y^{j^*})$ is a convex function. Since the function $f(y)$ is obtained as the sum of the single j-th components, we can conclude that $f(y)$ is also a convex function. Furthermore, from the formula in equation (4) it is possible to calculate the components of the hessian matrix, which are:

$$(Hf)_{jj} = \frac{\partial^2 f}{\partial f(y^j)^2} = 2\sum_{i=1}^{l} w_{ij} + 2\sum_{i \neq j}^{u} \bar{w}_{ij}, \ \forall j \qquad (Hf)_{jk} = \frac{\partial^2 f}{\partial f(y^j)\partial f(y^k)} = -2\bar{w}_{jk} = -2\bar{w}_{kj}, \ \forall j \neq k$$

The matrix H is a symmetric matrix and, with the similarity criteria defined in the previous point, its eigenvalues are all greater than zero[3]. Given this we can conclude that the matrix is positive definite and this allows us to rewrite the function present in equation (1) as a quadratic programming problem:

$$\min_{y \in \mathbb{R}^u} \quad f(y) = \frac{1}{2} y^T Q y - c^T y + k \tag{5}$$

Where:

- The vector $y \in \mathbb{R}^u$ is the vector whose components are the variables y^j, so the unlabeled observations;
- The matrix Q is equal to the hessian matrix H;
- The vector $c \in \mathbb{R}^u$ is the vector whose components are $c_j = 2\sum_{i=1}^{l} w_{ij}\bar{y}^i$;
- The constant $k \in \mathbb{R}$ with $k = \sum_{i=1}^{l}\sum_{j=1}^{u} w_{ij}\bar{y}_i^2$.

The gradient can be recalculated via:

$$\nabla f(y) = Qy - c \tag{6}$$

2.3 Optimization algorithms

Finally, in order to optimize the function in equation (5) we will use the algorithms GD, BCGD via cyclic rule and BCGD via randomized rule, which we studied for the optimization of convex functions and of which we will also perform a comparison[4]. The choices that have been made for the implementation of the 3 algorithms are:

- Starting point. For the initialization of the algorithms we have generated the vector to be optimized $y \in \mathbb{R}^u$ sampling from a distribution $U(-1, 1)$;
- Stopping condition. In this case we used different criteria. For the GD algorithm we have imposed that the algorithm terminates the iterations if the difference between the loss calculated in the current iteration compared to the previous one is lower than a certain threshold, if the gradient norm is lower than a certain value and finally we have imposed a maximum number of iterations. For

[3] For the application of our algorithms we will check everytime the truth of this statement.
[4] In the case of algorithms BCGD blocks have size 1.

the BCGD algorithms we have also in this case imposed an exit condition if the update of the loss is less than a certain threshold and if the maximum number of iterations is reached. Moreover, in the case of the BCGD algorithms we have imposed that if the descent direction with respect to the $d_k = 0$ coordinate the algorithm changes the descent coordinate, this helps the BCGD algorithms to increase the speed of convergence;

- Stepsize. Having defined the initial problem in terms of a quadratic programming problem we can use many of the tools seen in theory. After implementing the exact line search, Armijo-rule and fixed stepsize techniques, we decided to use the first criterion, which by far returned the best performance in terms of convergence time for all the algorithms considered. We calculated the stepsize for the GD:

$$\alpha_k = -\frac{\nabla f(x_k)^T d_k}{d_k^T Q d_k}$$

For every iteration $k = 0, 1, ..., n$. Instead, for the BCGD algorithms:

$$\alpha_{i_k} = \frac{1}{Q_{ii}}$$

with $i = 1, ..., b$ blocks and for every iteration $k = 0, 1, ..., n$.

3 Synthetic dataset

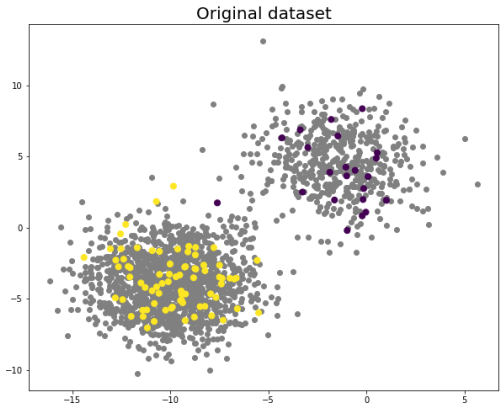

Figure 1. Plot of the randomly generated dataset with unbalanced class

In order to implement the algorithms in a simpler way, it was necessary to generate a synthetic dataset. In the Python library, sklearn, there is the *make_blobs()* function that allows us to automatically generate clusters for classification problems that are sampled from a normal distribution and of which it is possible to specify the mean and standard deviation. The dataset we generated has $|l| = 100$ observations already labeled and $|u| = 1900$ unlabeled observations. Furthermore, to make our procedure more robust we have chosen to distribute the various observations in an unbalanced manner between the two classes. In particular, we initially tested the algorithms with a perfect balance between the two classes and we progressively unbalanced them. At the end of these testing phase we decided to assign the 75% of the observations to one of the two classes and the remaining 25% to the remaining class. Thanks to the gaussian similarity criterion we used, our procedure achieves excellent performance, thus demonstrating its robustness. The fact that the procedure has excellent performance even in the case of unbalanced datasets is very important as often the real datasets suffer from different levels of unbalance.

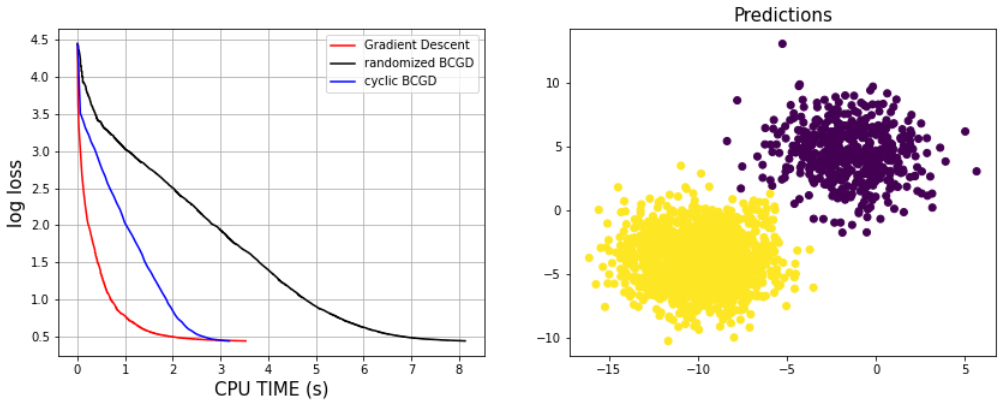

Figure 2. Convergence plot and result on the randomly generated dataset

We can easily see from the previous plot how all the algorithms have reached convergence in a relatively short time. Given the nature of the dataset, the algorithm that has reached convergence in a shorter time is the Gradient Descent. All the algorithms considered have reached a level of accuracy very close to 100%.

4 Real world dataset

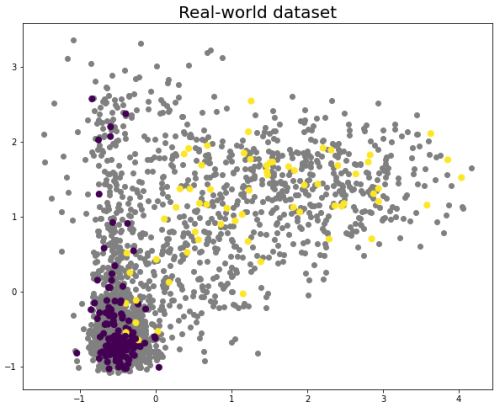

Figure 3. Plot of the real-world dataset

To test the algorithm on real data we have chosen to use the Pulsar Star Detection dataset[5]. The dataset has 12,528 observations but a pre-processing is needed. We removed the NaN values and given the too strong imbalance between the observations in the two classes, a sub-sample from the most frequent class was selected. The final size of the dataset is 3,000 observations on which 8 variables were measured, with a balance between the two classes similar to that of the dataset from the previous point. Since the variables in the dataset were collected on different scales, we decided to standardize it in order to do not create distortion when we calculate the similarity matrices and to accelerate the convergence of the algorithms.

[5]Website for the dataset: https://www.kaggle.com/colearninglounge/predicting-pulsar-starintermediate

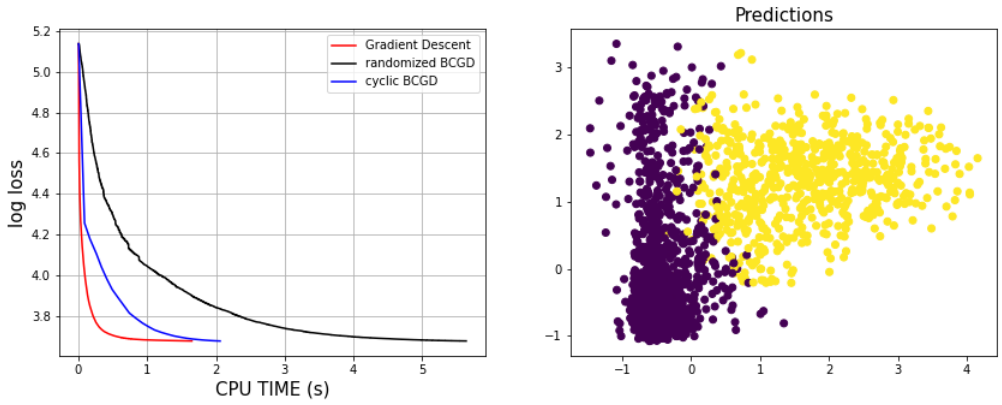

Figure 4. Convergence plot and result on the real-world dataset

Also in this case all the algorithms reached convergence. The performances achieved for each are between 90% and 92% despite the difficulty of the problem due to the fact that the classes are not clearly separable. It is possible to see how the algorithms demonstrate a behavior similar to that seen in the dataset of the previous point, due to the fact that the dimensionality of the problem is comparable.

5 Conclusions

As we can see from the plot in figure (2) and figure (4) our procedure assigns the labels to the various classes in a very satisfactory way even in the case in which the starting dataset is unbalanced and in the case in which the classes are not easily distinguishable. The performances obtained can be attributed to the gaussian similarity measure that we have defined, but for future works different criteria could also be used (polynomial, exponential, etc.) depending on the characteristics of the dataset. It is important to say that we took special care when implementing the algorithms to ensure that the gradient computation and loss function were efficient but there may be further possibilities for improvement. Finally, to study the behavior of the various algorithms as the size of the problem varies, we also decided to test them by varying the size of the problem and comparing them in a similar way to what was seen in the course theory. The results we have obtained are:

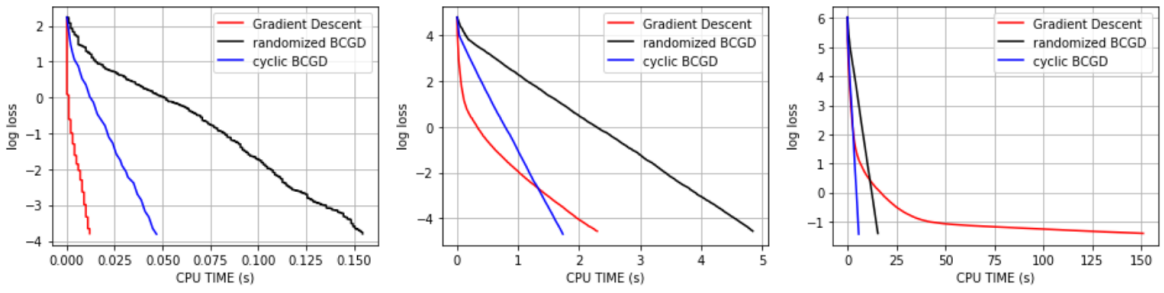

Figure 5. Convergence results increasing the dimensionality of the problem (from left to right)

As we can see from the plot in figure (5), the GD algorithm is the fastest to converge when the dimensionality of the problem is limited, while the BCGD algorithms tend to converge more slowly. By increasing the dimensionality, on the other hand, the criticalities of the GD algorithm are highlighted and also the positive aspects of the BCGD algorithms are highlighted. Consistently with what we saw in theory, the GD algorithm takes a long time to converge since at each iteration it has to calculate the entire gradient which is computationally very expensive, the other two algorithms instead converge much faster as the gradient is calculated and updated only with respect to a single coordinate.