Optimization for Data Science
Homework 1

Analysis of Gradient descent and BCGD methods

Camilo Betancourt Nieto , Tommaso Davì

15 May 2023

1 Introduction

In real-world scenarios it is rather frequent to get data with a lot of unlabeled examples, while it is more uncommon to obtain huge amount of labeled data points. However, the labeled examples can be exploited to classify unlabeled data through the semi-supervised learning process. The goal of this technique is extracting an estimate of the function that correlates the data to their label from a given finite set of training data pairs. In the end, the labels of the originally unclassified examples should be obtained.
In this experiment we randomly generate a dataset of 1,000 points distributed into two clusters, where only one percent of them is labeled as belonging to class 1 or class 2.
A proper measure is introduced in order to quantify the degree of similarity between each couple of points: the closer they are, the higher the weight related to them. It is thus possible to define a convex function -the objective or loss function- that depends on all the labels of the unclassified group of points.
The function is then minimized using different gradient-based algorithms. In particular:

- Gradient Descent,
- Randomized BCGD (with one-dimensional blocks),
- Gauss-Southwell BCGD (with one-dimensional blocks).

For the first method, the descent algorithm is implemented with a fixed step size. On the other hand, for the two BCGD methods, the optimization is performed by using both a fixed step size and the exact line search approach.
These methods are then compared by plotting the accuracy of the algorithms versus the computing time and the number of iterations.
Subsequently, the efficacy of the code is tested on a real dataset containing thousands of credit card transactions, of which a small percentage is fraudulent.

2 Datasets

2.1 Synthetic dataset

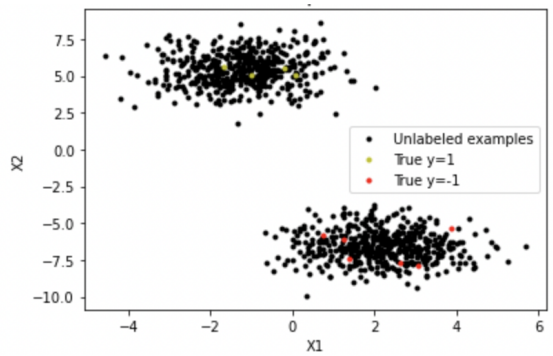
As mentioned in the introduction, we generated a set of 1,000 2D points that constitute two normally-distributed clusters, each with a standard deviation of 1.05. However, only a uniformly distributed random sample of 10 data points (1% of the data) is labeled as belonging to class 1 and class 2, with the "1" and "-1" tag, respectively, as we can see in the graph below.

We can subsequently define the set of labeled and unlabeled data pairs respectively as follows:

$$L = \Big\{ (\bar{x}^i, \bar{y}^i) |\, \bar{x}^i \in \mathbb{R}^2, \bar{y}^i \in \{\pm 1\},\, i = 1, 2, ...10 \Big\}$$

and

$$U = \Big\{ (x^i, y^i) |\, x^i \in \mathbb{R}^2, y^i \in \{\pm 1\},\, i = 1, 2, ...990 \Big\},$$

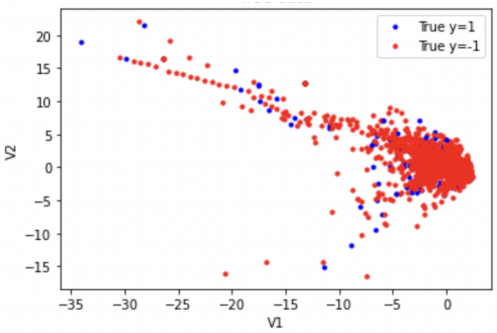


Initial problem with the synthetic dataset

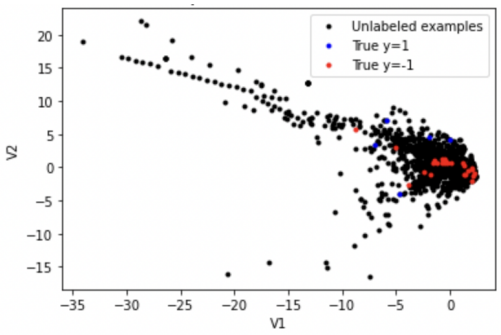
where the values of y^i are initially generated at random.

2.2 Real dataset

In order to test our implementation with real data, we used a dataset related to fraudulent credit card transactions. This dataset is publicly available in Kaggle[1] and shows transactions made by credit cards in two days of September 2013 by European cardholders, in which 492 out of 284,807 transactions were fraudulent, making it a very unbalanced dataset in which the minority class count represents 0.17% of the majority class size.
Since our goal in this report is focused on the optimization problem rather than the classification tasks, we first processed the dataset in order to tackle it as a semi-supervised learning problem that is comparable to the one performed with the synthetic dataset. Firstly, due to the unbalance of the dataset, we used an undersampling strategy so that the minority class (fraudulent transactions) accounts 20% of the majority class (non-fraudulent transactions).
Then, regarding the labeling, we randomly selected (with uniform probability) some points and removed their tags, making them unlabeled examples. After this procedures we ended up with only 30 labeled data points and 2,922 unlabeled data points, which have the same order of magnitude as the synthetic dataset.
It should be mentioned that the dataset that is available in Kaggle had already been processed by means of a PCA transformation, so its features were V1,V2,...V28: the principal components. Because of our optimization approach, we only selected the first two principal components (the ones that account for the most variability within the data points) to characterize our data. Also, this allows us to graphically represent our semi-supervised learning problem:



Real dataset



Real dataset with 90% of the data unlabeled

[1] https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

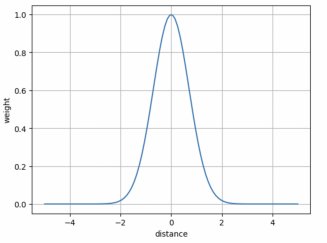
3 Formalization of the problem

3.1 Weights

In order to quantify the relatedness between two points x^i, x^j or x^i, $\bar{x}^j$ a form of similarity measure is necessary. We hence define coefficients as *Gaussian weights* via the function:

$$w_{ij} = e^{\frac{\left\|x^i - x^j\right\|_2^2}{\epsilon^2}} \qquad or \qquad \bar{w}_{ij} = e^{\frac{\left\|x^i - \bar{x}^j\right\|_2^2}{\epsilon^2}} \quad ,$$

where ϵ is set to 1 in our case.
The weight matrices W and $\bar{W}$ that gather all the coefficients for the **2.1** synthetic dataset have thus dimensions of (10×990) and (990×990) respectively.



similarity function

3.2 Loss function

In the context of the semi-supervised learning problems, the loss function is the objective function that we want to minimize. In this case, it is defined as follows:

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

where:

- y^j is the label assigned to the j-th unlabeled point;
- $\bar{y}^i$ is the label of the i -th labeled point;
- w_{ij} is the weight relating the i-th labeled point with the j-th unlabeled point;
- $\bar{w}_{ij}$ is the weight relating the i-th unlabeled point with the j-th unlabeled point.

For this reason $y \in \mathbb{R}^u$ and $\bar{y} \in \mathbb{R}^l$. In particular, $l = 10$ and $u = 900$ for the **2.1** dataset.
Henceforth the problem consists in finding

$$\min_{y \in \mathbb{R}^u} f(y).$$

We can notice that the loss function is formed by two terms, each of them containing two sums. The left term is minimized when unlabeled points close to labeled ones get similar labels, while the right part when similar unlabeled data close to each other get similar labels. The function is quadratic, hence it can be rewritten in vector form

$$f(y) = (y - b)^T W (y - b) + \frac{1}{2} y^T \bar{W} y \quad ,$$

where:

- y is the $1 \times u$ vector of the unlabeled points;
- b is the $1 \times l$ vector of the labeled points;
- W is the $l \times u$ matrix of generic element w_{ij}
- $\bar{W}$ is the $u \times u$ matrix of generic element $\bar{w}_{ij}$

3.3 Gradient and Hessian

The partial derivative of $f(y)$ with respect to the j-th element y^j is

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

Each element of the hessian is defined as follows:

if $j = k \rightarrow \frac{\partial^2 f(y)}{\partial y^{2j}} = \nabla_{y^j y^j} f(y) = 2\Big(\sum_{i=1}^{l} w_{ij} + \sum_{\substack{i=1 \\ i\neq j}}^{u} \bar{w}_{ij}\Big)$

if $j \neq k \rightarrow \frac{\partial^2 f(y)}{\partial y^j y^k} = \nabla_{y^j y^k} f(y) = -2\bar{w}_{jk}$

so the resulting matrix is:

$$H = 2\begin{pmatrix} \sum_{i=1}^{l} w_{i1} + \sum_{i=1}^{u} \bar{w}_{i1} - \bar{w}_{11} & -\bar{w}_{12} & \cdots & -\bar{w}_{1u} \\ -\bar{w}_{21} & \sum_{i=1}^{l} w_{i2} + \sum_{i=1}^{u} \bar{w}_{i2} - \bar{w}_{22} & \cdots & -\bar{w}_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ -\bar{w}_u 1 & -\bar{w}_{u2} & \cdots & \sum_{i=1}^{l} w_{iu} + \sum_{i=1}^{u} \bar{w}_{iu} - \bar{w}_{uu} \end{pmatrix}$$

It is worth noticing that, since the loss function is strictly convex, the Hessian has to be positive definite, so its eigenvalues are all greater than zero. Moreover, it is definite all over the space of the data points, so the loss function has a Lipschitz continuous gradient with constant L equal to its maximum eigenvalue. Since exists a constant $\sigma > 0$ s.t. $H - \sigma I$ is positive definite, the function is strongly convex.

4 Algorithms

In this section we briefly discuss the algorithms, highlighting the performance and the parameters adopted for the problem. The main idea of all gradient based methods is using a Taylor approximation of the objective function at a generic iterate k to compute the best search direction d_k and the best step-size $\hat{\alpha}_k$. In particular, the goal is minimizing $f(y_k + \hat{\alpha}_k d_k) \approx f(y_k) + \hat{\alpha}_k \nabla_{y_k} f(y_k)^T d_k$. It is possible to prove that the optimal descent direction, expressed as a unit vector, is $d_k^* = \frac{-\nabla f(y_k)}{\|\nabla f(y_k)\|}$, while there are different methods to determine an efficient step-size.
In order to end stop the algorithm once it gets to convergence a tolerance on the norm of the gradient, a maximum number of iteration and a maximum computing time are fixed. For the **2.1 problem** these are set on 10^{-2}, 50000, 200 seconds respectively.
Also, for all the algorithms we used a random starting point $f(y_1)$ in which all examples were labeled as -1 or 1 with uniform probability. We used a fixed random seed in order to make the algorithm results comparable.

4.1 Gradient Method

By redefining $\alpha_k = \frac{\hat{\alpha}_k}{\|\nabla f(y_k)\|}$, the Gradient Method's k-th iterate is as follows:

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

In our experiment only the fixed step-size case is considered, so $\alpha_k = \alpha$.
Since f is σ-strongly convex and has Lipschitz continuous gradient with constant $L > 0$, setting $\alpha = \frac{1}{L}$, the algorithm is expected to converge with

$$f(y_{k+1}) - f(y*) \leq \Big(1 - \frac{\sigma}{L}\Big)^k (f(y_1) - f(y^*))$$

and needs a number of iterations of the order $\mathcal{O}\Big(log(\frac{1}{\epsilon})\Big)$, where ϵ is the gap to the solution.
A sub-case can be considered: using a step-size of $\alpha = \frac{2}{\sigma+L}$, the convergence rate is even better than the previous one. However, in our experiment this case was not contemplated.

4.2 BCGD Methods

The BCGD methods divide the variables into b blocks of dimension n_i each, with $i = 1, ..., b$ and at each iteration pick some blocks according to a specific rule. They thus update the variables after computing the gradient only with respect to the variables in those blocks. Two remarks are worthy:

1) in our case all the blocks have dimension one. It means that at each iteration the algorithm computes the partial derivative of the loss function with respect to just one variable.

2) Two different step-sizes are chosen: firstly, a fixed step-size, then the exact one (called exact line search).

In particular the exact step-size for quadratic function is given by

$$\alpha_k = \frac{-\nabla f(y_k)^T d_k}{d_k^T H d_k}, \text{where in our case } d_k = -U_i \frac{\partial f(y_k)}{\partial y_i} \quad ,$$

so it results

$$\alpha_k = -\frac{1}{\frac{\partial^2 f(y_k)}{\partial y_i^2}} = -\frac{1}{H(i,i)} \quad .$$

It is worth noting that, from a computational point of view, the exact line search, which maximizes the reduction at each step, is really cheap given the particular characteristics of our problem. For that reason, we considered this kind of step-size in our implementation.
Depending on the strategies adopted to select the blocks at each iteration, we can divide them into multiple frameworks.

4.2.1 Randomized BCGD method

This algorithm simply uses the random sampling rule to choose the only block to be updated at iteration k. A uniform distribution between the blocks is considered.
In the case of one-dimensional blocks, the method has the k-th iterate as:

$$y_{k+1} = y_k - \alpha_k U_{i_k} \frac{\partial f(y_k)}{\partial y_{i_k}} \quad ,$$

where

- U_{i_k} is the column vector of the identity matrix corresponding to the i-th variable choosen at iteration k;
- $\frac{\partial f(y_k)}{\partial y_{i_k}}$ is the the partial derivative of f computed with respect to the i-th variable.

In the case of a fixed step-size, we set $\alpha_k = \frac{1}{L}$. We expect the algorithm to converge as

$$\mathbb{E}[f(y_{k+1}) - f(y^*)] \leq \Big(1 - \frac{\sigma}{bL}\Big)^k (f(y_1) - f(y^*) \quad .$$

4.2.2 Gauss-Southwell BCGD method

The idea behind this algorithm is picking at each iteration the block i such that $i = \underset{j \in \{1,...,b\}}{\arg\max} |\nabla_{y^j} f(y_k)|$,
i.e. the block of coordinates with the higher gradient norm. It is worth remembering that each block has dimension one, hence the method just picks the index of the higher partial derivative.
For this reason the k-th iterate expression is the same as in the previous method. However, what

really differs in this algorithm is the incremental gradient update at each iteration, which is computationally cheap and takes uses the results of previous steps:

$$\nabla f(y_{k+1}) = \nabla f(y_k) - \alpha_k H_i \nabla_{y^i} f(y_k)$$

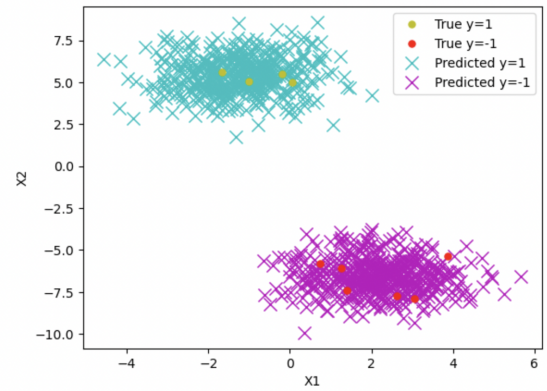
In the case of the fixed step-size $\alpha = \frac{1}{L}$ the method has a convergence rate of

$$f(y_{k+1}) - f(y^*) \leq \left(1 - \frac{\sigma}{bL}\right)^k (f(y_1) - f(y^*) \quad .$$

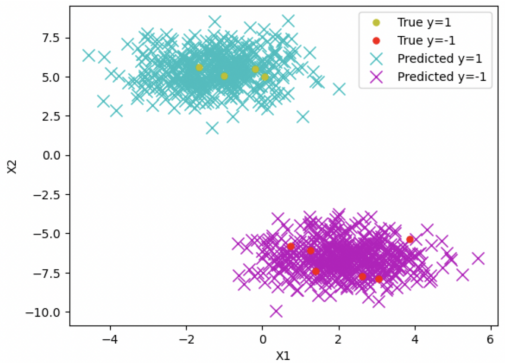
5 Synthetic data: results and discussion

5.1 Classification

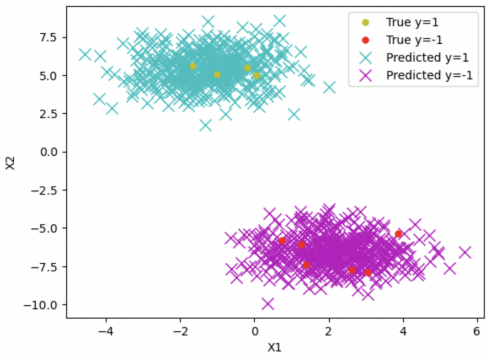
For the sake of plotting the classification results, we considered the labeling as the sign of the examples in the resulting y vector. Here we can observe the classification performance of each algorithm based on how well the labeling was assigned.



Predicted labels via the gradient descent method



Predicted labels via the randomized BCGD method, exact step-size

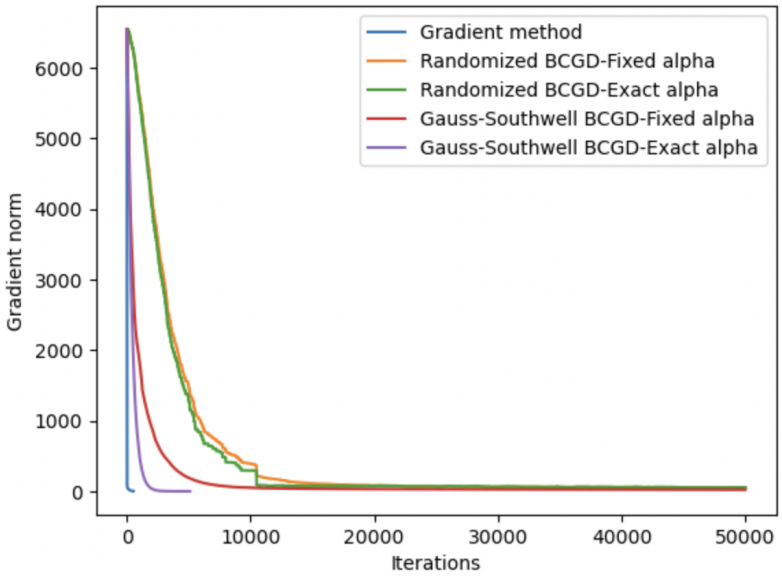


Predicted labels via the Gauss-Southwell BCGD method, exact step-size

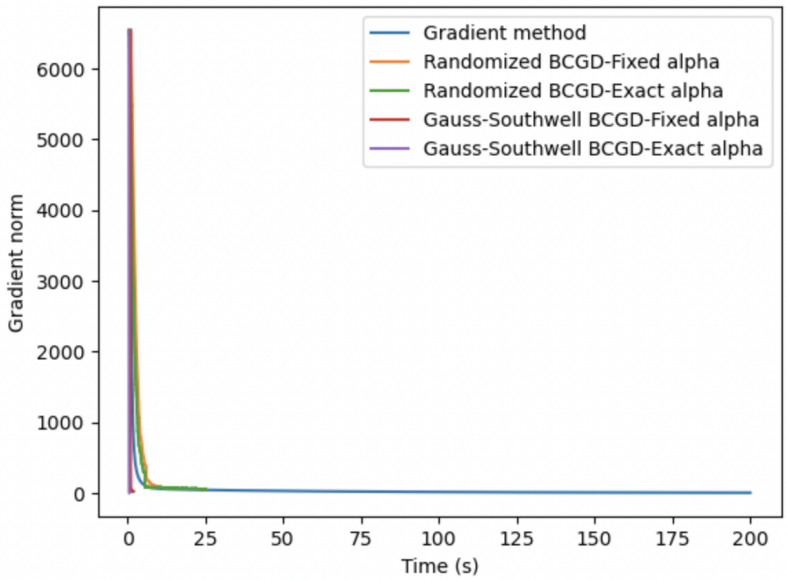
From the plots above we can see that for the three algorithms we obtained a correct classification for all the data points. In the next section we will discuss the performance of the methods from the computational point of view.

5.2 Computational performance

As a means of evaluating the computational performance, since we are dealing with a convex function and the optimality conditions should be reached when the gradient norm is equal to 0, we plotted the accuracy as the reduction in the gradient's norm versus the iterations and the computing time (measured in seconds).



Gradient norm vs number of iterations



Gradient norm vs time

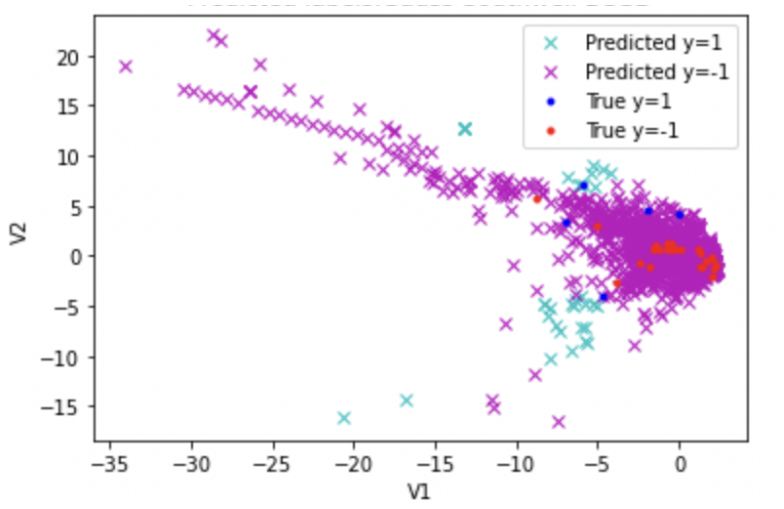
We can see that, as expected, the gradient method has a steeper reduction of the gradient's norm with respect to the amount of iterations, whereas the other methods, especially the randomized BCGD approaches, show smaller decreases per step. However, the Gauss-Southwell BCGD algorithm also showed a steep reduction per iteration, particularly the version that ran with the exact line search.
Nevertheless, when using the gradient method, every update requires that the whole gradient is computed, so each iteration is much more costly in comparison to the other algorithms. This makes the performance with respect to time very different. If we consider the computing time,

the fastest algorithm is the Gauss-Southwell BCGD algorithm and, again, the exact line search approach is the one that behaves most efficiently. Meanwhile the gradient method takes much longer to converge. On the other hand, we can notice that in the case of the randomized BCGD, the exact line search seems to have less impact on the efficiency of the algorithm.
Therefore, because of the difference between the behaviour of the accuracy when we measure it with respect to time or with respect to iterations, we consider that, in addition to an "iterations budget" limitation, implementing a "time budget" restriction is quite important to understand the actual performance of the methods.
In fact, we had made some runs without forcing the algorithms to stop after 200 seconds and what we obtained is that, even if all the methods were heading to convergence or converged, the algorithms kept doing many iterations in which the gradient's norm was reduced, but marginally. After an abrupt decrease at the beginning, the accuracy plots tend to flatten, both with respect to iterations and time.
Also, it is worth mentioning that the randomized BCGD performance can vary from one performance to another, because of its random nature. Also, since we are considering a time limitation to stop for all the methods, the amount of norm reduction may vary in different runs. For example, a faster machine would reach convergence sooner and a slower computer would do the converse. In this sense, mention should be made that, once we introduced the time criterion for stopping, the only method that reached full convergence was the Gauss-Southwell BCGD (it took 0.42 seconds). In scenarios in which a real time limitations exists, this phenomenon should be considered.

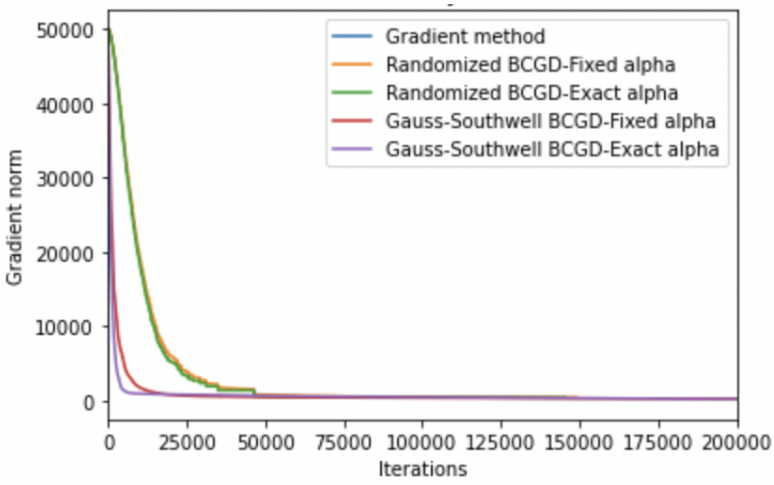
6 Real data: computational performance and classification

Since we focused on the optimization point of view of our problem, we noticed that the difference between the two clusters is harder to distinguish. However, for the sake of completeness, we measured the quality of our prediction using the Area Under the Precision-Recall Curve (AUPRC), due to the original class imbalance ratio. The best result we obtained was using the Gauss-Southwell BCGD method with exact line search and we obtained an AUPRC=0.16. The predicted labels are shown in the plots below.

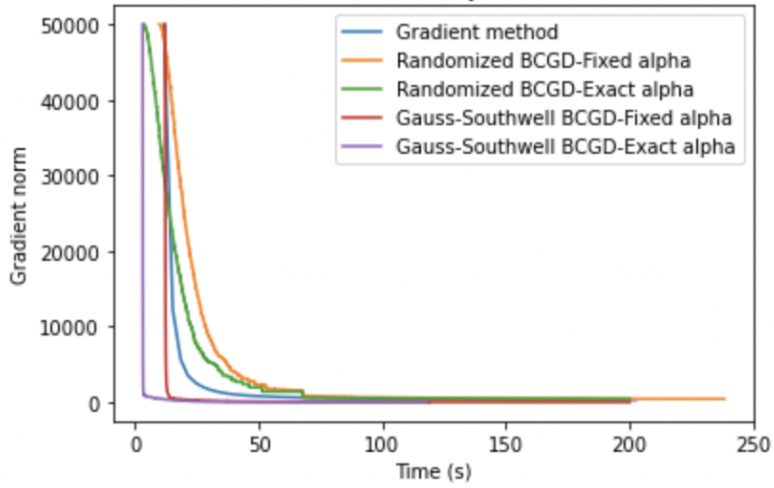
Regarding the computational performance, we can see similar results to the ones that we obtained with the synthetic data in the sense that, again, the Gauss-Southwell BCGD with exact line search seems to be the best performing method. In fact, for some runs we removed the maximum iterations restriction and the algorithm actually reached convergence (i.e. the norm of the gradient was less than our 0.01 tolerance) in 109 seconds after 1,872,568 iterations. Meanwhile the other methods could not converge within the time frame of 200 seconds.
In contrast, since our input was larger this time, the standard gradient method had to perform more calculations in order to complete an update, so it only could carry out 59 iterations in 200 seconds, yielding a gradient norm of around 315, which is much larger than our 0.01 tolerance. Moreover, the algorithm that showed the second best performance was the Gauss-Southwell BCGD with a fixed step-size.
Thus, even if the theoretical framework states a good convergence rate for the gradient method, the cost per iteration can really make a difference and, in this semi-supervised learning context and for the datasets that we considered, the Gauss-Southwell algorithm showed a much better performance in terms of time.



Classification results via Gauss-Southwell BCGD



Gradient norm vs number of iterations



Gradient norm vs time