UNIVERSITÀ DEGLI STUDI DI PADOVA

MASTER DEGREE IN DATA SCIENCE

Gradient Descent and BCGD methods

Optimization for Data Science

2022/2023

Filippo Boldrini
Camilla Colanero
Nicolò Rinaldi
Gabriele Sanguin

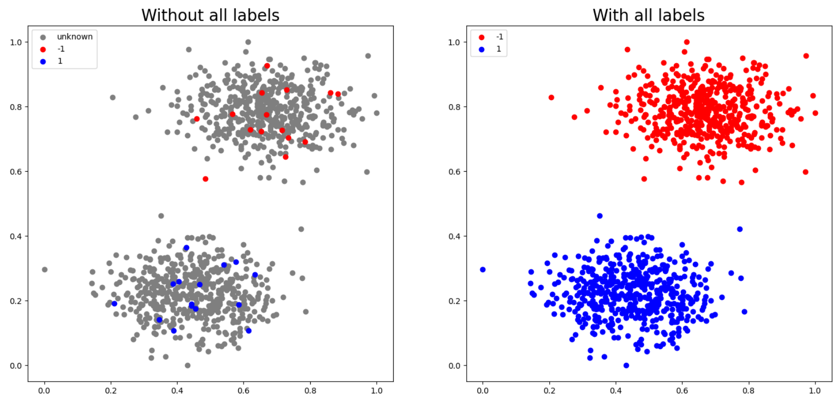
Contents

1 Introduction

In this report we are going to analyze different methods to approach semi-supervised learning as an optimization problem that is, we are going to study which one among the Gradient Descent, the Block Coordinates Gradient Descent with randomized rule and the Block Coordinates Gradient Descent with Gauss-Southwell rule is the most efficient in terms of time and accuracy.
We started creating a synthetic dataset for our case study and we then passed to a real world dataset to verify that our methods did function correctly. Our goal is to minimize a specific loss function and classify some unlabeled points with binary classification given a small set of labeled samples.

2 Dataset generation

For our synthetic dataset we created two clusters with the two labels -1 and 1. Using the standard deviation of the clusters we made sure that the points were separated enough. We only kept a small part of the labels in order to have a lot of unlabeled points that are going to be our targets. We rescaled all the points in the set $[0, 1] \times [0, 1]$ with the minmaxscaler for a better stability.
The following plot represents the dataset after and before the removal of the labels.



3 Formalization of the problem

3.1 Function to minimize

Our goal is to optimize the following problem:

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

where:

- w_{ij} are the weights between the labeled point i and the unlabeled point j
- $\bar{w}_{ij}$ are the weights between the unlabeled point i and the unlabeled point j
- $\bar{y}^i$ are the labeled points
- y^i are the unlabeled points on which we are minimizing the function
- l is the number of labeled points and n is the number of unlabeled points

The first term is related to the penalization of mistakes made while labeling unlabeled points that are close to labeled ones, while the second term is related to the penalization of mistakes made while labeling unlabeled points that are close to each other.

3.2 Gradient

For the calculations in our study we are going to use the following formula for the gradient:

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

We compute the gradient with respect to each y^j that is going to give us the direction of the Gradient Descent at every step.

3.3 Hessian matrix

The Hessian matrix that we are going to use for the computation of α for the fixed step size is the following:

$$M = \begin{bmatrix} 2\sum_{i=1}^{l} w_{i1} + 2\sum_{i=2}^{n} \bar{w}_{i1} & \dots & -2\bar{w}_{1n} \\ \vdots & \ddots & \vdots \\ -2\bar{w}_{n1} & \dots & 2\sum_{i=1}^{l} w_{in} + 2\sum_{i=1}^{n-1} \bar{w}_{in} \end{bmatrix}$$

where:

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

$$\nabla_{y^j y^i} f(y) = -2\bar{w}_{ij}$$

4 Implementation choices

4.1 Weights

The weights play an important role in the minimization of our problem due to the fact that we want to have higher weights when the distance is small since we want to deeply penalize the mistakes in the labeling of very near points.
We used two different similarity functions to compute our weights:

- the Euclidean distance: $\frac{1}{\|x^i - x^j\|_2 + \epsilon}$ where ϵ is the machine precision that in our case is $2.22 \cdot 10^{-16}$.
- the Gaussian kernel: $e^{-\beta\|x^i - x^j\|_2^2}$ where β is a constant.

Both of this functions are high when the distance is small.
For the first similarity function, we rescaled the weights in the interval [0,1] in order to have smaller values of the function f. For the second similarity function we do not need to do that because the weights are already in that interval.
In the study of our synthetic dataset we used the first similarity function since in the tests the convergence was faster than using the second function, while for the real dataset we used the second one.

4.2 Gradient update

In the first method, that is the simple Gradient Descent, we update the whole gradient at each iteration. Instead in the BCGD with randomized rule and GS rule we only update a subset of the gradient.

4.3 Step size

For the step size we chose to try two different ways of calculation: the Armijo rule and the fixed step size.

- The Armijo rule is the following: we fix parameters δ and γ, with $\delta \in (0,1)$ and $\gamma \in (0, \frac{1}{2})$ and we give a starting step size Δ_k. We then try steps

$$\alpha = \delta^m \Delta_k$$

with $m = 0, 1, 2, ...$ until inequality

$$f(x_k + \alpha d_k) \leq f(x_k) + \gamma\alpha\nabla f(x_k)^{\mathrm{T}} d_k$$

is satisfied and set $\alpha_k = \alpha$.

- For the fixed step size we computed the Hessian matrix and we found an estimate of the Lipschitz constant by taking the highest absolute value of the eigenvalues. Our α was then set to $\frac{1}{L}$.

  With our dataset the machine spent approximately 2.72 seconds to compute L, which resulted in L = 2.18. The time used by the machine to calculate the maximum eigenvalue is not negligible. For bigger datasets this implies a great increase in the time of calculation of both the Hessian matrix and the constant L.

We are going to make a comparison between the two choices of α in the next sections.

4.4 Stopping condition

We tried two different stopping conditions for our methods:

- The first one, that is the one we actually used, is that the norm of the gradient is smaller than a certain epsilon.
- The second one is that the difference between the values of the function in the current iterate and in the previous one is smaller than a certain epsilon.

We used the first condition because it performs better than the second one.
We were indeed able to use the first stopping condition even for the BCGD with random rule and with GS rule after a caching strategy (we stored the whole gradient and its norm and at each iteration we only updated the entry that had been changed during that iteration).
To be sure to obtain a good result we implemented the patience, i.e we set a number of consequent iterations the stopping condition should be true before actually stopping the method. For the different methods we used different values of patience: we chose the number of samples over the number of blocks for both BCGD methods and a priori chosen number for the Gradient Descent method.

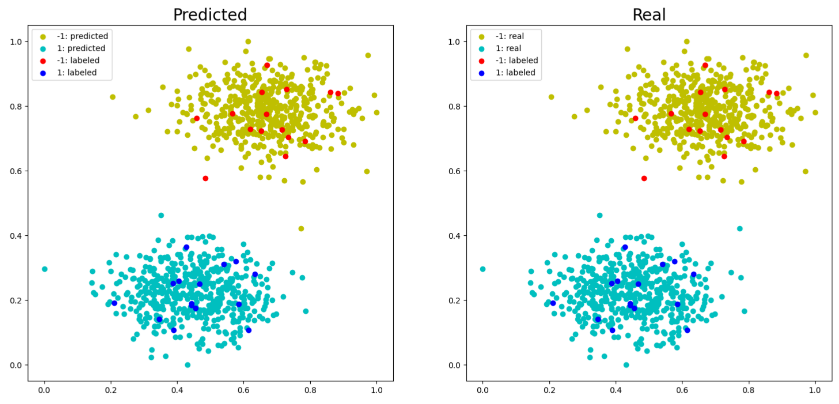
5 Algorithms

5.1 Gradient Descent

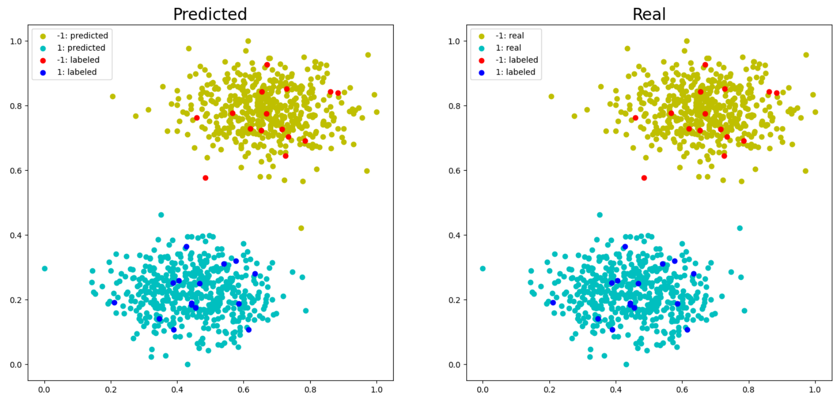
We implemented the Gradient Descent, which step is:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

with both the Armijo rule and the fixed step size. As already mentioned, in the Gradient Descent method we update all the entries of the gradient.
In the following plots we show the results of the labeling through the Gradient Descent with the Armijo rule:



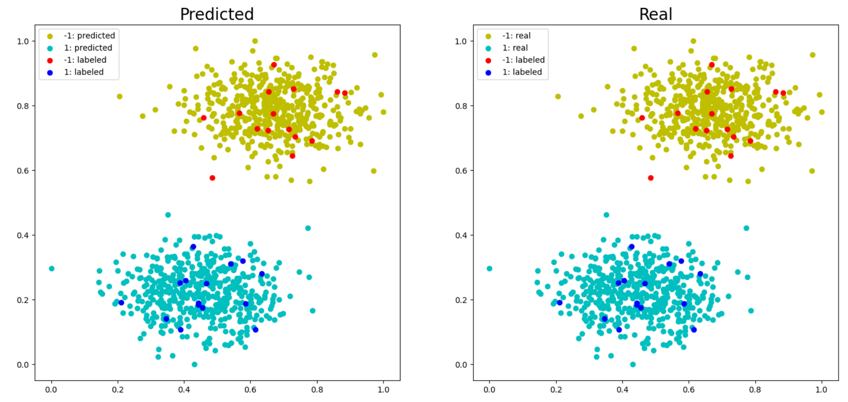
We notice that the method misclassified one point. Even using the fixed step size to compute α we get one misclassified point.



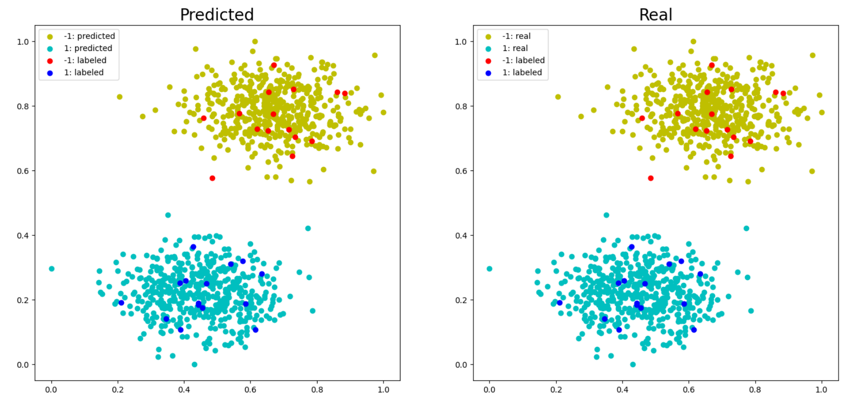
5.2 BCGD with random rule

For BCGD with random rule (and also for BCGD with GS rule) we used blocks of magnitude one. Nevertheless, the code that we implemented makes the generalization to greater magnitude possible.

We implemented the BCGD with random rule both with the Armijo rule and with the fixed step size.
The following plots represent the labels predicted with this method with the Armijo rule against the real ones.



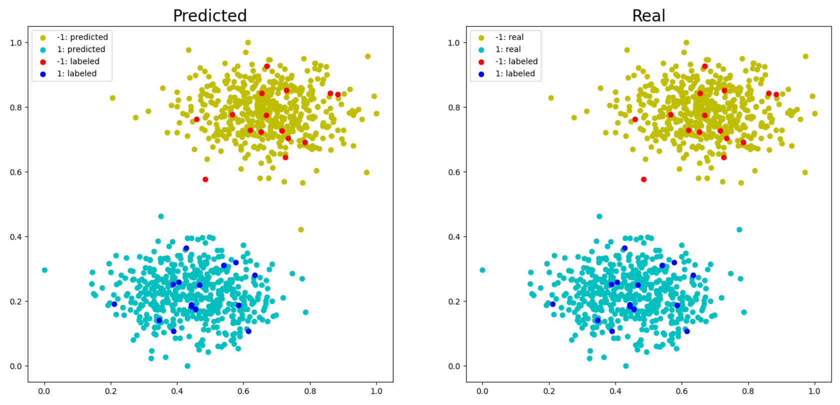
We can see that this method with the Armijo rule did not misclassify any point. Using the fixed step size, we get the same result, that is zero misclassified points.



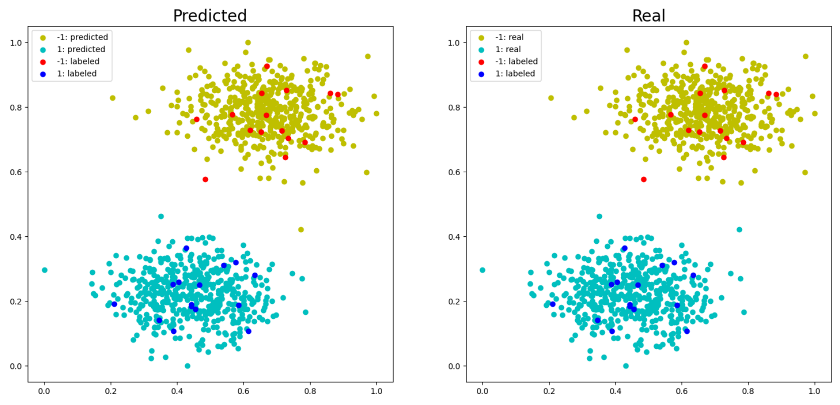
5.3 BCGD with Gauss-Southwell rule

We also implemented the BCGD with Gauss-Southwell rule with the two different ways to compute alpha. In this method, to choose the index to update,

we compute the maximum among the entries of the gradient that we calculated before.
In the following plots we represent the result of the labeling through this method with the Armijo rule.

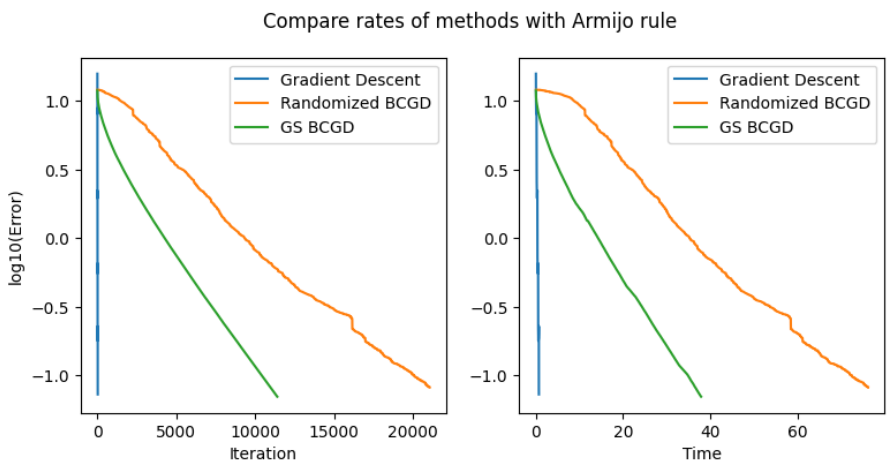


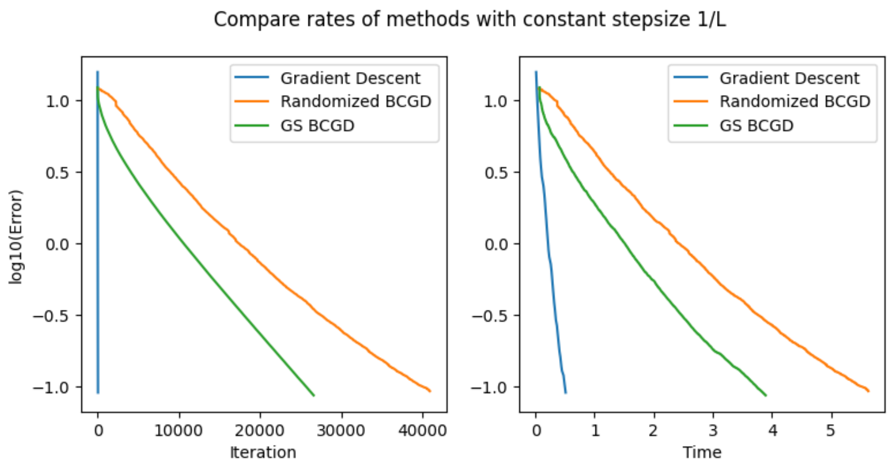
There is one misclassified point.
Using the fixed step size, we get the same result, that is one misclassified point.



6 Comparisons

We now compare the three methods by the number of iterations and the time they need to converge. The first couple of plots refers to the methods with Armijo rule while the second refers to the same methods with fixed step size.





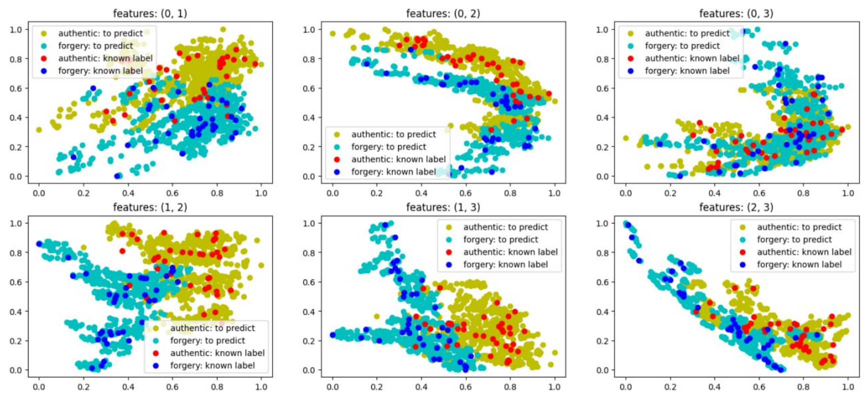
As the plots show us, the methods implemented with the fixed step size are much faster than the ones with the Armijo rule and they keep a very good accuracy.
We notice that the Gradient Descent method is much more efficient than the other two methods both in number of iterations and in time: this may be due to the fact that we use for the two BCGD methods blocks of magnitude one.

7 Real dataset

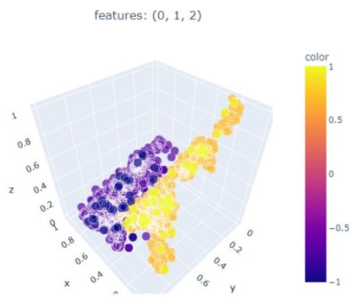
The real dataset that we chose for our study is the Bank Note Authentication UCI ML Repository dataset. Data were extracted from images that were taken from genuine and forged banknote-like specimens. Wavelet Transform tools were used to extract features from images. Our dataset has four features:

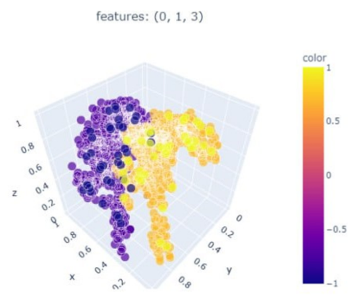
- variance: variance of Wavelet Transformed image (continuous)
- skewness: skewness of Wavelet Transformed image (continuous)
- kurtosis: kurtosis of Wavelet Transformed image (continuous)
- entropy: entropy of image (continuous)

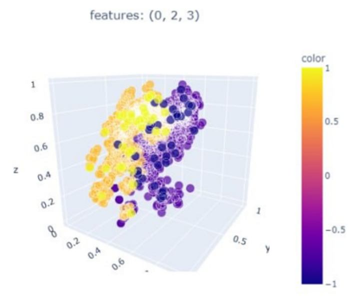
The last column, class, represents our target column, where 0 stands for authentic banknotes and 1 is for forged banknotes. We changed the label 0 to -1 to be coherent with the problem we defined for the synthetic dataset.
This is an unbalanced dataset with 762 samples labeled with -1 and 610 samples labeled with 1.
We rescaled each feature in the set [0,1] and we only kept a few of the label of the target column.
In the following plots we show, for each pair of features, which are the points that we kept labeled and to which of the two classes the points belong to.

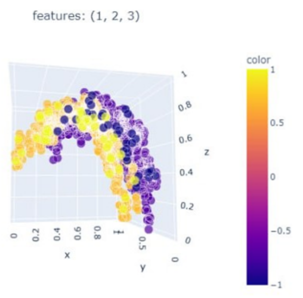


For a better understanding of the situation, we also plotted the 3d version of the points, where each plot represents the points viewed in triplets of features.









As we can see from the plots the points are divided in two clusters very close to each other: there are in fact areas where the two clusters blend.

As for the synthetic dataset, we implemented the three methods. From the previous study we changed some of the parameters.

7.1 Changes

7.1.1 Weights

As already mentioned, for the implementation of the methods with the real dataset we used the Gaussian kernel as a similarity function.
We made this choice because, since the points are very close to each other, with the Euclidean distance there were too many high weights and therefore the labels of the points were all assigned to the larger class. What we actually wanted was to attribute to fewer points high weights in order to have a smaller set with weights significantly different from zero.

7.1.2 Step size

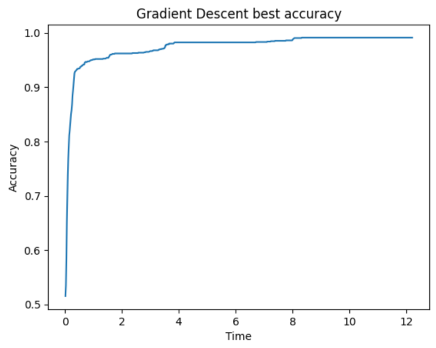
For the step size we decided to use only the fixed step size that we described before (computing $\frac{1}{L}$) since with the Armijo rule the time of convergence of the methods was high because of the oracle call in the implementation of the rule.

7.1.3 Stopping rule

We noticed that with a small epsilon in the stopping condition that we implemented relative to the norm of the gradient, the accuracy, towards the end, did not really improve for several iterations. We therefore added a stopping rule relative to the accuracy, also controlled by a patience, to stop earlier the algorithm of Gradient Descent.

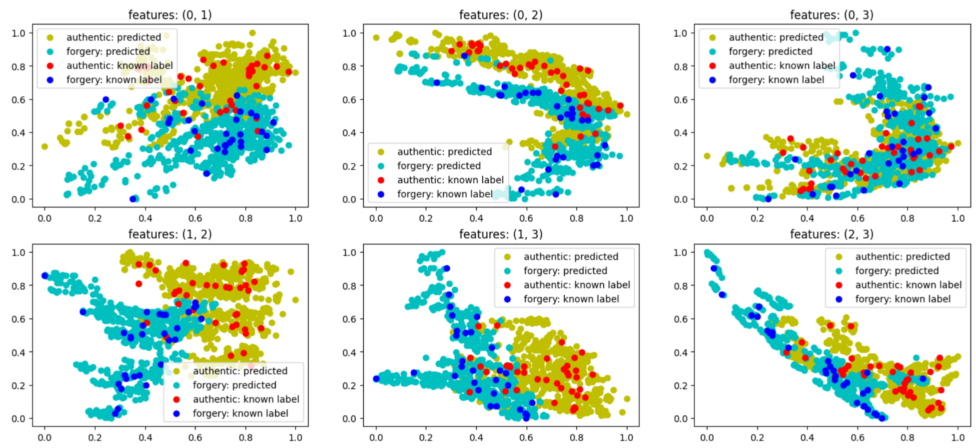
7.2 Best result

With some trials to set the hyperparameters, we were able to achieve only 12 misclassified points, having thus an accuracy of 99.12%. We reached this result with 751 iterations of the method of Gradient Descent (with $\beta = 200$ and patience $= 250$).

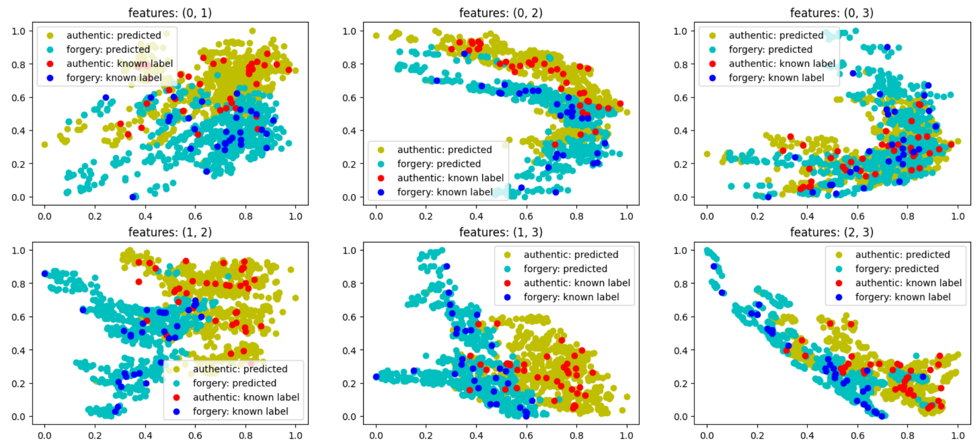


7.3 Test on the three algorithms

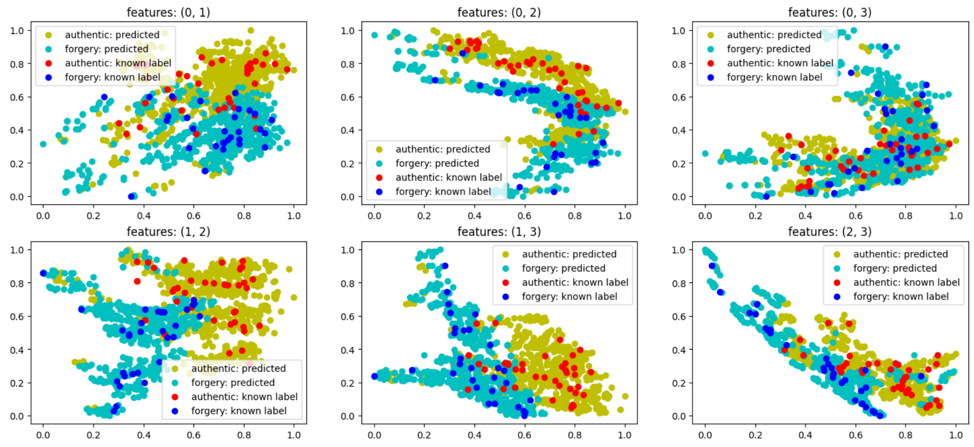
For a faster convergence, we decided to lower the patience and β parameters, even though in this way we lost a little bit of accuracy. In this way we reduced a lot the number of iterations (we achieved 101 iterations with $\beta = 100$ and patience $= 10$), obtaining 23 misclassified points and an accuracy of 98.32% using the Gradient Descent method.



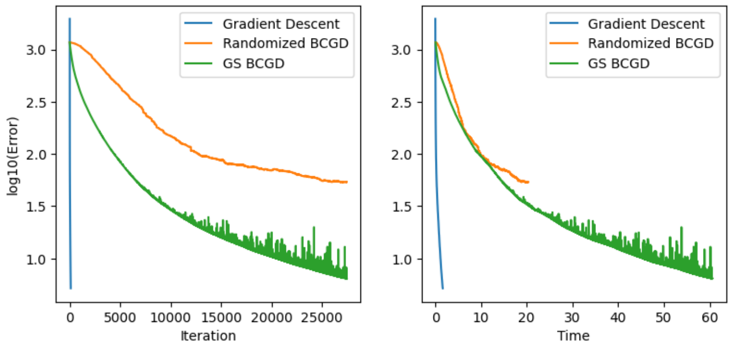
We used the same parameters (except for the patience related to the accuracy, that we set for both BCGD methods to $2 \cdot n_{samples}$) even for the other two methods and we obtained the following results.
For the BCGD with Gauss-Southwell rule we got 29 misclassified data, reaching therefore an accuracy of 97.88%, with 27441 iterations.



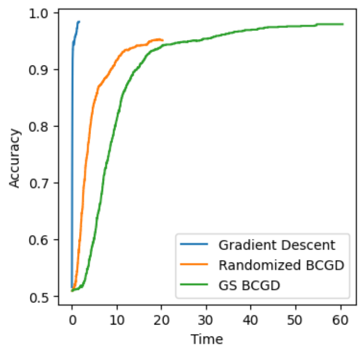
With the same parameters, with the BCGD with random rule we got a worse result: we obtained 68 misclassified points, that is an accuracy of 95.04% with 27441 iterations. This is probably due to the fact that to obtain the same accuracy we should heavily increase the patience because this method takes more time to converge.



In the following plots we show the behaviour of the norm of the gradient (that was our error) for each method, plotting the logarithm of the error against the iterations and the time.



As we can see, the behaviour for the BCGD with Gauss-Southwell rule is very oscillatory. This is probably due to the fact that we are minimizing the gradient in different directions, therefore when we change the entry of the gradient we are updating, the norm of such part of the gradient increases a little with respect to the direction we already minimized.
Nonetheless, we obtain a very good accuracy with each method.

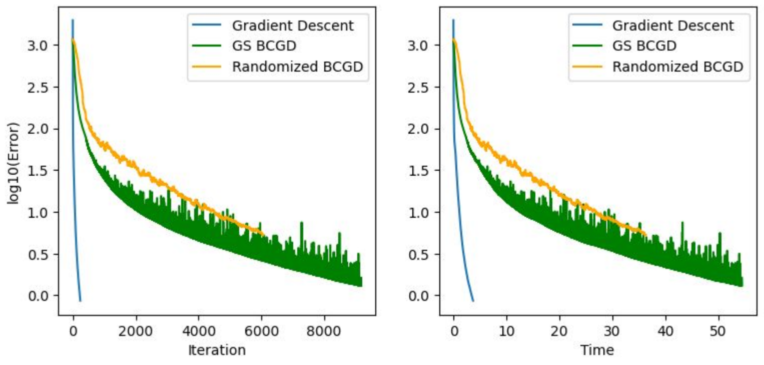


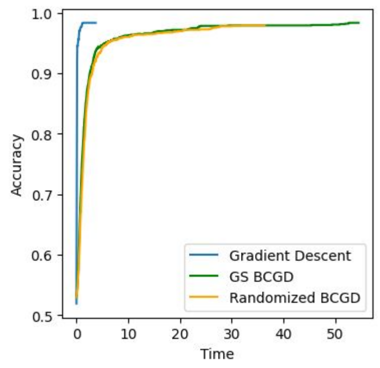
7.4 Increase in blocks dimension

Since the strength of the BCGD methods is the fact that they use blocks to update the gradient, we wanted to try to implement these methods with blocks of bigger magnitude.
We tried implementing the two methods with blocks of magnitude 22 and with both the methods to calculate the step size α.

7.4.1 Fixed step size

With the BCGD with random rule we get 29 misclassified points, i.e. an accuracy of 97.88% with 6067 iterations.
Instead, with the BCGD with Gauss-Southwell rule we get 23 misclassified points, that is an accuracy of 98.32% with 9178 iterations.

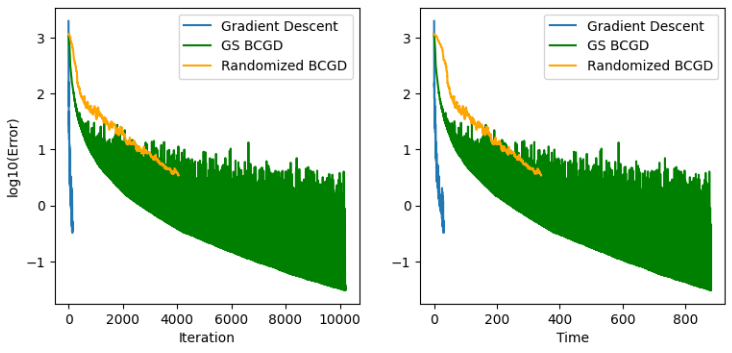


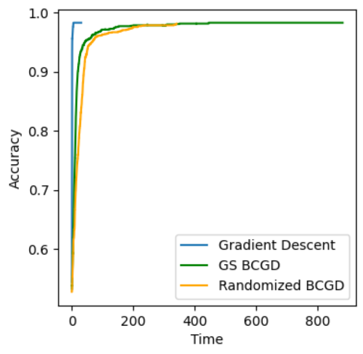


The BCGD with Gauss-Southwell rule, though, ran for more time since its gradient at each iteration changed a lot as we can see from the error plot. This resulted in a higher variability of the f function evaluated at each iteration.

7.4.2 Armijo rule

With the BCGD with random rule we get 25 misclassified points, i.e. an accuracy of 98.18% with 4055 iterations.
Instead, with the BCGD with Gauss-Southwell rule we get 23 misclassified points, that is an accuracy of 98.32% with 10210 iterations.





8 Conclusions

In our studies we end up having much better time and accuracy with the classic Gradient Descent method, since we do not have huge datasets. With increased block dimension we have less iterations with almost the same CPU-time, nonetheless we reached a better order of magnitude of the error.