Semi-supervised clustering with gradient-based methods

Pavel Ianko
pavel.ianko@studenti.unipd.it

Chiara Cavalagli
chiara.cavalagli@studenti.unipd.it

1. Introduction

This paper leverages gradient-based optimization methods to handle a problem of semi-supervised clustering.

The goal is to compare performance and convergence of the following optimization algorithms:

- Gradient Descend (GD),
- Block-coordinate Gradient Descend (BCGD) with cyclic block strategy,
- BCGD with randomly permuted coordinate blocks,
- BCGD with random sampling/randomized.

The algorithms are initially tested on a large scale balanced clustering problem consisting of 5 000 randomly generated points in a *2D* space, with two classes and only 3% of labeled data.

After the comparison on the synthesised dataset, the models are applied to clustering types of wine and to a clustering of pulsar neutron stars.

As a result, concerning the artificial data, all methods, except for the randomized BCGD, achieve 99.9% labelling accuracy after the first iteration. BCGD methods with cyclic and permutation strategies were characterized with slightly steeper decrease of the loss function over iterations.

As opposed to mentioned models, BCGD method with randomized block strategy achieved worst results on both generated and real data. This method took maximum CPU time (9.46 seconds against an average of 2.4) and number of iterations (300 versus 27), despite the lower accuracy on wine classification (88% agaist 90% for other algorithms) and the pulsar classification (49% against 89% for other algorithms).

Visualizations of algorithms' mistakes show, that all models, except for randomized BCGD, are prone to mislabel the data instances at the intercluster interface.

2. Synthesised Dataset

The artificial dataset is randomly generated and represented into two clusters of 5 000 points in total. Further, 3% of data was assigned classes $\{0, 1\}$, while the rest of the dataset is kept unlabeled (Fig. 1).

We then distinguish the following groups of data:

- l labeled examples of the format

$$(\overline{x}^i, \overline{y}^i) \text{ for every } i = 1, \dots, l;$$

  where, in this case, l stands for 300;

- U unlabeled examples of the format

$$x^j \text{ for every } j = 1, \dots, U;$$

  where, in this case, U stands for $5\ 000 - 300 = 4,700$.

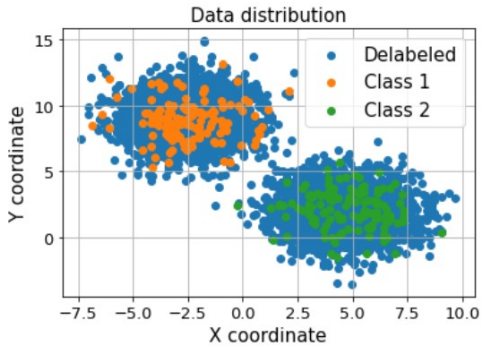

Figure 1. Distribution of the data. 4,700 blue points correspond to unlabeled instances, while the other objects are labeled

3. Task statement

The task is to predict the unknown labels y^j. Thus, the gradient based optimization methods are leveraged, to find class assignments for unlabeled instances, that yield minimum to the *loss function*.

The minimization problem can be formalized in the following way:

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

The search for optimal assignments is performed in the space R^U, as the equation shows. One crucial aspect is the

choice of the *weights* w_{ij}. The approach will be based on concept of "similarity distance": the more the features are close to each other in the space, the more likely is that they will belong to the same class. In other words,

$$\text{similar features} \equiv \text{similar labels.}$$

In this project, the chosen similarity measure between two vectors x, y is the following:

$$\text{similarity}(x,y) = \frac{1}{\|x-y\|_2 + \epsilon}, \text{ where } \epsilon = 0,001.$$

The term ϵ is written in order to obtain a real-valued matrix of weights (this refers to the case where the vectors x and y are equal).

4. Algorithms

Here we recall the general scheme of a gradient-based minimization method:

Choose an initial approximation point $Y \in \mathbb{R}^U$
for $k = 1, \ldots, k_{max}$ **do**
  Stopping Conditions
  Compute Descent Direction d_k
  Compute Stepsize α_k
  Update Rule: $Y_{k+1} = Y_k - \alpha_k d_k$
end for

In this paper, every algorithm has the same stopping condition, that will be explained later (see Section Stopping Conditions), a fixed stepsize α_k. All considered models belong to the family of I-order optimization methods, as they use the first derivative in the updating rule as information on the objective function.

4.1. Gradient Descent

The Gradient Descent (GD) calculates the descent direction as follows:

$$d_k = -\nabla f(x_k), \tag{1}$$

where $f(\cdot)$ is the function to minimize This, a function, subject to minimization in this study, is the clustering loss.

4.2. BCGD and its variants

A general idea of BCGD methods is the same of the GD: the descent direction is the opposite of the gradient, but the main feature of this family of algorithms refers to its usage that is in high dimensional spaces.

Now, with high dimensionality of a parameters space, evaluation of the whole gradient each iteration (as with GD method) is computationally expensive. To fasten the algorithms, in this strategy we partition the input into b blocks of coordinates, each of them of dimension n_i such that:

$$n = \sum_{i=1}^{b} n_i,$$

where n is the total number of coordinates. In order to update a block, it's necessary to define b matrices $U_i \in \mathbb{R}^{n \times n_i}$ such that each block $x^{(i)}$ can be obtained through matrix multiplication:

$$x^{(i)} = U_i^T x.$$

Thus, instead of calculating the whole gradient vector (1), BCGD methods update it block by block:

$$\nabla_i f(x) = U_i^T \nabla f(x).$$

In this paper, all blocks are of one dimensional ($n_i = 1 \; \forall \; i = 1, \ldots, n$), so from now on we will put $n \equiv U$. As a consequence, each matrix U_i will be a column vector and so for each block there will be an update for a single component. The several BCGD methods implemented differs on block selection strategy at each iteration, that are:

- BCGD with *cyclic strategy*: for every iteration, one coordinate at a time is updated sequentially;
- BCGD with *permuted strategy*: for every iteration, one coordinate at a time is updated but with a random order;
- BCGD with *randomized strategy* or *sampling strategy*: for every iteration, a single coordinate is updated and its value it's chosen randomly according to uniform probability distribution.

Moreover, the general scheme takes the stepsize α_k fixed.

5. Algorithms implementation

The first step of implementation is to choose the initial approximation for the unlabeled data. For the classes 0 and 1, a value of 0.5 was chosen as an initial approximation for an unlabeled data instances.

5.1. Stopping Conditions

To save temporal and computational resources, we set several stopping conditions for the algorithms, that are:

- Reaching the limit of iterations (k_{max});
- Achieving the target value of loss function;
- Reaching the plateau of loss curve. That is, when $\delta_i/\delta_{i-1} < 5\%$, where $\delta_i = Loss_i - Loss_{i-1}$.

The above values are specified for each experiment scheme, depending on the size of data. To raise computational efficiency, a predefined pairwise similarity matrix was calculated for the whole dataset. Each iteration, the loss values, as well as required CPU time were saved for algorithms' comparison.

5.2. Experiment scheme

As mentioned before, for all studied algorithms, the values of iterations limit, target loss value and learning rate were fixed. Thus, for a set of 5 000 points, the iterations limit was fixed to 30 while stopping loss target and learning rates were equal to 173500 and 0.0001 respectively. When a new delta of loss function is no more than 10% of the previous delta, the learning was stopped (plateau condition). Before running the experiment, these values were experimentally verified to allow for algorithms convergence and achieving high classification accuracy.

6. Results and discussion

6.1. GD, BCGD Cyclic and BCGD permuted

Figures 2 and 3 illustrate the loss decrease versus iterations.

If a function, subject to optimization, meets the criteria for Lipschitz continuity, but not sigma strong convexity, GD and BCGD methods guarantee sublinear convergence rate. However, in terms of computational resources, evaluating whole gradient is demanding, especially in high dimensional cases like this one. It can also be difficult to meet all the possible assumptions in order to obtain a good convergence.

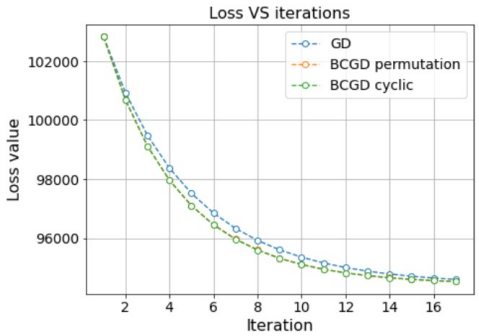

Figure 2. Loss function against number of iterations for gradient-based methods (GD, BCGD with cyclic traversal, BCGD with random permutations)

Thus, theoretical convergence of these methods has to be complemented with computational demands. It's then shown in the plots on figures 4 and 5 that despite sublinear convergence rate in terms of iterations, evaluation of the whole gradient at each iteration inhibits GD algorithm in terms of CPU time.

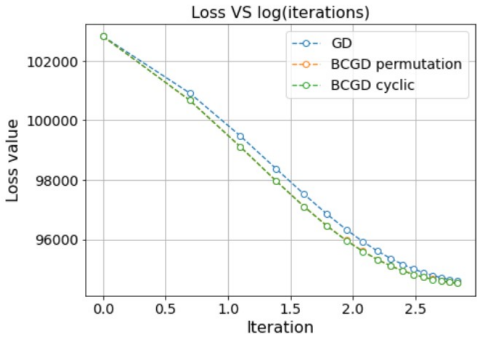

Figure 3. Loss function against logarithm of iterations for gradient-based methods (GD, BCGD with cyclic traversal, BCGD with random permutations)

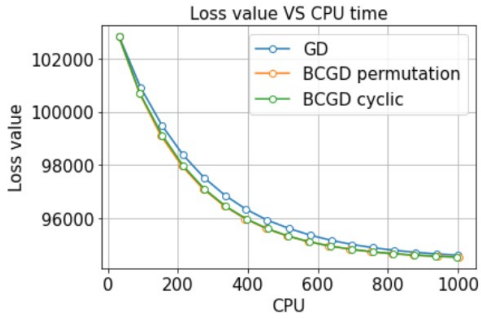

Figure 4. Loss function against CPU time for gradient-based methods (GD, BCGD with cyclic traversal, BCGD with random permutations)

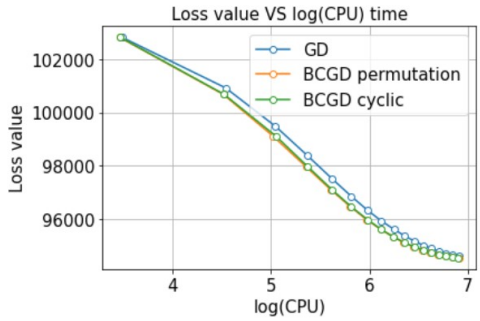

Figure 5. Loss function against logarithm of CPU time for gradient-based methods (GD, BCGD with cyclic traversal, BCGD with random permutations)

Also, all studied algorithms, except for Randomized BCGD, yield maximum achieved accuracy immediately after the first iteration (Fig. 6). Because of picking one coordinate block at a time, randomized gradient descend method gradually increases accuracy each iteration.

In summary, a bar plot on figure 10 compares algorithms overall performance on the toy dataset of 5 000 points with 3 percent of labeled data.

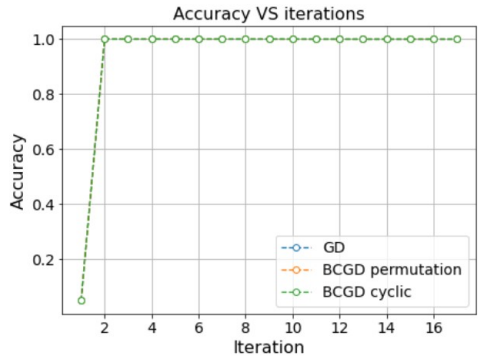

Figure 6. The evolving accuracy across iterations, for gradient-based methods (GD, BCGD with cyclic traversal, BCGD with random permutations)

6.2. Randomized BCGD

Here, we introduce the results of randomized block coordinate gradient descend method. Unlike previous algorithms, randomized BCGD method takes one block at each iteration, where one block has dimension equal to 1 so it means taking a single component. This leads to a remarkable difference in convergence and performance.

For example, figures 7 and 8 demonstrate consistent loss decrease, therefore picking one block each iteration still allows for decreasing loss value and finding optimal solution.

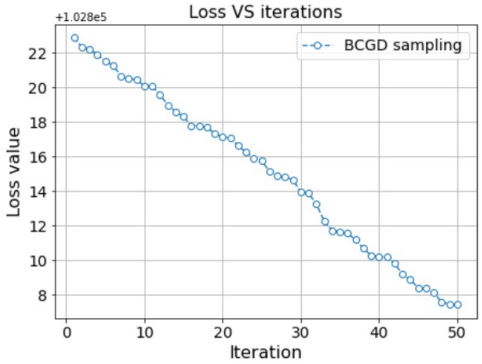

Figure 7. Decrease of loss function for randomized BCGD method, with respect to each iteration

However, the accuracy growth is much less promising (9). For example, as opposed to the optimization methods, which take advantage of full gradient each iteration 6, selecting one coordinate at a time results in a gradual growth of labelling accuracy.

Therefore, randomized BCGD method requires more iterations than the rest of the methods. Figure 10 demonstrates – in the experiment with 5 000 data instances, after 50 iterations, randomized BCGD method only reached 5% labelling accuracy, as opposed to other methods, which achieved 99.9% accuracy after first

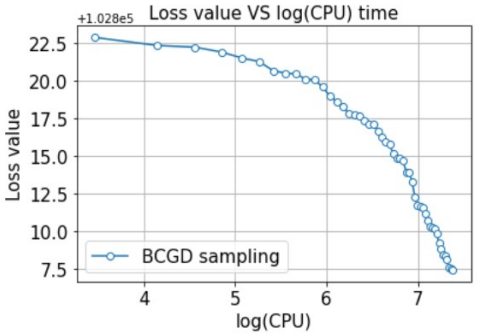

Figure 8. Decrease of loss function for randomized BCGD method, with respect to logarithm of CPU time

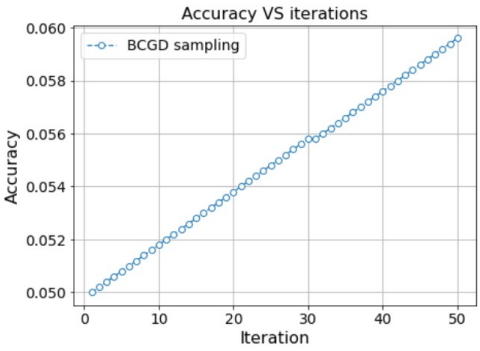

Figure 9. Accuracy versus iterations for randomized BCGD method

iteration (Fig. 6), required 40% less CPU time and only 17 iterations (almost 65% less).

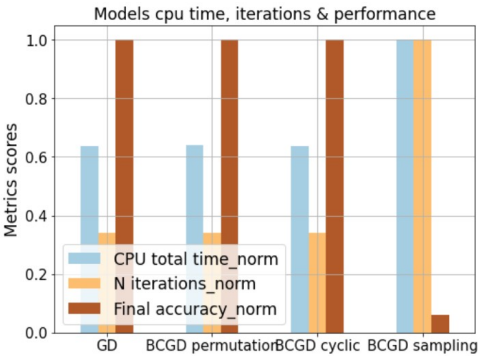

Figure 10. Normalized accuracy, number of iterations and total CPU time for four gradient-based methods (GD, BCGD with cyclic traversal, BCGD with random permutations and randomized sampling)

7. Real Dataset

7.1. Wine Dataset (not large scale)

In this part, we test the algorithms on real data. We start with a dataset consisting of 178 samples of Italian wine.

The chosen features are alcohol saturation and presence of od280/od315 compounds 11.

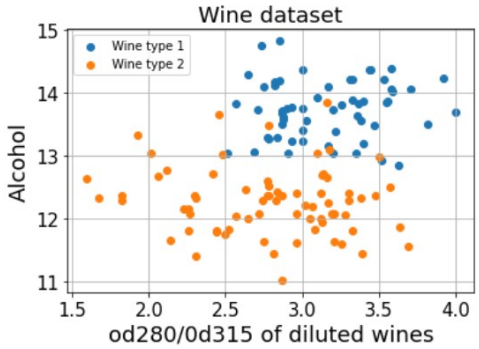

Figure 11. Distribution of classes for wine dataset. The chosen features are alcohol saturation, and content of 0d280/od315 components

The experiment scheme involves following steps:

- Delabelling 95% of the dataset (Fig. 12);
- Running each optimization method with best-fit learning rate, with limit of 30 iterations;
- Stopping condition if $\delta_i/\delta_{i-1} < 1\%$;
- Comparing achieved accuracies and analyzing mistakes.

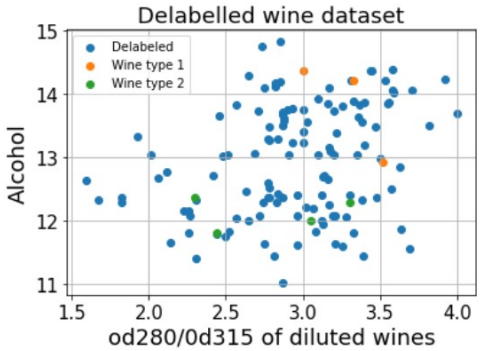

Figure 12. Distribution of classes for wine dataset. 95% of data was delabelled.

Thus, figure 13 summarizes the performance of algorithms, for wine dataset. Except for randomized BCGD with 88% accuracy, all algorithms achieved 90% of correct predictions. In addition to worse performance, randomized BCGD method required ten times more iterations (300 against 27) and computational time (9.46 seconds against 1.21, 3.13 and 2.4 for GD, BCGD cyclic and BCGD with permutations).

An interesting example could be visualizing the incorrect predictions. All methods, except for the randomized BCGD, have the same mistake pattern, as

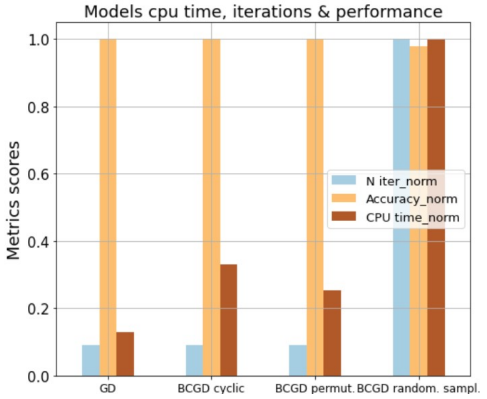

Figure 13. Summarized metrics of optimization algorithms for wine dataset

shown on figure 14. Virtually all mistaken instances are located at the inter-cluster interface. The randomized BCGD method does the same mistakes, however also prone to misclassifications in other areas of feature space 15.

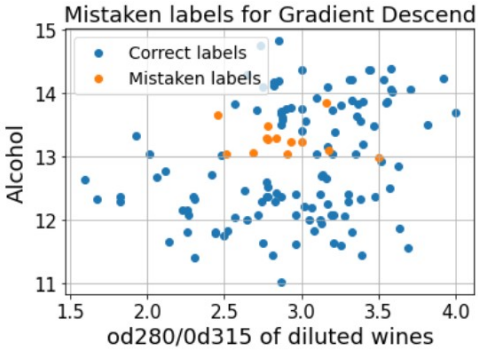

Figure 14. Visualized mistakes of Gradient Descend method. Note that virtualy all mistakes correspond to border points, at the class interface. All BCGD methods, except for the randomized BCGD, exert the same mistake pattern

7.2. HTRU2 Dataset

The second real dataset is a large scale one, taken from the online UCI repository, [Lyo+16]. The dataset is a collection of detected pulsar candidates. A pulsar is a rare type of Neutron star that produce radio emissions. The task is to classify if a given sample is a pulsar or not.

The original dataset is around 17000 samples, so to scale back the problem in our contex, we took a subset of 5 000 examples. Each observation of the dataset has 8 features, so in order to choose the best couple to fit we initially visualized all the possible clusters between the features and then picked the one that seemed most separated.

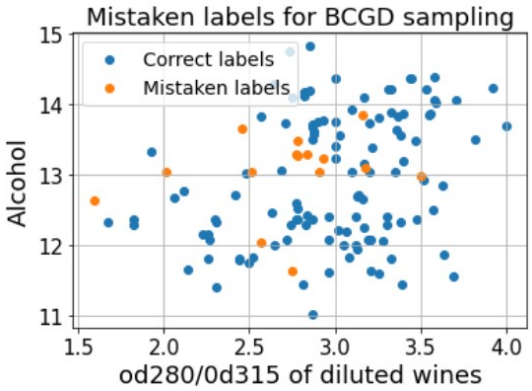

Figure 15. Visualized mistakes of randomized BCGD method. Note how the method makes mistakes not only at the border, but in different areas of feature space

After min-max scaling the data, the experiment scheme was introduced:

- Delabelling 97% of the dataset (16;
- Running each optimization method with best-fit learning rate, with limit of 30 iterations;
- Stopping condition if $\delta_i/\delta_{i-1} < 1\%$;
- Comparing achieved accuracies and analyzing mistakes.

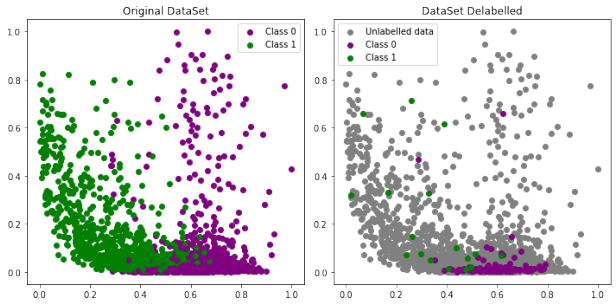

Figure 16. On the left, the original labelled dataset; on the right, the dataset after having delabelled the 97% of the original one

As in the previous sections, the algorithms can be explained into two parts: the GD and BCGD cyclic, permuted models behave consistently, while the BCGD randomized takes more computational time and iterations.

While the GD and BCGDs methods reach the achieve significant decrease in loss in just 2 iterations, the BCGD randomized reaches the maximum of the iterations (30 for didactic purpose) with loss values shill much higher.

Thus, the labelling accuracies, achieved by GD and BCGD methods are approximately 89% (Fig. 17).

The BCGD randomized method, after 30 iterations, managed to achieve loss, 21 times higher than that of the

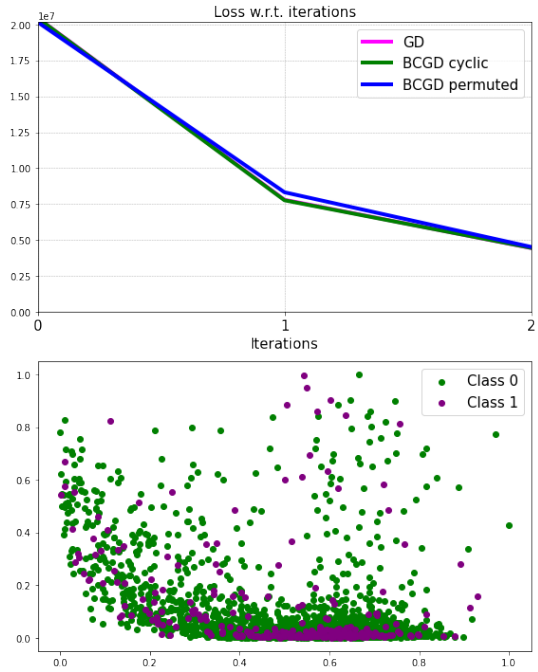

Figure 17. Loss decrease for GD, BCGD cyclic and BCGD permuted algorithms. The loss decreases significantly just after one iteration; The plot on the bottom illustrates classification results after last iteration for GD algorithm

rest of algorithms. After only 2 iterations, all considered models reach loss values, which are 4% of randomized BCGD loss after 30 iterations. The final accuracy was around 49%, which is almost twice less, compared to the rest of the models (Fig. 19). The visualized predictions for randomized BCGD indicate that the algorithm underperforms especially at the inter-cluster interface 18.

Number of iterations, accuracies and CPU time are summarized on figure 19. One can see, the under-performance of randomized BCGD, compared to the rest of the algorithms.

8. Conclusion

This paper compares the performance of gradient-based optimization algorithms, applied to solve the semi-supervised labelling problem.

In the part with synthesised dataset of 5 000 points, GD and BCGD algorithms with permutation and cycling strategies achieved 99.9% accuracy after the first iterations. Over iterations, a decrease of loss function was visualized, however no significant difference between the methods was observed.

In contrary, randomized BCGD method was characterized with 5% labelling accuracy and around 2 times higher CPU and iteration requirements.

With fine-tuned hyperparameters, GD, BCGD cyclic

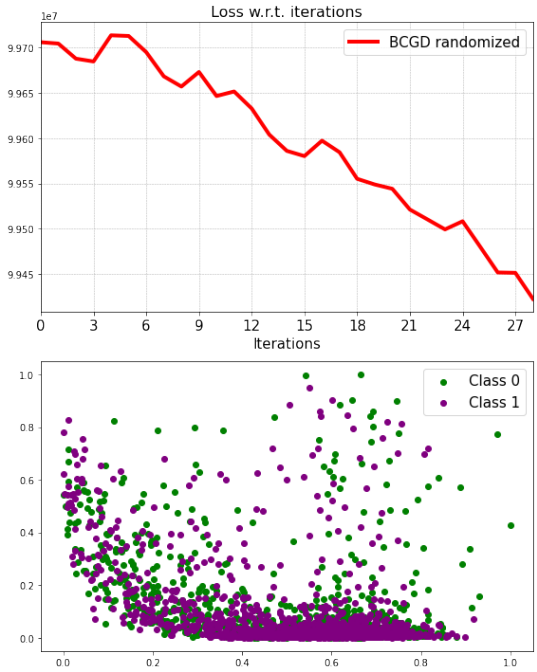

Figure 18. Loss decrease for randomized BCGD algorithm. Notice the slow and gradual loss decrease, which is not enough for a good performance. After 2 iterations, loss value of the rest of algorithms, is 4% of randomized BCGD loss after 30 iterations; On the bottom, the visualization of randomized BCGD predictions shows, that the algorithm hasn't learned enough to distinguish the two clusters

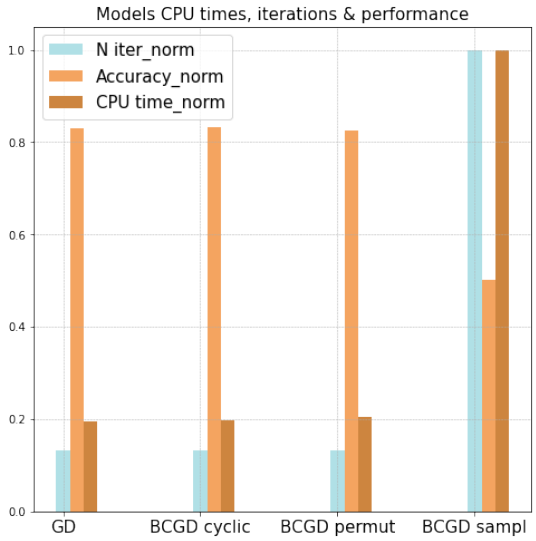

Figure 19. Performance of the algorithms for HTRU2 dataset (accuracy, number of iterations and CPU time)

and BCGD permuted methods showed the best accuracy of 89% on HTRU2 real dataset, consisting of 5,000 samples, 97% of which were delabelled. The algorithms converged just within 2 iterations and achieved loss value of 4% of the one, reached by BCGD randomized after 30 iterations. According to visualized predictions, the majority of mistakes account for the data at the interface between clusters.

Also with the wine dataset, CPU requirements of the GD, BCGD cyclic and permuted BCGD methods do not exceed 30% of that for randomized BCGD (around 10 seconds against the average of 2.24). In addition, randomized BCGD required 300 iterations to achieve 88% of correct classifications, while the rest of the methods achieved 90% metrics within 27 iterations.

In summary, according to three considered datasets, the randomized BCGD algorithm is the least performing in terms of accuracy, as well as CPU and iterations requirements. GD and BCGD methods allowed to reach similar accuracies (99.9%, 90% and 89% for the considered datasets respectively), while requiring approximately equal CPU time and number of iterations.

References

[Lyo+16] R. J. Lyon et al. "Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach". In: *Monthly Notices of the Royal Astronomical Society* 459.1 (2016), pp. 1104–1123. ISSN: 0035-8711. DOI: 10.1093/mnras/stw656. URL: https://doi.org/10.1093/mnras/stw656.