University of Padua - Department of Mathematics
Course: Optimization for Data Science
Academic year: 2021-22.
Professsor: F. Rinaldi

Pietro Maria Sanguin - 2063170 - pietromaria.sanguin@studenti.unipd.it
Emma Roveroni - 2058618 - emma.roveroni@studenti.unipd.it
Marco Tesser - 2058223 - marco.tesser.1@studenti.unipd.it
Angela Bissacco - 2058164 - angela.bissacco.1@studenti.unipd.it

Report delivery date: 11/05/2022

Clustering with unlabeled data

1 The problem

Most of the times real world data come without a description (or label) causing trouble in creating complete databases. In this report we implement some methods to achieve this purpose. We start considering a classification problem with two classes $\{-1, 1\}$ and, since our aim is to classify data based on their features, we want data with similar features to be labeled the same. In order to do this we should minimize a function $f : \mathrm{R}^u \longrightarrow \mathrm{R}$. Therefore the problem is:

$$\min_{y \in \mathrm{R}^u} \sum_{i=0}^{l} \sum_{j=1}^{u} w_{ij}(y^j - \bar{y}^i)^2 + \frac{1}{2} \sum_{i=0}^{u} \sum_{j=1}^{u} \bar{w}_{ij}(y^i - y^j)^2$$

where: 'l' is the number of labeled examples, 'u' is the number of unlabeled examples, w_{ij} is the weight (similarity) between i-th labeled example and j-th unlabeled example and similarly $\bar{w}_{ij}$ is the weight between i-th unlabeled example and j-th unlabeled example. The terms $\bar{y}^i$ and y^j stand for the i-th labeled example and j-th unlabeled example, respectively.
The gradient of the problem is:

$$\nabla_{y^j} f(y) = 2 \sum_{i=0}^{l} w_{ij}(y^j - \bar{y}^i) + 2 \sum_{i=0}^{u} \bar{w}_{ij}(y^i - y^j)$$

To find the order of magnitude of the step size we computed the Hessian of the problem which results:

$$H_{ij}(y) = -2\bar{w}_{ij} + \left(2 \sum_{i=0}^{l} w_{ij} + 2 \sum_{i=0}^{u} \bar{w}_{ij} \right) \delta_{ij}$$

where δ_{ij} is the Kronecker delta. One way to choose a step size α is by taking the inverse of the maximum eigenvalue of the Hessian which, in this convex case, is the Lipschitz constant (L) of the problem. We are using the fixed step size $\alpha = \frac{1}{L}$ in all the implemented algorithms.

1.1 Similarity measures

Similarity measure is a way of measuring how data samples are related and close to each other. In particular similarity is calculated from the spatial distance between the data samples. Usually if two objects belong to the same class their distance will be small, otherwise being the distance greater, the similarity will be lower.

There are many ways to define a similarity measure. We defined the weight matrix w_{ij} as:

$$w_{ij} = e^{-\gamma \|\vec{x}_i - \vec{x}_j\|^2}$$

where $\vec{x}_i$ and $\vec{x}_j$ are the coordinates of the i-th labeled and j-th unlabeled example, respectively and the γ parameter is a shrinking parameter useful when dealing with real datasets.[1] The matrix $\bar{w}_{ij}$ has the same form, but is computed among the i-th unlabeled and j-th unlabeled example.

1.2 Stopping criteria

The stopping criteria, which are the same for all the algorithms, are the following:

- the iterations of the algorithm have reached the maximum number;
- one (determined by parameter stopcr) between:
  - the function has reached the target value fstop;
  - the absolute value of the gradient has reached the optimal tolerance.

2 Data set

2.1 Random generated data

The computer generated data are points extracted randomly from two normal distributions which produce two clusters of points as in the following figure. In all algorithms the unlabeled

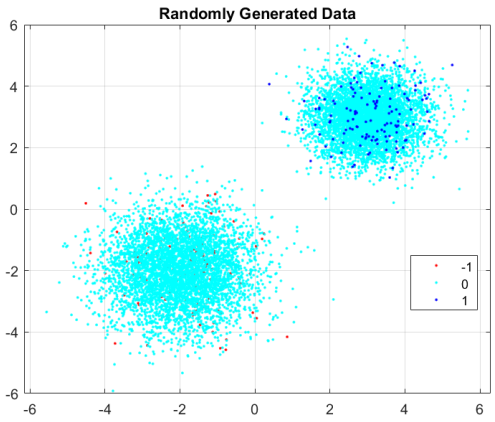

Figure 1: Random generated data classes, zero stands for unlabeled

examples are marked with zero.

[1] If not differently specified $\gamma = 1$.

2.2 Real dataset

The real dataset we chose to observe the behaviour of the implemented algorithms, is the *Dry Bean Dataset* [1]. This data set is provided by a Kaggle challenge and describes seven different type of dry beans (Horoz, Sira, Seker...), taking into account seventeen features such as: area, perimeter, eccentricity, solidity.... Considering the dimension of this data set, we performed some data preprocessing before using it. We selected only two features (perimeter and eccentricity) and two classes (Horoz and Sira). The data are labeled with 0, if it is going to be an unlabeled data, otherwise 1 (Horoz) or -1 (Sira). Furthermore we scaled the data into the range $[0, 1]$. The next plot shows the beans data and their random labelling.

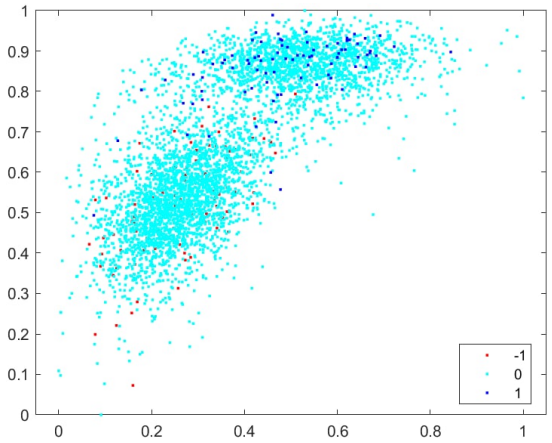

Figure 2: Beans data classes

3 Minimization algorithms

We use three different versions of the gradient method:

- Gradient Descent;
- Randomized Block Coordinate Gradient Descent (BCGD);
- Cyclic Block Coordinate Gradient Descent (BCGD).

Gradient methods are algorithms used to solve problems of the following type:

$$\min_{y \in \mathrm{R}^n} f(y)$$

Since the clustering problem belongs to this class, we implement, with MATLAB code, the three aforementioned algorithms in order to get an optimal solution. Fortunately the problem is convex and therefore there is only one optimal solution. In order to make the attached code more readable and understandable we put a list of the input and output parameters in the appendix section (§ 5).

3.1 Gradient Descent

Gradient Descent is the most famous gradient optimization algorithm. This algorithm minimizes the function $f(y)$ iteratively moving in the direction of the negative gradient (steepest descent)

of $f(y)$.
More specifically the steps involved in the algorithm are the following:

1. Choose a starting point
2. Compute the gradient at the current point
3. Compute the step size
4. Execute a step in the opposite direction to the gradient:

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

5. Repeat from point 2 until stopping condition criteria are reached.

In our project the starting point is passed as an input argument in the function: the vector *y_un*. The function contains a `while` in which all the parameters are updated following the gradient descent rules.
So as the first step of the algorithm we compute the gradient at the point of the current iteration, then we perform the gradient descent step to obtain the new point and finally we compute the value of the function in the new point.

3.1.1 Randomly generated data results

The stopping criteria we decided to use was the absolute value of the gradient. The accuracy is computed at each iteration using as 'activation function':

$$h(y) = \Theta(y) - \Theta(-y)$$

where $\Theta(y)$ is the Heaviside and $\gamma = 1$.
In the following plots, it is possible to observe the results we obtained. Since the accuracy reaches a value equal to 1 very soon, we chose 150 as the maximum number of iterations. In figure 3 we can see the evolution of the function's error during the execution of the algorithm. The error corresponds to the difference between the value of the function at each iteration and the minimum value of the function. In figure 4 the accuracy of the algorithm is shown which, because of the activation function, takes immediately the value 1.

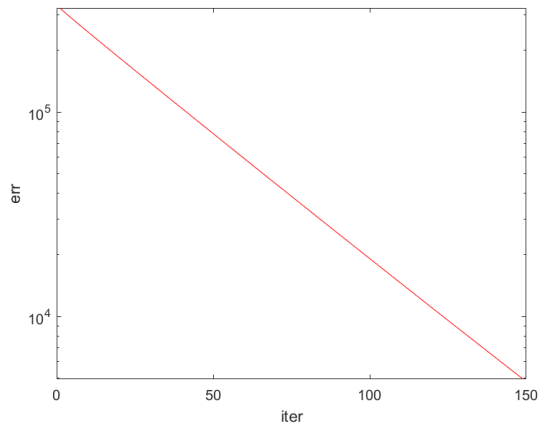

Figure 3: Problem function descent

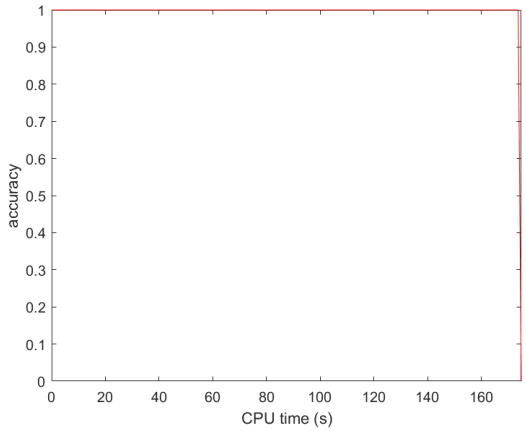

Figure 4: Accuracy as CPU time increases

In figure 5 is shown how the gradient descent algorithm has predicted the y. As we can see, now each unlabeled point has assigned a corresponding class between $\{-1, 1\}$, confirming the high value of the accuracy.

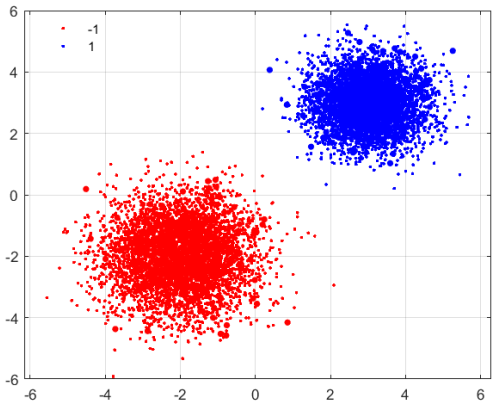

Figure 5: Predicted cluster with gradient descent method and random generated points

3.1.2 Real data set results

In this first case the algorithm gave since the beginning very high accuracy. The γ parameter, in this real data set computation, is set to 1000 as with all others real data sets computations. The value was selected with a grid-search method in which different values of γ were tried and then the one giving the best performance was chosen.

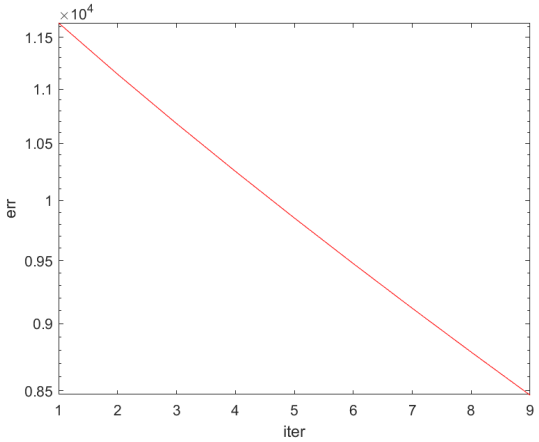

Figure 6: Problem function descent

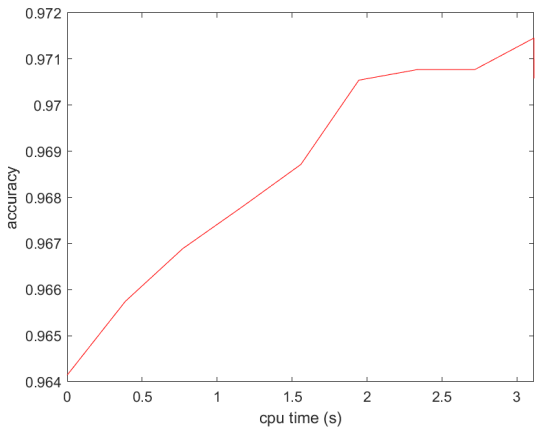

Figure 7: Accuracy as CPU time increases

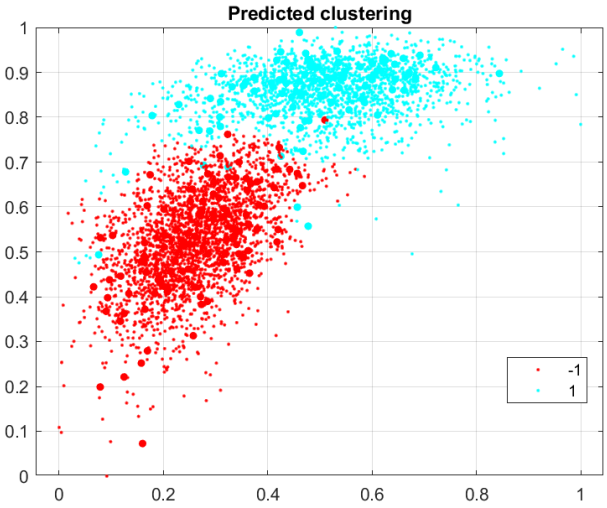

Figure 8: Predicted cluster with gradient descent method and real data set

Block Coordinate Gradient Descent

The main problem with the gradient descent method is that finding the full gradient can be computationally expensive. For that reason we also tried to use the family of block coordinate gradient descent algorithms. These algorithms divide the whole gradient into blocks each with a certain number of variables and then the gradient descent step is performed at each iteration for a single block, thus decreasing the computational cost of each iteration. In our project the blocks are chosen to be of dimension 1. The identity matrix U is defined with size equal to the number of blocks. The i-th block is given by $y^{(i)} = U_i^T y$. The i-th component of the partial derivative is denoted by $\nabla_i f(y) = U_i^T \nabla f(y)$. The general scheme of these algorithms imposes these steps:

1. Choose a starting point
2. Pick a block following the chosen rule
3. Compute the partial gradient (only the one of the variables present in the chosen block) at the current point.
4. Compute the step size
5. Execute a step in the opposite direction to the gradient, following the formula

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

6. Repeat from point 2 until stopping condition criteria are reached.

The stopping criteria are the same as the ones used in the gradient descent method.

3.2 Randomized Block Coordinate Gradient Descent (Random BCGD)

The randomized block coordinate gradient descent is a type of BCGD method in which the choice of the block at each iteration is completely random. As before, there is a `while` in which all the parameters are updated following the randomized BCGD rules. Therefore, the first thing to do is to randomly choose one of the blocks and subsequently compute the gradient relative only to that block. After that, the gradient step is performed to obtain the new value of y_{un}. Like in the gradient descent case, the same stopping criteria are exposed and the value of the function is updated.

3.2.1 Random generated data results

As for the gradient descent the activation function used is $h(y)$ and $\gamma = 1$. After 10000 iterations we get the following results:

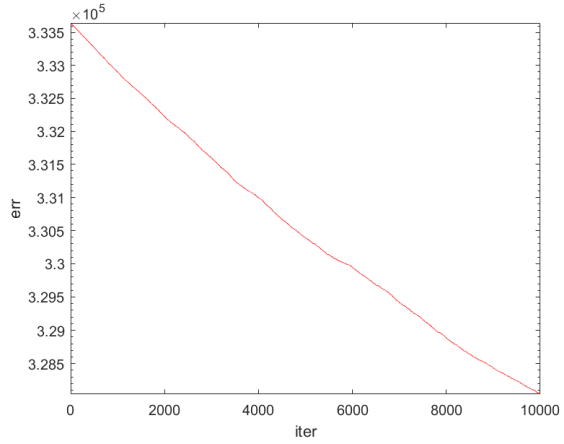

Figure 9: Problem function descent

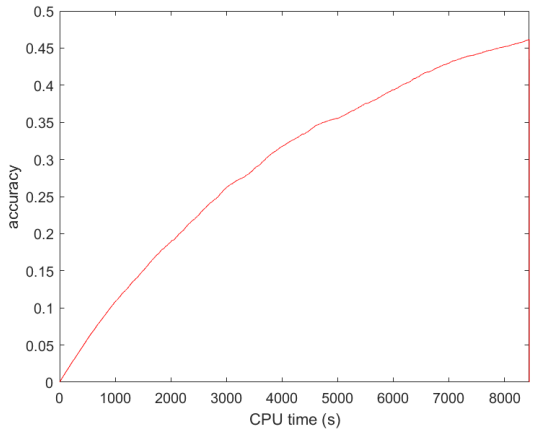

Figure 10: Accuracy as CPU time increases

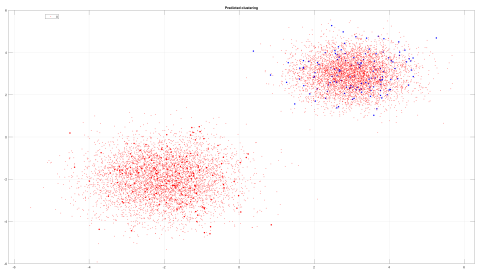

Figure 11: Predicted cluster with random BCGD method

As shown in the previous figures the accuracy grows as time passes since more and more components are updated. In this case the result is poor due only to the low number of iterations. Had the algorithm been run for longer it would have provided a better accuracy.

3.2.2 Real data set results

In this scenario we performed 25000 iterations of the algorithm. It is worth noting that the accuracy obtained also in this scenario was limited by the number of iterations of the algorithm, if it had been running for longer it would have provided an even better accuracy.

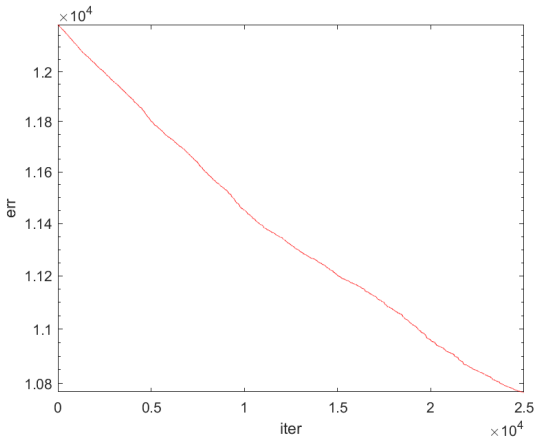

Figure 12: Function descent

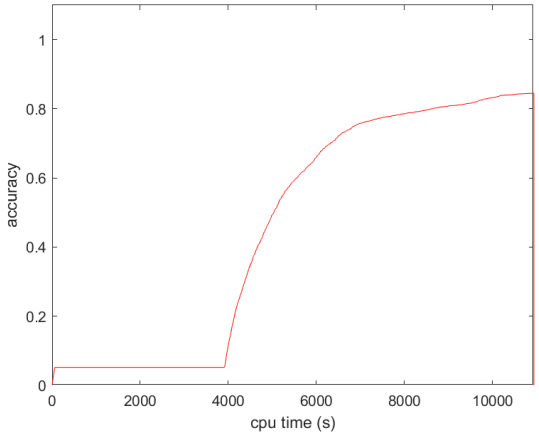

Figure 13: Accuracy as CPU time increases

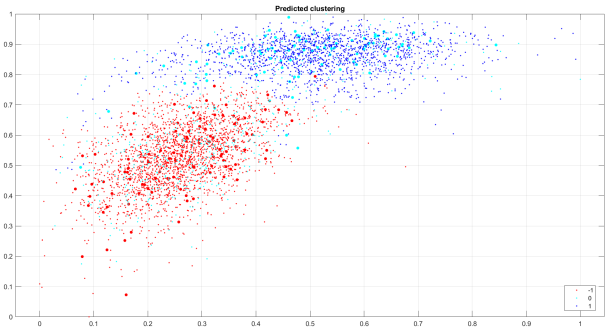

Figure 14: Predicted cluster with gradient descent method and real data set

3.3 Cyclic Block Coordinate Gradient Descent (Cyclic BCGD)

The cyclic block coordinate gradient descent iterates cyclically the blocks from which the gradient is computed. The procedure is similar to the one explained for the random BCDG in 3.2. The only difference is in the selection of the blocks. In this case they are used in order and, once the partial gradient has been computed with all the blocks, the cycle restarts from the first one.

3.3.1 Random generated data results

As for the BCGD randomized the activation function used is $h(y)$ and $\gamma = 1$. After 10000 iterations it results:

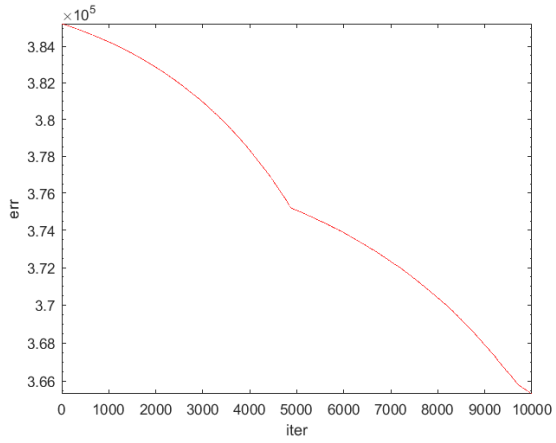

Figure 15: Function descent

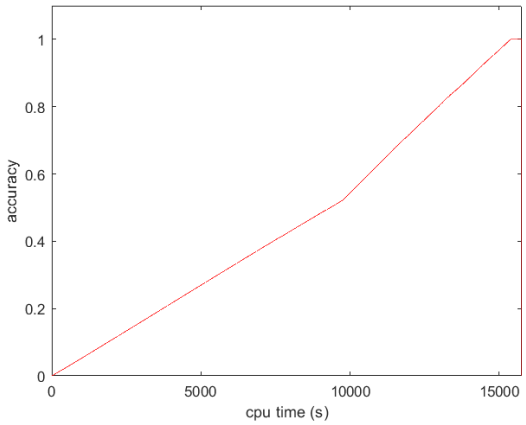

Figure 16: Accuracy as CPU time increases

As shown the accuracy goes to one after running the algorithm therefore the predictions are all correct as shown in figure 5.

3.3.2 Real data set results

In this scenario we performed 6000 iterations and reached an almost perfect accuracy.

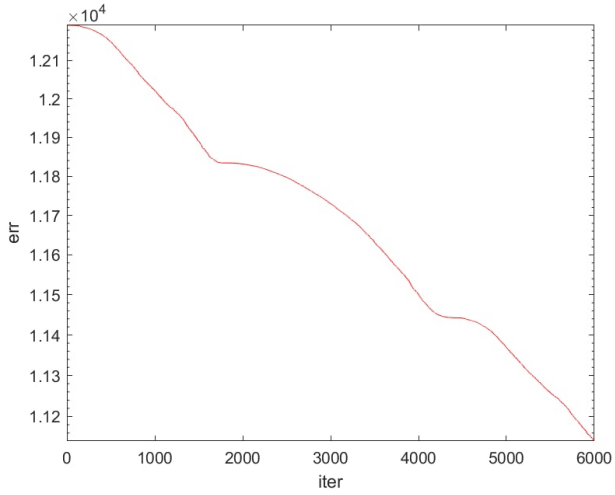

Figure 17: Function descent

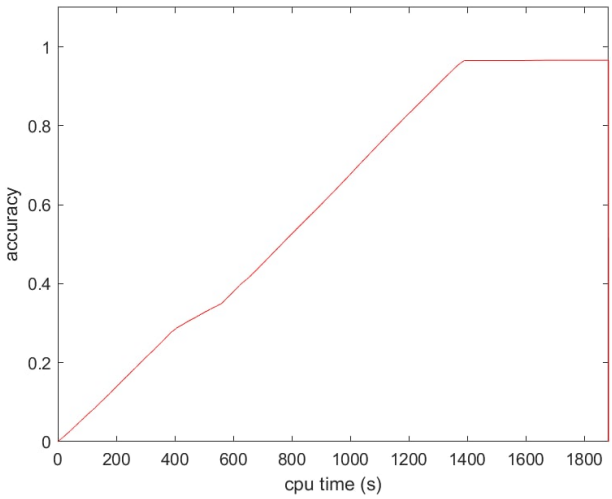

Figure 18: Accuracy as CPU time increases

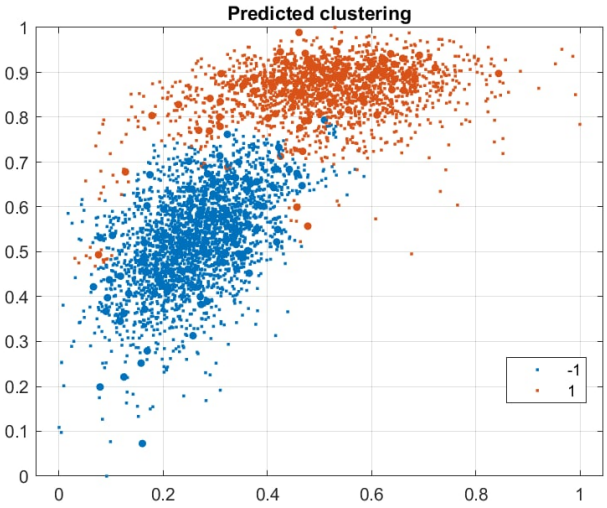

Figure 19: Predicted cluster with cyclic BCGD and real data set

4 Conclusions

In all algorithms, except the BCGD random due to the more iterations needed, we achieve good accuracies in both real and random generated data.

5 Appendix

The algorithms take as parameters the following arguments:

- w: matrix with the weights between labeled and unlabeled data;
- y_lab: vector with classes of labeled data;
- w_bar: matrix with the weights between unlabeled data;
- y_un: starting point;
- lc: Lipschitz constant, which is computed at the maximum of the Hessian matrix's eigenvalues;
- *verbosity*: if it's a value larger than 0, it shows what the algorithm is doing;
- *maxit*: number of maximum iteration that the algorithm can reach (we chose 10000);
- *esp*: optimal tolerance (1.0e-4);
- *fstop*: target value of the function for stopping the algorithm;
- *stopcr*: indicates which stopping criteria we want to choose.

At the end of the calculation the algorithms will return the following parameters:

- x: minimum point;
- *it*: number of iteration the algorithm reached once it has finished to run;
- *fx*: value of the function updated at each iteration;
- *ttot*: total CPU time;
- *fh*: array containing the value of the function at iteration;
- *timeVec*: array containing CPU time at each iteration;
- *gnrit*: norm of the gradient at each iteration.

References

[1] M. Koklu and I.A. Ozkan. Multiclass classification of dry beans using computer vision and machine learning techniques. computers and electronics in agriculture, 2021. https://doi.org/10.1016/j.compag.2020.105507.