Optimization for Data Science – Homework 1

Semi-supervised classification problems: analysis with Gradient Descent and BCGD algorithms

Authors:

Omid Airom *student id:2071818* Mehran Farajinegarestan *student id: 2071980*

Nazli Hanifi *student id: 2072020* Laura Legrottaglie *student id: 2073222*

1-Intro

Semi-supervised learning is a machine learning technique that deals with situations where the available data has only a small portion of labeled data. Unlike supervised learning, where the input data is fully labeled, or unsupervised learning, where there are no labels, semi-supervised learning tries to make use of both labeled and unlabeled data to train a model. The aim is to leverage the information present in the unlabeled data to improve the performance of the model. Semi-supervised learning is a crucial technique in scenarios where obtaining labeled data is expensive, time-consuming, or even impossible, but there is still a large amount of unlabeled data that can be used. This approach has been successfully applied in various domains, such as natural language processing, computer vision, and speech recognition.

This study explores the semi-supervised learning problem using various optimization algorithms and compares their results. The analyzed algorithms are standard Gradient Descent and Block Coordinate Gradient Descent (BCGD) with the Randomized rule and with the Gauss-Southwell rule. These algorithms have been evaluated on a synthetic dataset and on two real datasets.

2-Problem Statement

2.1-Loss function

We are facing a binary classification problem where there are l labeled examples: $(x^i, \bar{y}^i)$, $i = 1, \dots, l$ and u unlabeled ones: (x^j, y^j) , $j = 1, \dots, u$. The objective is to determine the labels y^j for the unlabeled points by assuming that comparable features will have similar labels. To accomplish this, it's possible to define a loss function to minimize. The optimization problem that we have is:

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

where the first term considers the relationship between labeled and unlabeled data points and the second one, instead, the relationship between unlabeled ones. In the cost function $w_{i,j}$ represents the similarity between the labeled instances i and unlabeled examples j, and $\overline{w}_{i,j}$ as the similarity between unlabeled instances.

2.2-Similarity function:

The weights in the loss function reflect the degree of similarity between the points. Higher weights indicate a stronger similarity between points, while lower weights suggest a weaker similarity. Therefore, further are the two points smaller will be the weight because in this case, the classes of the two points should be different $y^j \neq \bar{y}^i$ or $y^j \neq y^i$ and it's the value of the weight that tries to shrink the single term in the summation in the loss function.

The similarity function used is the following:

$$w(\vec{v}, \vec{z}) = e^{-a||\vec{v} - \vec{z}||_2^2}$$

where $||\vec{v} - \vec{z}||_2^2$ is the L2 norm or the Euclidean distance between two points. We use the exponential function to normalize the value of the weight measures and the presence of a hyperparameter (a) determines the extent to which a point should be deemed relevant for the evaluation of another point, based on their proximity.

2.3-Generating the synthetic Dataset

To undertake our project, we generate a dataset in the form of two moon-shaped curves: using cos and sin functions 10000 numbers have been created between certain ranges:

(-1, 1) for x_1 of the class purple with label -1

(0, 1) for x_2 of the class purple with label -1

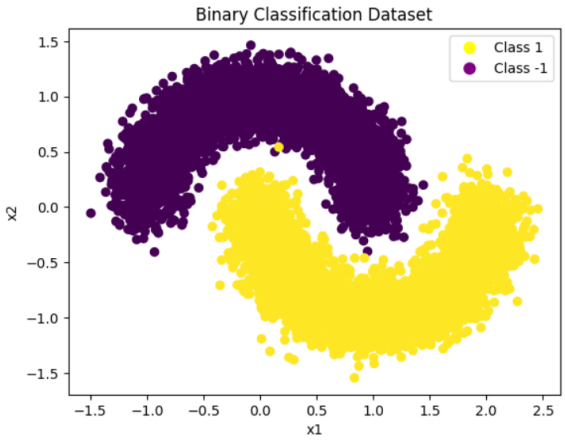
(0, 2) for x_1 of the class yellow with label -1

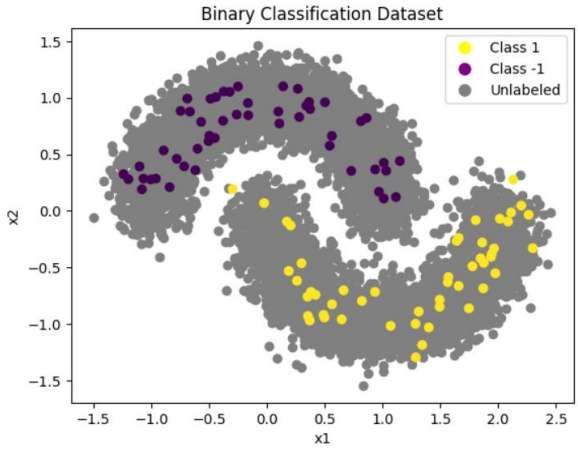
(-1, 0) for x_2 of the class yellow with label -1

obtaining in this way two intersecting semi-circles. Then on those points, a noise sampled from $N(0,0.15)$ distribution has been applied.

As we are doing the semi-supervised learning, we assign a 1% of the points to a set which was randomly chosen for the labeled section in a stratified manner. We want that the number of labeled points of each class in the labeled set is based on the proportion of the initial points in each class. Therefore, in this case, since the number of samples in each class is the same (5000) we'll have an equal number of points of purple and yellow class also in the labeled set. In addition, in the initialization phase, we set all the unlabeled data points labels equal to zero as starting "point" of the algorithms.

where the first term considers the relationship between labeled and unlabeled data points and the second one, instead, the relationship between unlabeled ones. In the cost function $w_{i,j}$ represents the similarity between the labeled instances i and unlabeled examples j, and $\overline{w}_{i,j}$ as the similarity between unlabeled instances. We set the value of the hyperparameter $a = 50$ in the entire evaluation of this dataset.





2.4-Gradient:

With respect to y^j the gradient of the function is:

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

$$= 2\left[\sum_{i=0}^{l} w_{ij}\left(y^j - \overline{y}^i\right) + \sum_{i=0}^{u} \overline{w}_{ij}(y^j - y^i)\right] =$$

$$= 2\left[\sum_{i=0}^{l} w_{ij}\left(y^j - \overline{y}^i\right) + \sum_{i=0}^{u} \overline{w}_{ij}(y^j - y^i)\right] =$$

$$= 2\left[(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij})y^j - \sum_{i=0}^{l} w_{ij}\,\overline{y}^j - \sum_{i=0}^{u} \overline{w}_{ij}\,y^i\right]$$

Since we want to speed up the computation of the gradient in the code, we can fix some constants that are the same in the entire computation:

$$c_1 = 2(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij})$$

$$c_2 = 2\sum_{i=0}^{l} w_{ij}\overline{y}^{j}$$

$$c_3 = 2\sum_{i=0}^{u} \overline{w}_{ij}$$

2.5- The Hessian and the step size:

Throughout the analysis, we use a fixed step size. To ensure a suitable step size, we initially calculate the Lipschitz constant, denoted as L. This constant remains the same throughout the process. It is determined by the highest eigenvalue of the Hessian matrix.

The Hessian of the function f is the following matrix:

$$H_f = 2 * \begin{bmatrix} (\sum_{i=0}^{l} w_{i1}) + (\sum_{i=0}^{u} \overline{w}_{i1}) - \overline{w}_{11} & -\overline{w}_{12} & \cdots & -\overline{w}_{1n} \\ -\overline{w}_{21} & (\sum_{i=0}^{l} w_{i2}) + (\sum_{i=0}^{u} \overline{w}_{i2}) - \overline{w}_{22} & \cdots & -\overline{w}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\overline{w}_{n1} & -\overline{w}_{n2} & \cdots & (\sum_{i=0}^{l} w_{in}) + (\sum_{i=0}^{u} \overline{w}_{in}) - \overline{w}_{nn} \end{bmatrix}$$

If $k = j \rightarrow$

$$\nabla_{y^j y^j} f(j) = 2\left[(\sum_{i=0}^{l} w_{ij}) + (\sum_{i=0}^{u} \overline{w}_{ij}) - \overline{w}_{jj}\right]$$

If $k \neq j \rightarrow \nabla_{y^j y^k} f(j) = -2\overline{w}_{kj}$

3-The algorithms' definition

As previously mentioned, the analysis incorporates three optimization algorithms: Gradient Descent, Randomized BCGD (using random permutations and random sampling of the indexes) and Gauss-Southwell BCGD. 100 iterations have been exercised for each of the algorithms mentioned above (for BCGM, one iteration is performed when we update all the unlabeled data points).

The stopping early condition before getting to the 100th iteration is ($\|\nabla_f(x_k)\| < eps$) with $eps = 10^{-4}$

3.1-Gradient Descent

The standard gradient descent algorithm at each iteration computes the partial derivative for the points all at once and adjusts their labels moving along the descent direction with the proper step size. This process continues iteratively until a predefined stopping condition is satisfied or we reach the maximum number of iterations.

$$y_k = y_{k-1} - \frac{1}{L}\nabla f(y_{k-1})$$

3.2-BCGD

For the BCGD (Block Coordinate Gradient Descent) algorithms we fix the size of the block equal to 1. These algorithms, instead of updating all the points at once using the gradient as GD (procedure computationally expensive since the gradient is a vector of |unlabeled points| elements), choose one element of the gradient at a time (since the block size is 1) and update just the label of that specific point. This process iterates for |unlabeled points| times for each iteration, and if the stopping condition is not yet met, a new cycle begins.

The difference between Randomized BCGD and BCGD with the Gauss-Southwell rule is how these algorithms choose the index of the element of the gradient to update. Randomized BCGD chooses the index randomly while Gauss-Southwell BCGD picks the index of the element that has the maximum norm of the gradient (since the size of the block is 1 the norm is equivalent to the absolute value). In this second case, the algorithm performs in this way in order to have the steepest slope towards the minimum and because we choose the block that violates the early stopping condition more than the other ones.

We analyze two options for point selection for the Randomized BCGD: random permutation and random sampling. In random permutation, at each iteration, all points are selected randomly without replacement, whereas in random sampling, at each iteration a point can be selected randomly multiple times.

Randomized BCGD:

$$y_k^J = y_{k-1}^J - \frac{1}{L}\nabla_{y^J} f\left(y_{k-1}^J\right)$$

BCGD with Gauss-Southwell rule:

$$j = \underset{i\in[1,u]}{Argmax}||\nabla_i f(y)||$$

$$y_k^j = y_{k-1}^j - \frac{1}{L}\nabla_{y^j} f(y_{k-1}^j)$$

3.3-BCGD with the Gauss-Southwell rule

Since at each iteration in BCGD with the Gauss-Southwell rule we need to find the block that has the maximum norm of the gradient and this requires the computation of the whole gradient, we need to find a smart way to update the gradient at each iteration.

Supposing that the index with the maximum norm of the gradient is *j* and therefore, the update is on the *j* variable. We need to consider the update at the $k+1$ iteration differently for the variable j and for all the other variables i.

Let's start with the update of the gradient for all the variables that remain constant in the iteration:

$$\nabla_{y_{k+1}^i} f(y_{k+1}) =$$

$$2\left[(\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij})y_{k+1}^i - \sum_{i=0}^{l} w_{ij}\,\overline{y}^i - \sum_{i=0}^{u} \overline{w}_{ij}\, y_{k+1}^i\right]$$

We fix the following constants:

$$a^i = (\sum_{i=0}^{l} w_{ij} + \sum_{i=0}^{u} \overline{w}_{ij}) \qquad b = \sum_{\boldsymbol{i}=0}^{\boldsymbol{l}} \boldsymbol{w_{ij}}\overline{y}^i$$

That follows to:

$$\nabla_{y_{k+1}^i} f(y_{k+1}) = 2[a^i y_{k+1}^i - b - \sum_{i=0}^{u} \overline{w}_{ij} y_{k+1}^i] =$$

$$\nabla_{y_{k+1}^i} f(y_{k+1}) = 2[a^i y_{k+1}^i - b - \sum_{i=0}^{u} \overline{w}_{ij} y_{k+1}^i] =$$

$$= 2\,[a^i y_{k+1}^i - b - \left(\overline{w}_{0j} y_{k+1}^0 + \cdots + \overline{w}_{jj} y_{k+1}^j + \cdots \overline{w}_{uj} y_{k+1}^u\right)]$$

Considering that all the labels are the same of iteration k except for the label in position *j* that is the one we update, we can substitute

$$y_{k+1}^j = y_k^j - \frac{1}{L}\nabla_{y^j} f(y_k^j)$$

in the previous equation obtaining

$$= 2\left[a^i y_k^i - b - \left(\overline{w}_{0i} y_k^0 + \cdots + \overline{w}_{ji}\left(y_k^j - \frac{1}{L}\nabla_{y^j} f\left(y_k^j\right)\right) + \cdots + \overline{w}_{ui} y_k^u\right)\right] =$$

$$= 2[a^i y_k^i - b - \sum_{i=0}^{u} \overline{w}_{ij} y_k^i] + 2[\overline{w}_{ji}\frac{1}{L}\nabla_{y^j} f(y_k^j)]$$

But $\nabla_{y_k^i} f(y_{k+1}) = 2[a^i y_k^i - b - \sum_{i=0}^{u} \overline{w}_{ij} y_k^i]$, so we have

$$\nabla_{y_{k+1}^i} f(y_{k+1}) = \nabla_{y_k^i} f(y_k) + 2[\overline{w}_{ji}\frac{1}{L}\nabla_{y^j} f(y_k^j)]$$

For the updated of the gradient of the *j* variable instead

$$\nabla_{y_{k+1}^j} f(y_{k+1}) = 2[a^j(y_k^j - \frac{1}{L}\nabla_{y^j} f(y_k^j)) - b - (\overline{w}_{0j} y_k^0 + \cdots + \overline{w}_{jj}(y_k^j - \frac{1}{L}\nabla_{y^j} f(y_k^j)) + \cdots + \overline{w}_{uj} y_k^u)] =$$

$$= 2\left[a^j y_k^j - b - \sum_{i=0}^{u} \overline{w}_{ij} y_k^i\right] + 2\left[-a^j\frac{1}{L}\nabla_{y^j} f\left(y_k^j\right) + \overline{w}_{jj}\frac{1}{L}\nabla_{y^j} f\left(y_k^j\right)\right]$$

But

$$\nabla_{y_k^j} f(y_{k+1}) = 2[a^j y_k^j - b - \sum_{i=0}^{u} \overline{w}_{ij} y_k^i]$$

Therefore, at the end we have

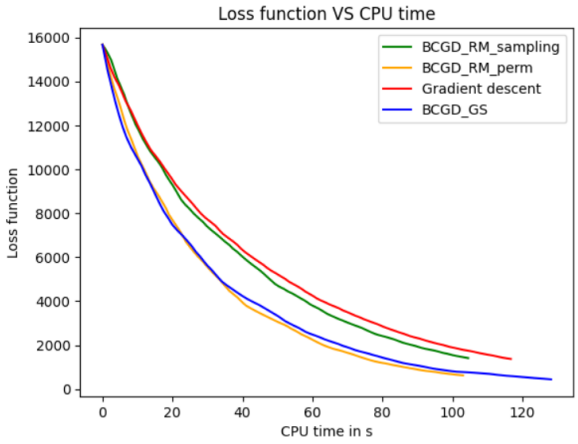
$$\nabla_{y_{k+1}^j} f(y_{k+1}) = \nabla_{y_k^j} f(y_k) + 2[(\overline{w}_{jj} - a^j)(\frac{1}{L}\nabla_{y^j} f(y_k^j)]$$

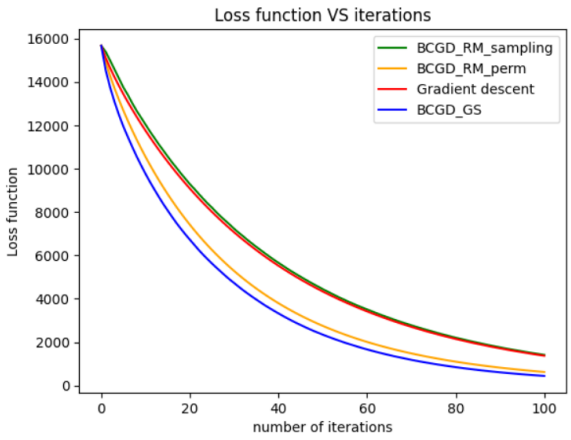
4-Results for the synthetic dataset

4.1- Analysis of loss function

Each algorithm's CPU time and number of iterations will be compared against its accuracy and loss.

All the algorithms complete the maximum number of iterations allowed without encountering the early stopping condition, resulting in all the subsequent images displaying the same number of iterations.



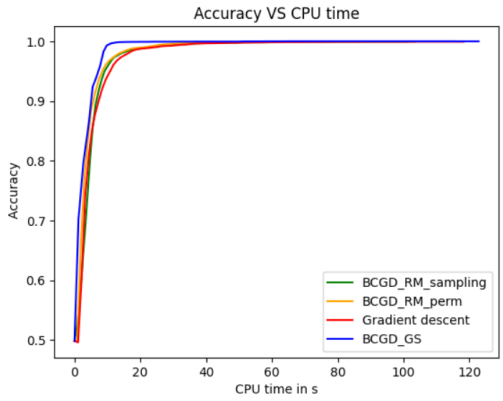


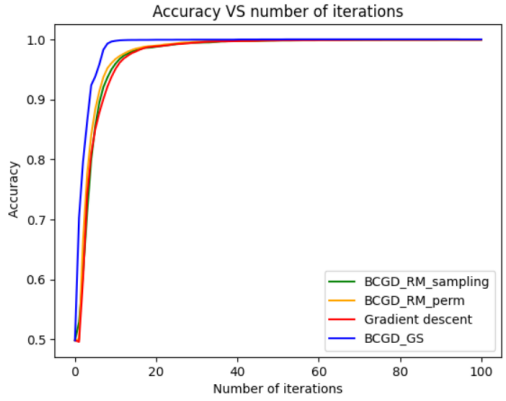
Concretely, we see that both the Randomized BCGD in the permutation version and the Gauss Southwell BCGD, outperformed the Gradient Descent in loss, using the same step size, considering the number of iterations and the CPU time.

The Randomized BCGD with random sampling, instead, has similar behavior with respect to the Gradient. The Gauss-Southwell method outperforms the Gradient Descent since we are facing a high dimensional problem and it's better than the Randomized BCGD since it utilizes first-order information to take steps, which the other method does not.

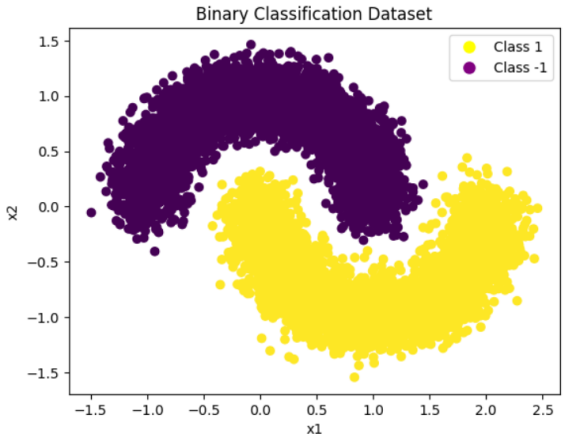
4.2- Accuracy analysis

Since initializing the points to all zeros leads to very high accuracy from the starting iterations, it's more useful to analyze the CPU time and the number of iterations VS accuracy with random initialization of the points between the two classes. Also, in this case, all the BCGD outperform the Gradient Descent and the best is BCGD with the Gauss Southwell rule.

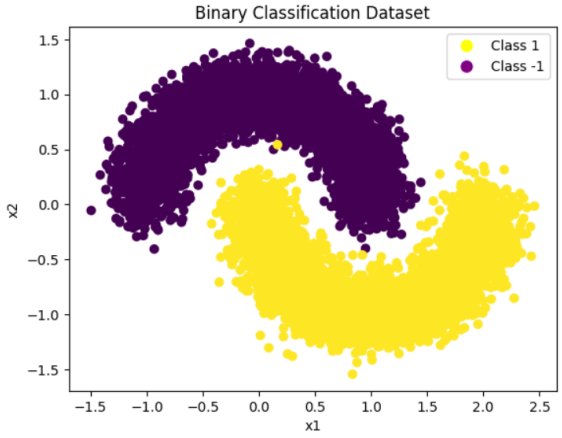




With Gradient Descent we reach a final accuracy of 0.99 and this is the final plot with predicted points compared to the initial one.



Scatter plot with initial values



Scatter plot with predicted values

5-Real datasets

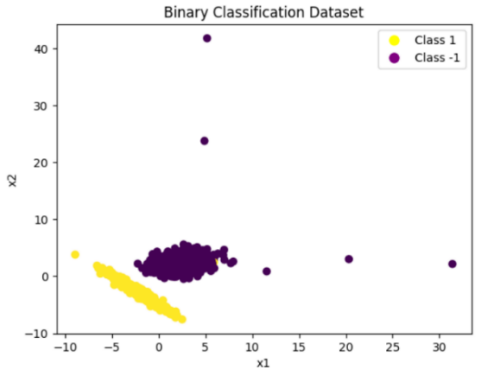
5.1 Smoker dataset

As previously stated, the comparison between the three algorithms is also being evaluated on real datasets.

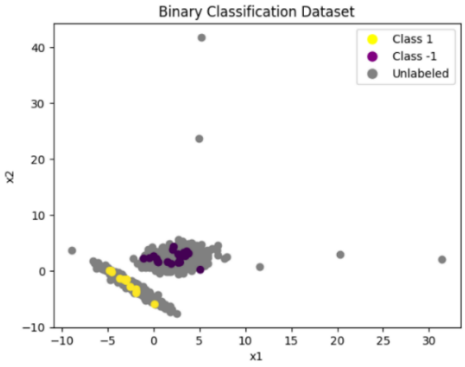
The first dataset is from Kaggle and regards a binary classification problem which aims to predict if a person will have cancer or not using genes and their smoking history. It has various features including specific gene identifiers (Gene2337, Gene35715, Gene12936, Gene1689), as well as genes or proteins associated with smoking-related conditions (FGFR1) and regulatory pathways affected by smoking (GATA4). These features can be analyzed to gain insights into the expression patterns, functional properties, and potential contributions of these genes or proteins to smoking-related health issues. Additionally, the dataset includes features such as "type," which categorizes individuals based on their smoking history (“Smoker”/ “Not smoker”), and "Condition" (“Cancer”/ “Normal”) which provides information about the specific health outcomes (if they can have cancer or not). We encode these categorical features and in particular “Cancer” class corresponds to 1, while “Normal” corresponds to -1.

The loss function of this model depends on the similarity weights, based on the distance between two points. As a result, this model is highly affected by the curse of dimensionality issue, especially when working with high-dimensional data that has a limited number of samples. Here, we have exercised methods based on mutual information and f-regression to reduce the dimensionality of the smoking dataset (from 8 dimensions to 2). Since the number of instances analyzed is 981, we label 3% of them in a stratified manner. The hyperparameter of the similarity function in this case has been set $a = 1$.

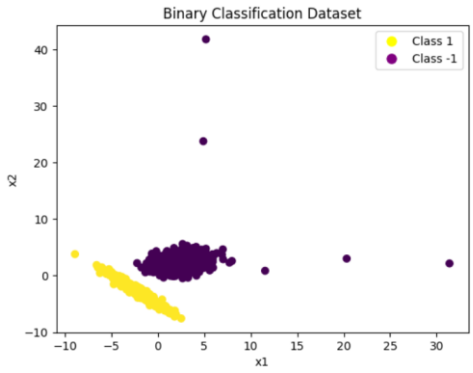
5.1.1- Results for the Smoker dataset



Scatter plot with initial values

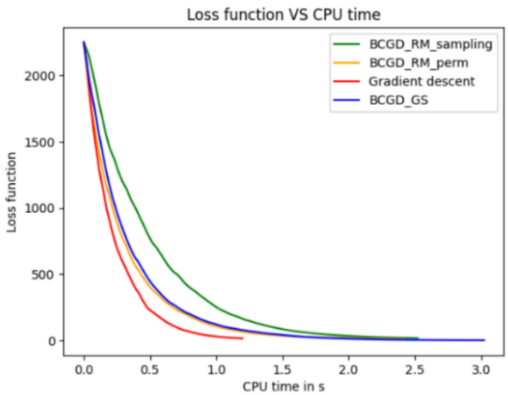


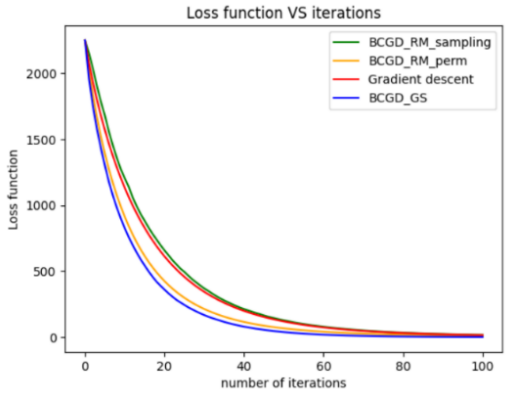
Scatter plot with labeled and unlabeled values



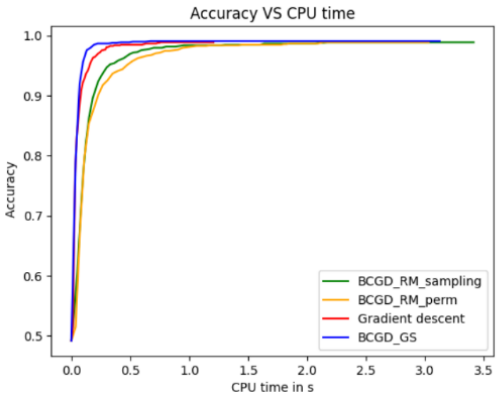
Scatter plot with predicted values

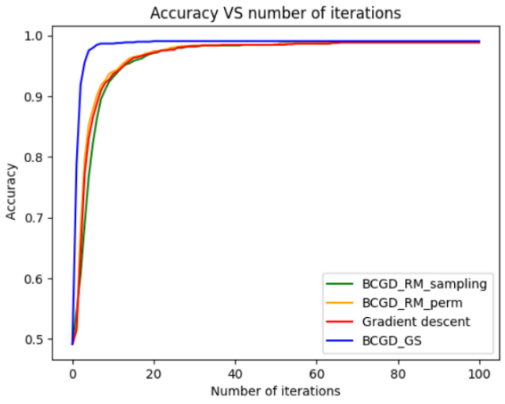
BCGD algorithms beat Gradient Descent in the reduction of the loss function considering the same number of iterations, but in terms of CPU time, however, a full iteration of BCGD approach costs more than one iteration of gradient descent. BCGD with the Gauss-Southwell rule performs the best since the gradient of the function is very sparse.





We reach an accuracy of 0.99 approximately for all the methods and BCGD with the Gauss-Southwell rule reaches the highest accuracy in the least time.



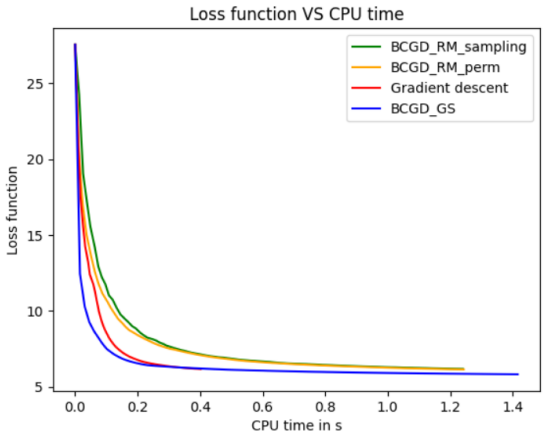


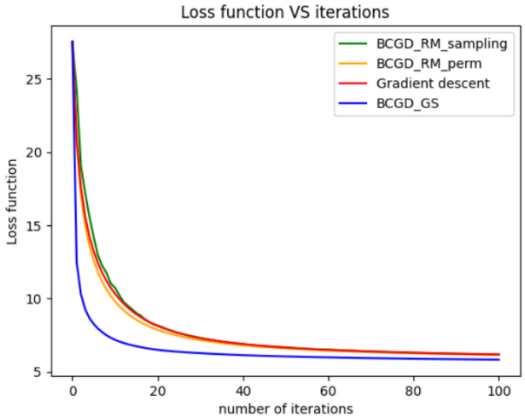
5.2 -Housing dataset

The second data set is a housing price dataset from Kaggle. In this case, to select the features and solve the curse of dimensionality problem the correlation between the attributes of the dataset has been analyzed, and non-relevant features have been removed. We pass from 14 features to 6. Afterwards, the rows with missing data have been dropped, and the remaining data have been normalized between 0 and 1 for each feature separately. Then, the dataset has been divided into two classes: Class 1, which included the data points with a price higher than the median, and Class -1, which included the datapoints with a price below the median. Next, since we have just 393 instances, 10% of the data points have been randomly labeled, while the rest remained unlabeled for the classification task. In this case, the value of the hyperparameter has been set $a = 200$

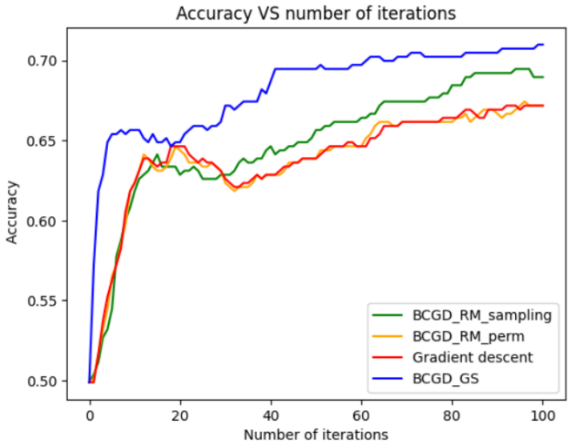
5.2.1 - Results for the Housing dataset

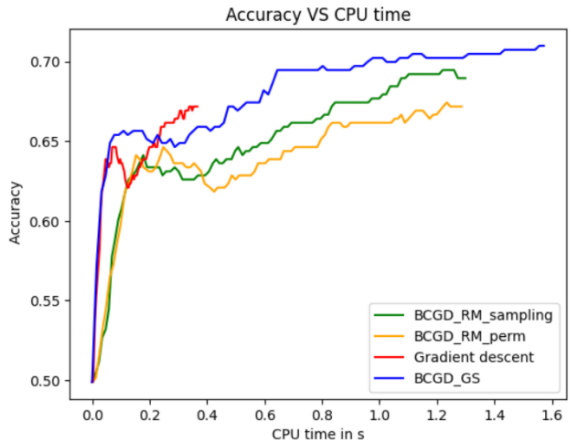
Also in this last case, Block Coordinate Gradient Descent with the Gauss-Southwell rule performs the best followed by Gradient Descent and Block Coordinate Gradient Descent with the Randomized rule.





In term of accuracy, since in this case the data are not very well separated, we reache about 76% accuracy with Gauss-Southwell rule and around 70% with all the others.





6 -Conclusions

The aim of classifying the data is reached with satisfactory accuracy with all approaches considered for all the datasets. The analysis of the loss function with respect to the number of iterations exhibits a victory of the BCGD with the Gauss-Southwell rule for all the datasets. For the synthetic dataset, where the number of unlabeled points is very high, BCGD with the Gauss-Southwell rule wins in terms of the decreasing of the cost function with respect to the CPU time, followed by the BCGD with the Randomized rule. In the case of real datasets, we need to take into consideration some aspects. In the first place, the number of points is relatively small and so we can see that also standard Gradient Descent does a great job, especially in terms of the reduction of the objective function with respect to the CPU time. Moreover, for the dataset of the Housing the two clusters are not very well separated and for this reason, the percentage of labeled points was chosen higher. The accuracy of 76% that has been achieved, though, is an acceptable result given the previously mentioned disadvantages.

To sum up, to have clearly better results for the BCGD methods compared to standard Gradient Descent more points are needed.