Optimization methods: Gradient Descent and BCGD methods with an application on Rice Seeds classification

Alejandra Cruces and Mario Tapia

April 5, 2024

1 Introduction

Semi-supervised learning is a technique that deals with data that has both labeled and unlabeled instances. The goal is to use the information from the labeled data to predict the labeling of the unlabeled data. In this report, we compare three different optimization methods for semi-supervised learning on a synthetic dataset and a real dataset about rice seeds. In both datasets we just keep a small fraction of data with labels. We evaluate the performance of the methods based on their accuracy and time.

2 Approach

In the following subsections, we will describe the approach used for running every optimization method implemented.

2.1 Loss function

The aim of the homework is to run a semisupervised learning task by minimizing the objective function

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

where u is the number of unlabeled examples, l is the number of labeled examples, y is an unlabeled example, $\overline{y}$ is a labeled example, w_{ij} are the weights (similarity) between labeled example i and unlabeled example j, and $\overline{w}_{ij}$ are the weights (similarity) between unlabeled examples i, j. The gradient of the function with respect to y^j, which is needed in order to implement the optimization algorithms, is given by

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

The optimization techniques implemented are: Gradient Descent, Randomized Block Coordinate Gradient Descent, and Gauss-Southwell Block Coordinate Gradient Descent methods.

2.2 Weights calculation

The weights of the function are obtained using similarity measures. A similarity measure is a key component for correctly assigning labels to unlabeled observations. This measure allows us to identify and group the examples that share similar features and assume that they belong to the same class. The first similarity measure implemented is based on the Euclidean distance, which is one of the most commonly used measures in clustering. For two points $\boldsymbol{p}$, $\boldsymbol{q}$, the weight is defined as

$$W(\boldsymbol{p}, \boldsymbol{q}) = \frac{1}{1 + ||\boldsymbol{p} - \boldsymbol{q}||_2}$$

which assumes values between (0, 1]. The smaller the Euclidean distance between two points is, the more similar they are (overlapping points would get the highest similarity 1).

The second function used is the RBF kernel, which is based on the exponential distance between examples. Considering two points $\boldsymbol{p}$, $\boldsymbol{q}$, the weight is given by the expression

$$W(\boldsymbol{p}, \boldsymbol{q}) = e^{-\gamma \|\boldsymbol{p}-\boldsymbol{q}\|_2^2}$$

where γ is a scaling factor that determines how quickly the similarity metric falls off with increasing distance. The expression assumes values between (0, 1]. One advantage of this exponential decay is that it improves the cluster separation when their centers are near, which helps to assign the unlabeled examples to the right clusters.

2.3 Line search

For the calculation of α_k, we define a fixed stepsize. To do this, we start by calculating the Hessian matrix, which represents the second-order partial derivatives of the objective function f. The elements of this matrix are calculated as

$$Q = \nabla^2_{y^p} f(y^j) = \begin{cases} 2\sum_{i=1}^{l} w_{ij} + 2\sum_{i=1}^{u} \overline{w}_{ij} - 2\overline{w}_{jj} & \text{if } p = j, \\ -2\overline{w}_{ij} & \text{if } p = i \end{cases}$$

In this case, the Hessian Q is symmetric as all the second partial derivatives are continuous. Moreover, the matrix is positive definite, thus ensuring that f is strictly convex. With these two conditions satisfied, we can calculate the Lipschitz constant for the Gradient Descent algorithm, which is given by,

$$L = \lambda_{\max}(Q)$$

where $\lambda_{\max}(Q)$ is the largest eigenvalue of Q. The stepsize α_k is then given by

$$\alpha_k = \frac{1}{L}$$

This ensures that the stepsize is small enough to guarantee convergence of the algorithm. In the case of the Block Coordinate Gradient Descent schemes, the Lipschitz constant is computed for every column j of Q as

$$L_j = \|Q_j\|$$

and chosen during each iteration of the algorithm according to the selected block (of dimension 1 for this homework). Then, the stepsize α_k is calculated as

$$\alpha_k = \frac{1}{L_j}$$

2.4 Stopping rule

A stopping rule is a criterion that tells us when we have reached a satisfactory solution, or when further iterations would not improve the solution significantly. In this homework, we implemented two stopping rules: one based on the number of iterations, and the other based on the convergence of the solution. The latter rule in all three algorithms implemented is given by

$$\|\nabla f(x_k)\|_2 \leq \epsilon$$

with $\epsilon > 0$ sufficiently small. In our case, we chose $\epsilon = 10^{-5}$

Moreover, a stopping rule that is independent of the total gradient can be beneficial for the BCGD with randomized rule. In particular, if we have

$$\|\nabla f_i(x_k)\|_2 \leq \epsilon$$

then,

$$\|\nabla f(x_k)\|_2 \leq n\epsilon$$

Therefore, we can set ϵ/n as stopping condition for the maximum slope along the coordinate axes, where n is the numbers of variables. We included this condition stop in the BCGD randomized, but the algorithm did not trigger it as an early stop.

2.5 Incremental updates

To practically implement the BCGD methods, we use incremental updates for the loss function and its gradient. Thus, in order to avoid the matrix vector multiplication $\nabla f(x_k) = Qx_k + b$ which leads to a cost of $O(n^2)$, it is possible to update recursively $f(x_k)$ and $\nabla f(x_k)$ to a cost of $O(n)$ as

$$f(x_{k+1}) = f(x_k) - \alpha_k |\nabla_{i(k)} f(x_k)|^2 + 0.5\alpha_k^2 Q_{i(k)i(k)} |\nabla_{i(k)} f(x_k)|^2$$

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

Additionally, this calculation allows us to effectively use the stopping rule based on the Gradient Descent.

2.6 BCGD Gauss-Southwell block selection

In each iteration BCGD Gauss-Southwell we will update the gradient in an efficient way explained in Section 2.5, picking the block i of size 1 as follows:

$$i(k) \in \arg\max_{i \in [0:n]} |\nabla_i f(x_k)|$$

2.7 Accuracy measure

To evaluate the accuracy of our model, we used two metrics: one based on the continuous values of y^j and the another one based on the max loss function. Regarding the first metric, since the labels for the unlabeled examples are continuous values, we can use them to calculate a "continuous" accuracy at each iteration as

$$accuracy = 1 - \frac{|y^j_{true} - y^j|}{2u}$$

One limitation of this rule is that it assumes we know the correct labels of the unlabeled examples, which is not the case in practice. This rule is only useful for evaluating the performance of different methods on a given dataset, but it does not reflect the reality of working with unlabeled data.

For the second metric, we want to find the worst possible outcome of our model. To do this, we use a loss function f. We assume that all the unlabeled examples belong to one class and all the labeled examples belong to the other class. Then, we find the maximum value of f over all possible predictions using

$$f_{max\_loss} = 4\sum_{i=1}^{l}\sum_{j=1}^{u} w_{ij} + 2\sum_{i=1}^{u}\sum_{j=1}^{u} \overline{w}_{ij}$$

which represents an upper bound for the highest loss that our model can make under this scenario. Then, the accuracy is calculated as

$$accuracy = 1 - \frac{f(y^j)}{f_{max\_loss}}$$

One limitation of this accuracy measure arises from the fact that the loss function f never reaches zero, even when all the unlabeled examples are correctly classified and have converged. Therefore, the accuracy can never be 1, which may not reflect the true performance of the model. However, this metric does not need the true labels of the unlabeled examples, which represents a more realistic scenario.

Another option to calculate the accuracy is to apply the *accuracy_score* function from scikit-learn library, which requires a function that rounds the predicted value to {-1, 1} based on a certain threshold. This approach was also implemented, but it is not shown in this document for simplicity reasons.

3 Results

3.1 Synthetic Data Set

3.1.1 Description

To evaluate the performance of the different approaches we used, we generated a simulated data set that mimicked the characteristics of real data. We applied each method to the synthetic data and compared the results. For this purpose, we used the function *make_blobs* from the scikit-learn library, resulting in the distribution of points shown in Figure 1. The data set is composed by 10,000 samples, of which 7,000 correspond to unlabeled points and the rest are the labeled ones belonging to the classes {-1, 1}.

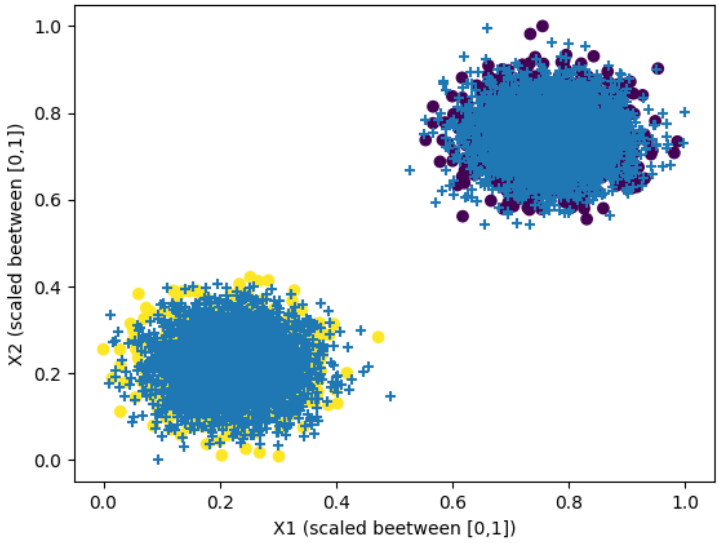

Figure 1: Distribution of examples for the synthetic data set before appliying optimization algorithms. Symbol + denotes unlabeled data, ● denotes labeled data (two classes: yellow and purple.)

In Figure 2, a comparison between similarity values as a function of the distance for the two measure functions presented in Section 2.2 is shown. When the data points are very close to each other, the Euclidean distance metric may not capture the true similarity between them. This led to convergence issues, where the algorithms did not reach the desired values of {-1, 1}. Therefore, we decided to use the similarity measure based on the exponential distance between examples for computing the weights. This function offers a better behaviour when the centers of the clusters are close to each other. Specifically, we use a scaling factor equal to 15.

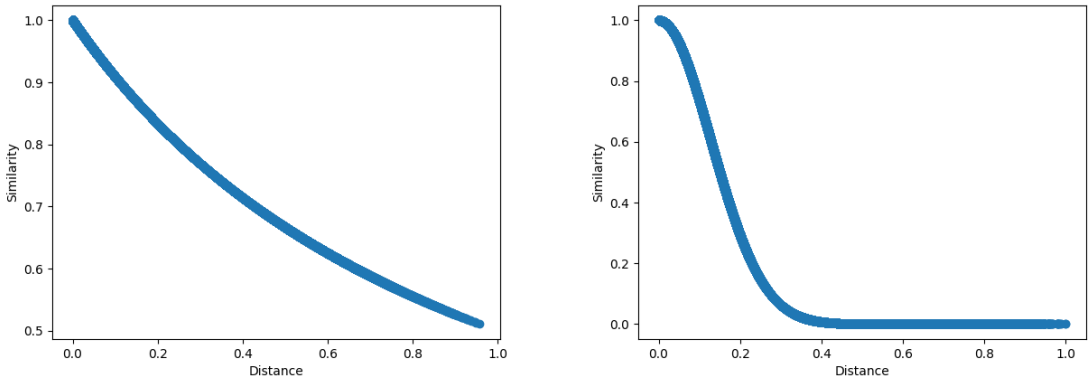

Figure 2: Similarity measure between examples based on Euclidean (left) and exponential (right) distances.

3.1.2 Results

The results for each method are reported in Table 1. As it can be seen graphically in Figure 3, all the unlabeled examples have converged and are correctly classified.

Table 1: Summary of results for the Synthetic Data Set

Method	Early Stop	Iterations	Time (ms)	Accuracy (Max Loss)	Accuracy (True Labels)	Final Loss
Gradient Descent	Yes	69	196,316	0.99999	0.99996	71,549.98
BCGD randomized rule	Yes	353,257	379,456	0.99999	0.99996	71,549.98
BCGD GS rule	Yes	176,068	187,805	0.99999	0.99996	71,549.98

The convergence behavior of different algorithms is illustrated in Figure 4, which plots the accuracy value versus the iteration number. It is evident that the Gradient Descent algorithm converges faster than the other two methods, requiring fewer iterations to reach a low value of the objective function. The BCGD Gauss-Southwell method follows next, while the BCGD Randomized method exhibits the slowest convergence rate. Nonetheless, the gradient calculation step in Gradient Descent method is costly, and the randomized BCGD is less effective in choosing the direction that minimizes the objective function. This can be seen in Figure 5, which shows that the BCGD Gauss-Southwell requires less time to converge, followed by the Gradient Descent and BCGD Randomized by a great difference.

The results for the loss function are also reported in Figure 6, following the same behaviour as the accuracy for the different methods implemented.

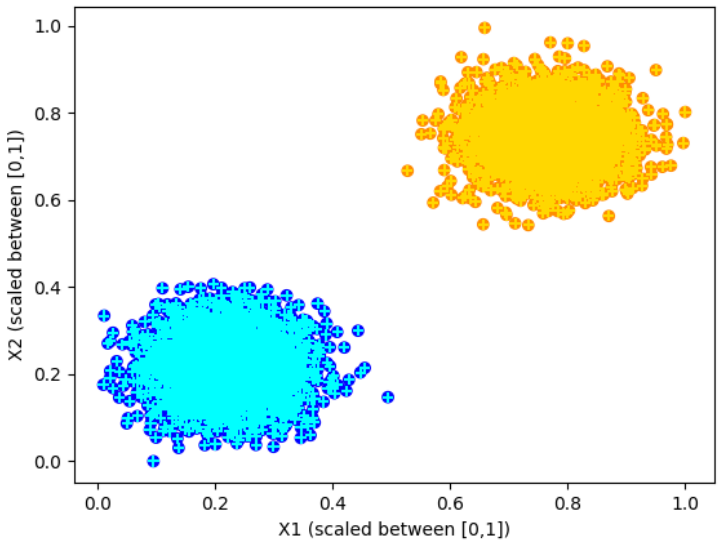

Figure 3: Distribution of examples for the synthetic data set after applying Gradient Descent algorithm. Symbol + denotes the predicted unlabeled data (now labeled with a class), ● denotes the true unlabeled data.

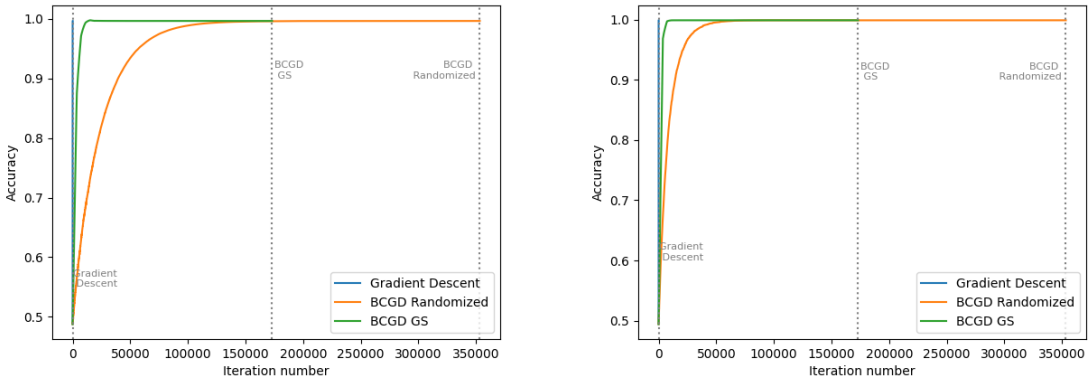

Figure 4: Accuracy as a function of iteration number based on continuous values (left) and on loss function (right).

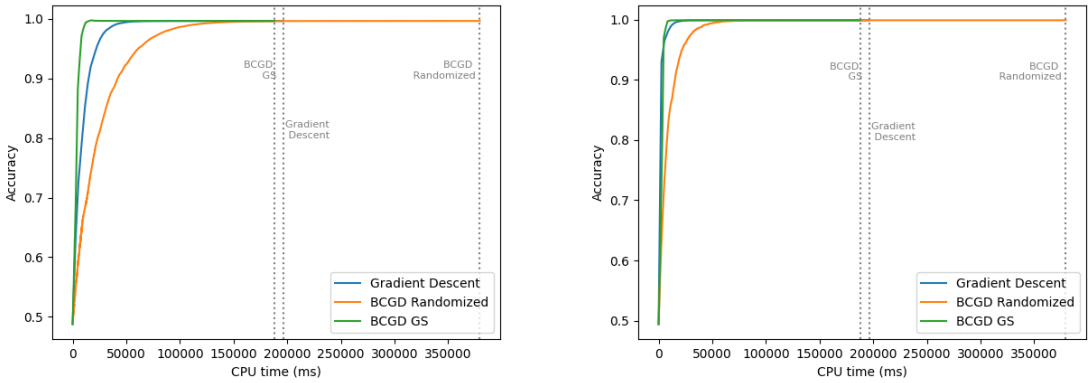

Figure 5: Accuracy as a function of CPU time based on continuous values (left) and on loss function (right).

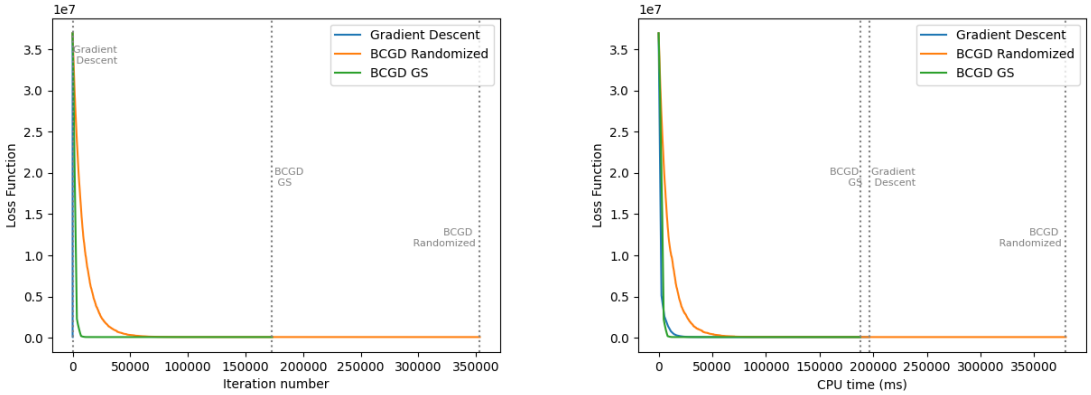

Figure 6: Loss as a function of iteration number (left) and as a function of CPU time (right).

3.2 Real Data Set

3.2.1 Description

Rice seeds classification is a task that aims to identify and separate different varieties of rice seeds based on their morphological features. This task is important for quality control, breeding, and genetic studies of rice. Automatic rice seeds classification can reduce the cost and increase the accuracy and efficiency of the task. In this context,the Rice Type Data Set was chosen to apply the algorithms for classifying rice varieties. This data set contains numerical attributes of rice grains, such as area, perimeter, major and minor axis length, eccentricity, convex area and extent, and a binary class: Jasmine (1), Gonen (0). The dataset consists of 18,185 data points, with 9,985 belonging to class 1 and 8,200 belonging to class 0. In particular, we selected eccentricity and perimeter as the features. To handle the large size of the dataset, we randomly select a subset of 10%, which gives us 1,818 observations. Out of these, 985 are in class 1 and 833 are in class 0, and used 70% of the total points as unlabeled examples. Figure 7 illustrates how the unlabeled and labeled data were distributed. We can notice that the purple dots (Gonen rice), have larger values of eccentricity and perimiter compared to the Jasmine rice.

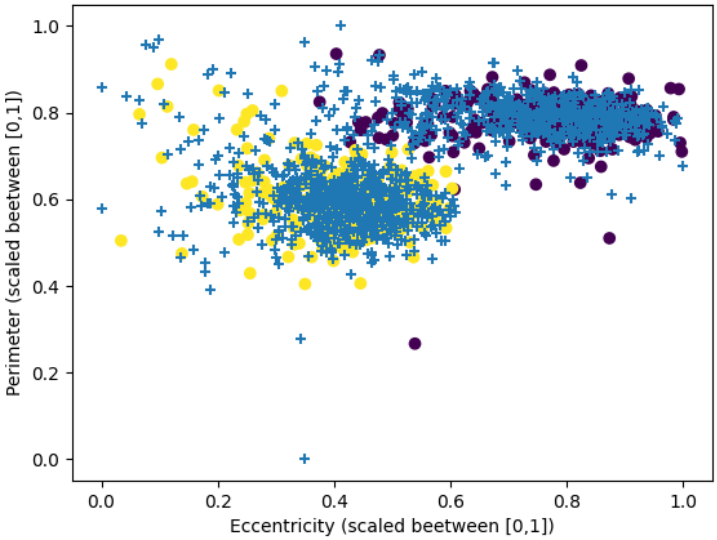

Figure 7: Distribution of examples for the real data set before applying optimization algorithms. Symbol + denotes unlabeled data, ● denotes labeled data (two classes: yellow and purple).

3.2.2 Results

The results for each method are reported in Table 2. As it can be seen graphically in Figure 8, unlike the synthetic data case, not all the examples are correctly classified. This is to be expected since there is some overlapping between the clusters, difficulting the clasification for the method.

Table 2: Summary of results for the Real Data Set

Method	Early Stop	Iterations	Time	Accuracy (Max Loss)	Accuracy (True Labels)	Final Loss
Gradient Descent	Yes	1,469	197,298	0.92186	0.70815	197,548.60
BCGD randomized rule	Yes	58,342	13,585	0.92186	0.70815	197,558.88
BCGD GS rule	Yes	29,949	7,216	0.92186	0.70815	197,548.60

The Figure 9 shows how the accuracy value changes with the number of iterations. Gradient Descent converges faster than BCGD randomized, as seen in the synthetic data set results. However, gradient descent is more expensive per iteration, because it computes the full gradient every time. In Figure 10, we see that BCGD randomized and with Gauss-Southwell rule are faster than Gradient Descent in terms of time.

The same figures show that the accuracy based on the maximum loss achieves nearly a 90% performance, while the accuracy based on the real values of the unlabeled data attains around 70%. The latter value, which may be a better way to assess the accuracy, is influenced by the fact that some examples are near the edges of the clusters, which makes it hard to classify them correctly into either cluster because they have some overlap, as seen in Figure 8. Another explantion could be the sparsity of the blue points, Jasmine rice has a wider range of values for both futures.

One possible explanation for the strange behavior of BCGD GS at the beginning of the accuracy based on continouos labels is that the values that it assigns to the unlabeled data are still fluctuating to assign the final label. This causes accuracy value to fluctuate. However, as the number of iterations increases, the values become more stable and converge to either -1 or 1, indicating the label assignment.

It is important to highlight that BCGD randomized is expected to be faster than Gradient Descent as the number of variables grows. However, this was not the case when we compared the synthetic data (7,000 unlabeled variables) with the real data set (1,272 unlabeled variables). Thus, we tested the synthetic data set for 2,000 data points. In this scenario, both methods took similar time to converge. We propose three possible explanations for why Gradient Descent performed faster in the synthetic dataset. First, the classes of the points were more separated, making the weighting matrix sparse. Second, the BCGD randomized was selecting blocks randomly and not based on the loss function

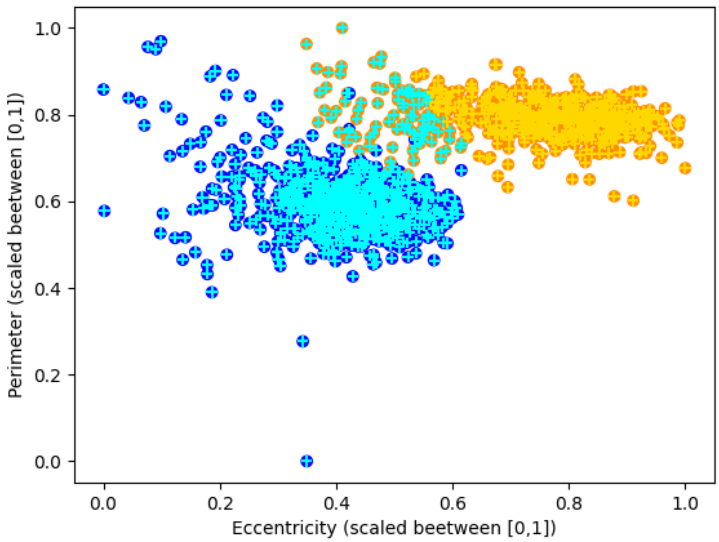

Figure 8: Distribution of examples for the real data set after applying Gradient Descent algorithm. Symbol + denotes the predicted unlabeled data (now labeled with a class), ● denotes the true unlabeled data.

reduction (non-uniform sampling could be used). Third, there could be better ways to implement a step size for the BCGD, that would accelerate the convergence of the algorithm.

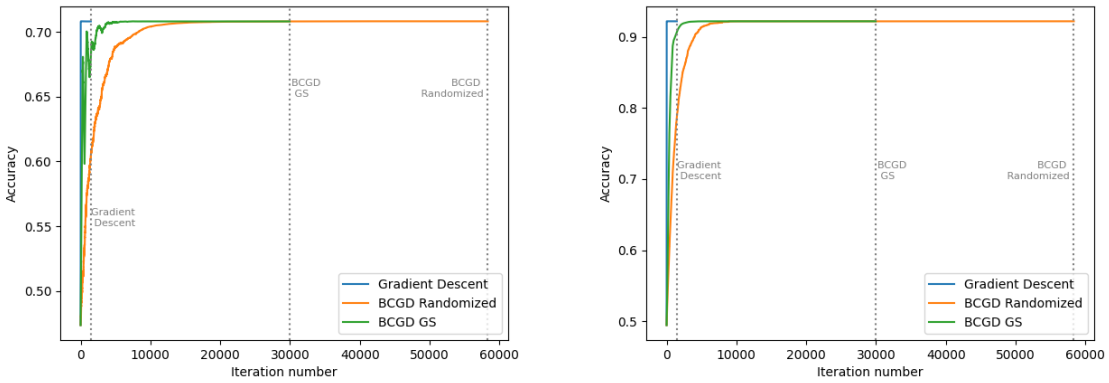

Figure 9: Accuracy as a function of iteration number based on continuous values (left) and on loss function (right).

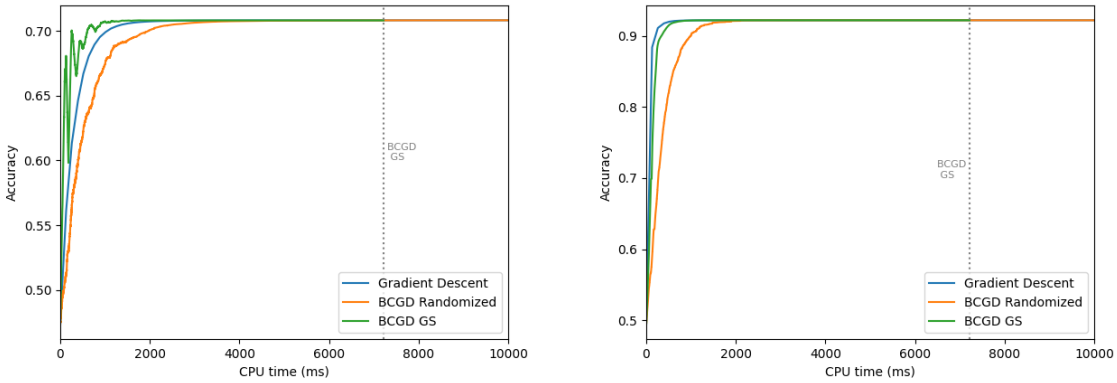

Figure 10: Accuracy as a function of CPU time based on continuous values (left) and on loss function (right) with a time window of 10,000 ms.

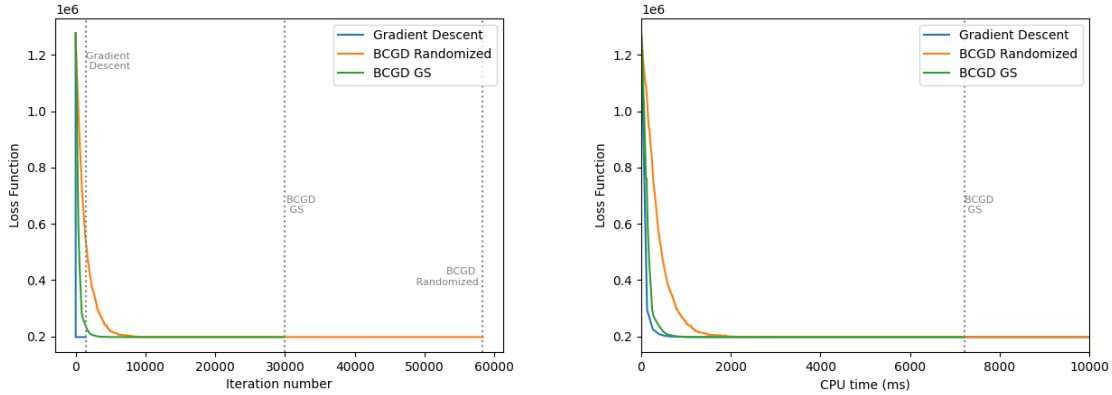

Figure 11: Loss as a function of iteration number (left) and as a function of CPU time (right) with a time window of 10,000 ms.

4 Conclusions

This report presents three methods to optimize a semi-supervised learning problem. We first discussed how choosing a suitable similarity function is crucial for this problem. We found that the exponential distance is more suitable, as it better reflects the true similarity when the points are close to each other. We then compared the different methods and observed that the Gradient Descent method converges faster than BCGD methods in terms of iterations, but takes more time. BCGD with Gauss-Southwell rule is the fastest among the three methods, mainly due to its block selection rule and the incremental calculation of the gradient. Accuracy based on true labels is important because it helps us to evaluate our algorithm's performance, but in real semi-supervised problems we will not have access to the true labels. Therefore, accuracy based on loss can be used as an approximation of how close we are to an upperbound loss. In particular, using only two features, eccentricity and perimeter, the real data set achieves an accuracy of 70%.