Univerità degli Studi di Padova
15/05/2023

Gradient Descent and BCGD Methods

Giovanni Piva, Roberto Vicentini

Data Science

Univerità degli Studi di Padova
15/05/2023

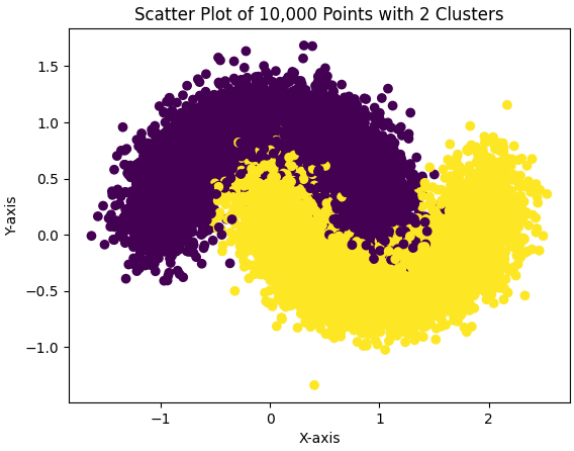
Abstract- In this report are going to analyze a common semi-supervised learning problem. The problem arises when we have a limited number of labeled data samples and many unlabeled data samples. The main objective is to determine the missing labels for the unlabeled data. To accomplish this, we will code in Python Gradient Descent techniques and BCGD methods to minimize a loss function that will be presented later.

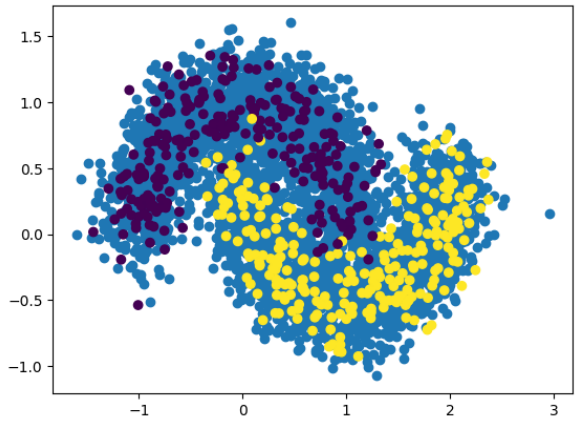
I. Introduction

In this analysis, we will adopt a two-step approach. Firstly, we will utilize a model with synthetic data to conduct our initial investigations. Once we achieve favorable outcomes, we will then proceed to acquire and employ real-world data. We will present the methodology employed for generating and manipulating synthetic data to achieve our objectives. Subsequently, we will showcase the obtained results and delve into the exploration of the real-world dataset, followed by a thorough discussion of the outcomes.

II. Synthetic Dataset

In order to generate 10000 synthetic points divided in 2 clusters, we are going to use **sklearn.datasets.make_moons**. This is a simple toy dataset to visualize clustering and classification algorithms. In origin we tried to generate an easiest dataset using **sklearn.datasets.make_blobs**, but we decided to change it in order to have a more variable points generation. The following is a visual representation of the generated points.





The first image represents the generated points divided into the two clusters (*binary class problem*). The second image shows how the data are divided into **labeled** (10% of the dataset) and **unlabeled** (90% of the dataset) points. This is needed to compute (*similarity*) weights between point, that will be used in our loss function later.

III. Weights

As said before, our loss function will need to include weights between points. We are interested in weights between:

- **Labeled** and **Unlabeled** points
- **Unlabeled** and **Unlabeled** points

There are many different methods to compute such weights, and there is no “best method” since it really depends on what data we are working with. After trying different methods, we decided to use weights based on the Euclidian distance between points. The Euclidean distance is bigger when two points are less similar, but a similarity wight needs to be bigger when two points are more similar. To achieve this, we used this formula that is inversely proportional to the Euclidian distance:

$$weight(a,b) = \frac{1}{1+||a-b||}$$

Once we have computed the two weights matrices, we can proceed with the loss function computation.

IV. Loss Function

The loss function we are going to work with was given a priori and it is defined as follows:

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

Let's investigate what is the meaning of each term in this function:

- y^j : is the label currently assigned to the j-th *unlabeled* point.
- $\bar{y}^i$: is the label of the j-th *labeled* point.
- w_{ij} : is the precomputed weight between the i-th *labeled* and j-th *unlabeled* point.
- $\bar{w}_{ij}$: is the precomputed weight between the i-th *unlabeled* and j-th *unlabeled* point.

Now is clear that the similarity method is fundamental for the loss function, so if the similarity method well represent our data comparison, we will have a better representing loss function for our problem.

V. Gradient

Now that we know which is our loss function, we can proceed computing the gradient. The gradient of our loss function has the following formula:

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

The highlighted version of the gradient is what we are interested in. We can see that there are terms that can be computed once a-priori since they are independent form the y^j label (that is our variable).

VI. Accuracy

To analyze, visualize and compare the behavior of the different methods we are going to use, we need to define an *accuracy function*. It's not easy to choose the optimal function for a given problem. Different functions were tested, but since we had a synthetic dataset and a real-world one, we opted fir the *rounding method*. This method consists in *rounding* the y^j label in this way:

- If $y^j >= 0$ then $\mathbf{y^j = 1}$
- Else $\mathbf{y^j = -1}$

It's important to underline that the *rounding method* involves a fast convergence of accuracy. This means that the accuracy output will be the same after few steps. But its ok since in this case we are interested in the convergence values rather than how fast it converges.

VII. Hessian, Lipschitz constant and σ

In order to obtain convergence, is useful to use the Lipschitz constant in our *step-size*.

First we need to compute the *Hessian* matrix, more in particular we need to obtain the maximum eigenvalue of the Hessian matrix, since it is constant. That being said, for our loss function we need to compute:

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

The Hessian matrix will be useful to compute **σ** too, in this way we can determine if our problem is strongly convex or not.

VIII. METHODS

We decided to use 4 different methods, in particular 2 for the **Gradient Descent** and 2 for the Block Coordinate Gradient Descent (**BCGD**):

- **Standard Gradient Descent:** this is the easiest Gradient Descent version. Note that uses the Lipschitz constant for the step-size.

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

- **Gradient Descent Improved Rate:** this method takes advantage of the strong convexity in our problem. In fact it can be used only if we have a strong convexity situation. This time the step-size will depend from σ , resulting in a larger step-size.

$$y_k = y_{k-1} - \frac{2}{\sigma + L}\nabla f(y_{k-1})$$

- **BCGD Random:** in this algorithm, the update is calculated using the gradient formula, which is evaluated on a single instance, denoted as y^j.

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

- **BCGD Gauss-Southwell:** This method is an extension of the BCGD algorithm that incorporates the Gauss-Southwell rule for updating the blocks. In this method, at each iteration, the block with the highest gradient magnitude is selected, and its parameters are updated using the gradient descent method. This approach allows the algorithm to take advantage of first-order information when updating the blocks.

$$j = \underset{i \in [1,u]}{\text{Argmax}} ||\nabla_i f(y)||$$
$$y_k^j = y_{k-1}^j - \frac{1}{L}\nabla_{y^j} f(y_{k-1}^j)$$

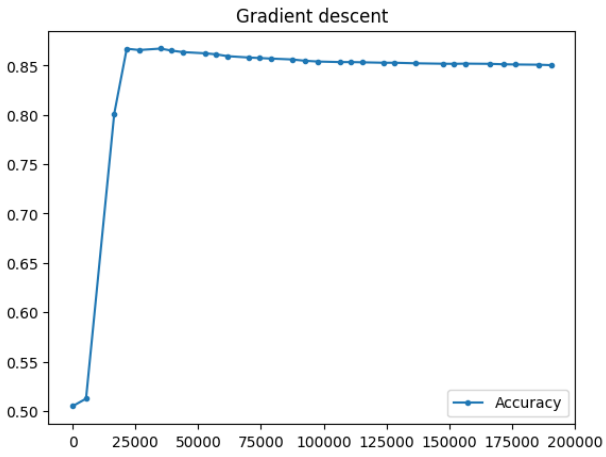
We can finally proceed with the computation and output analysis with a Jupyter Notebook file. The following are the results.

IX. Results and comparison

IX.I Synthetic Dataset

First, we are going to display the results obtained with the *synthetic* dataset. The following are the results with the methods described before. We are going to compare **Accuracy** over **CPU time** usage and **Loss** over **Time** for every algorithm.

Accurancies



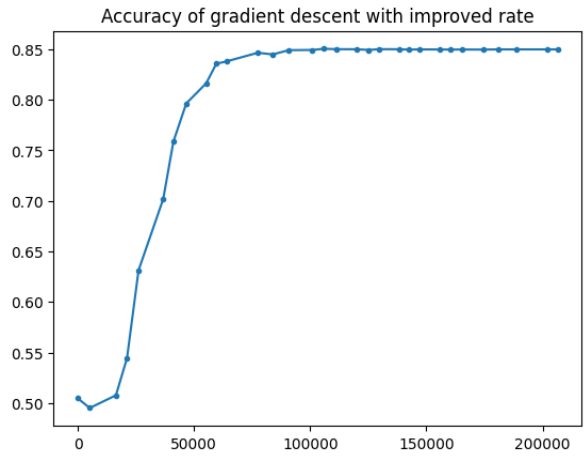
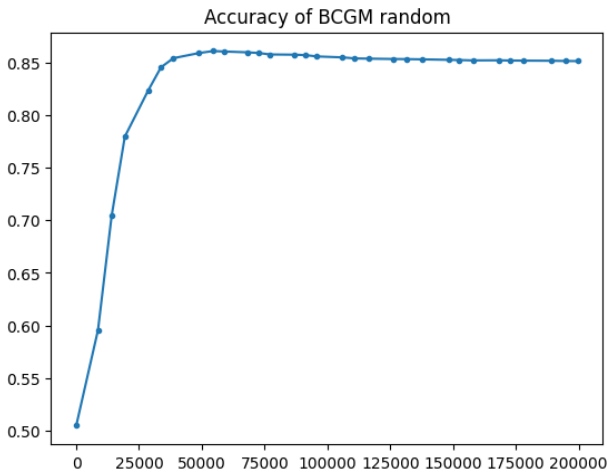


Fig 1 and 2: Gradient Descent Methods



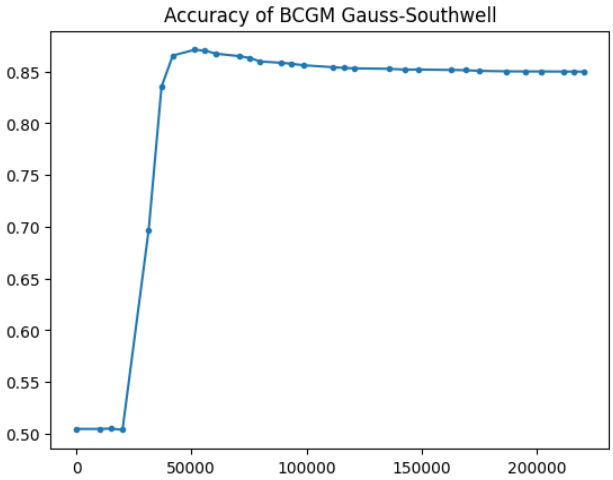


Fig 2 and 3: BCGD Methods

Loss

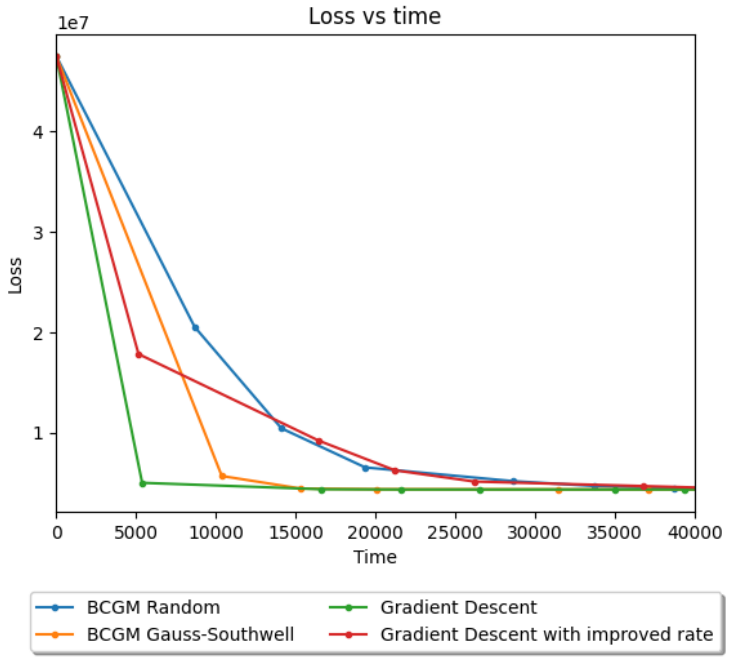


Fig 5: Loss of every method

We can observe that Gradient Descent and BCGM Random perform better both in term of *accuracy* and *loss.*

Gradient Descent is a widely used optimization algorithm known for its simplicity and efficiency. It iteratively updates the model parameters by taking steps proportional to the negative gradient of the loss function. This approach allows Gradient Descent to converge quickly and efficiently, resulting in faster convergence and a rapid decrease in the loss function.

Similarly, BCGM Random utilizes a randomized approach in the optimization process. By incorporating randomness into the selection of search directions, it explores the parameter space more diversely, potentially leading to faster convergence. This randomness enables BCGM Random to escape local minima and find better solutions, allowing for a quicker decrease in the loss function compared to other methods.

On the other hand, Gradient Descent with improved rate and BCGM Gauss-Southwell may have slower convergence and a less rapid decrease in the loss function due to their specific characteristics. Gradient Descent with improved rate adjusts the learning rate, which can be beneficial for avoiding oscillations and overshooting. However, this adaptation may introduce additional complexity, resulting in slower convergence. BCGM Gauss-Southwell, while potentially more robust, may require more precise settings to have better performances due to its specific optimization techniques.

In summary, the superior performance of Gradient Descent and BCGM Random in terms of convergence and loss function improvement can be attributed to their simplicity, efficiency, and the exploration enabled by their respective optimization approaches. These findings highlight the importance of selecting the appropriate optimization algorithm based on the specific problem and desired trade-offs between convergence speed and computational complexity.

IX.II Real-world Dataset

For the real world dataset after a search on Kaggle we found an interesting one to predict if in case of emergency for a specific disease, after the surgery, you're going to run into In-Hospital complications.

The dataset has 24 dimension so we choose to reduce the dataset dimensionality using Principal Component Analysis, in particular Surgical dataset passes from 24 to 5 dimension.

We plot the distribution over the variable we want to predict for verify we have enough data:

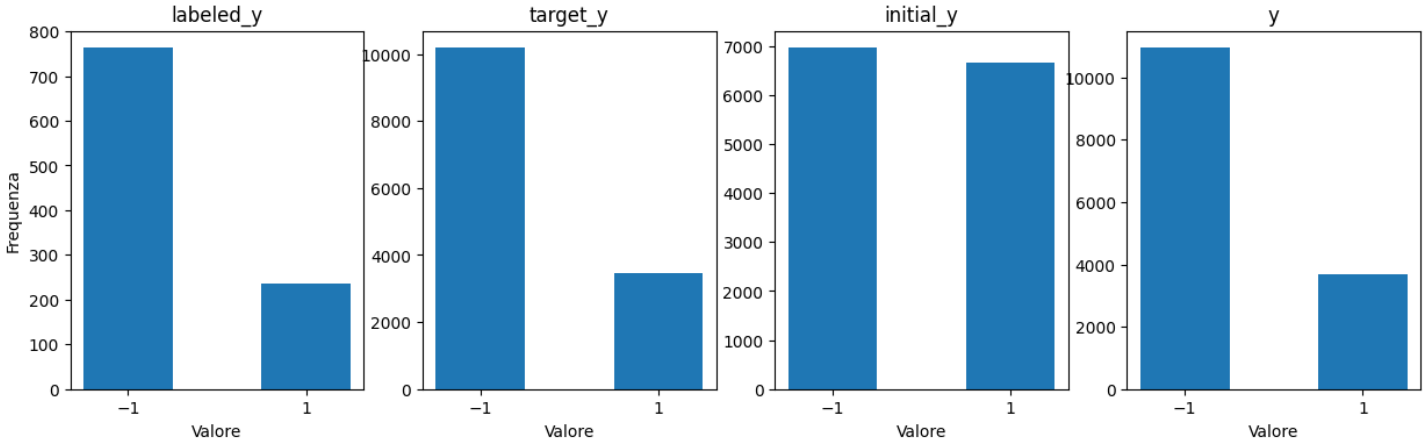


Fig 6: Values Distribution

As we can see we have about 30% of points with label 1 (no In-Hospital complication) and 70% of points with label -1 (In-Hospital complication) in every subset so we have enough data for each class to make a prediction.

We now are display the results obtained with the this dataset. As before, the following are the results with the methods described before. We are going to compare **Accuracy** over **CPU time** usage and **Loss** over **Time** for every algorithm.

Accurancies

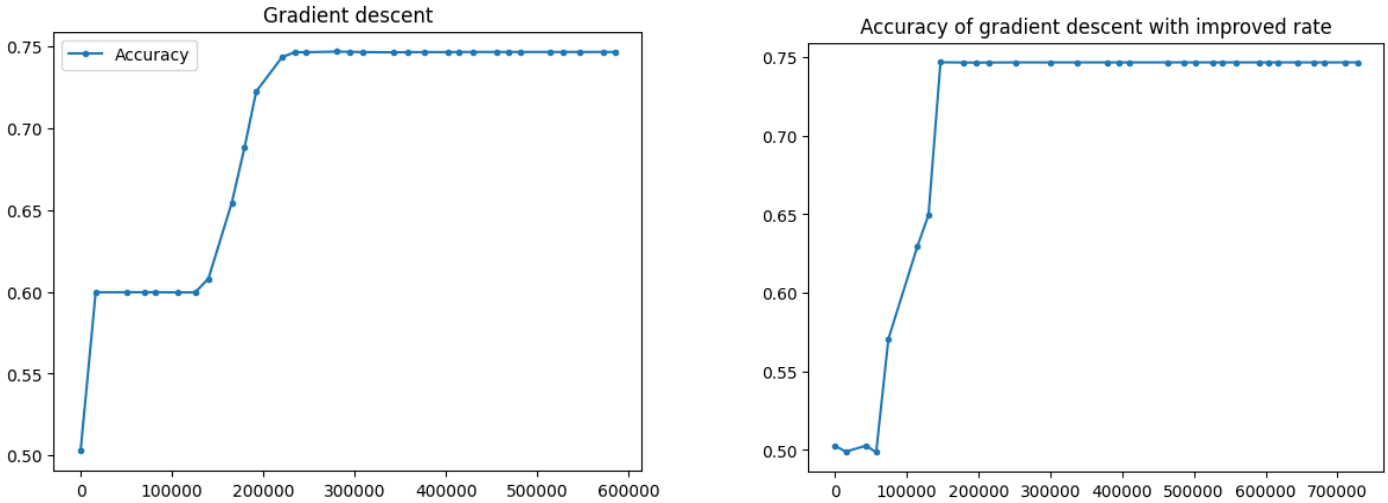


Fig 7 and 8: Gradient Descent Methods

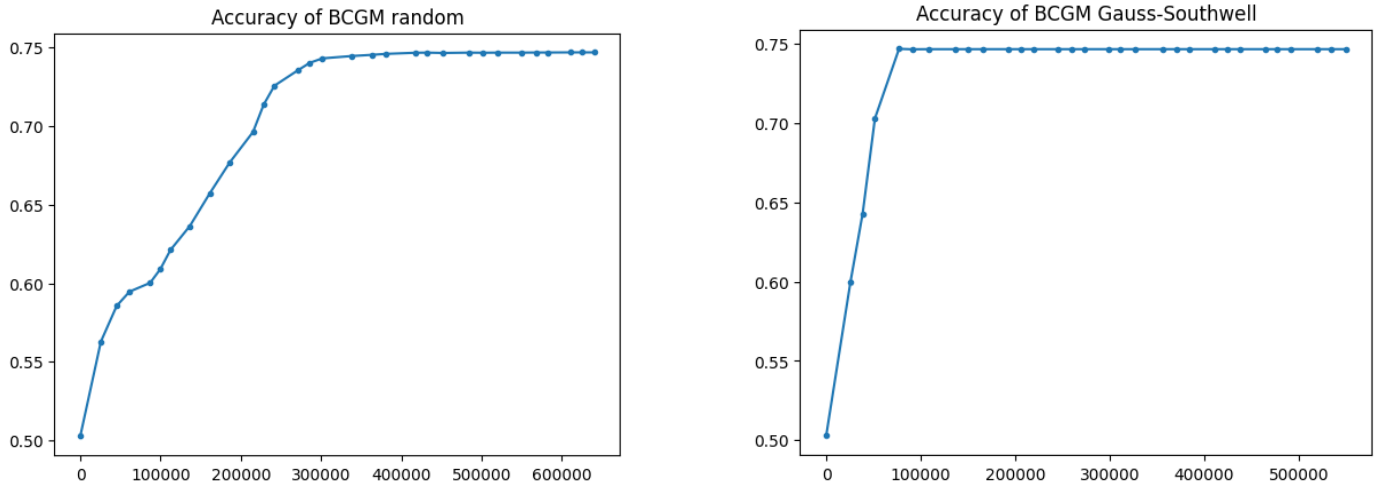


Fig 9 and 10: BCGM Methods

LOSS

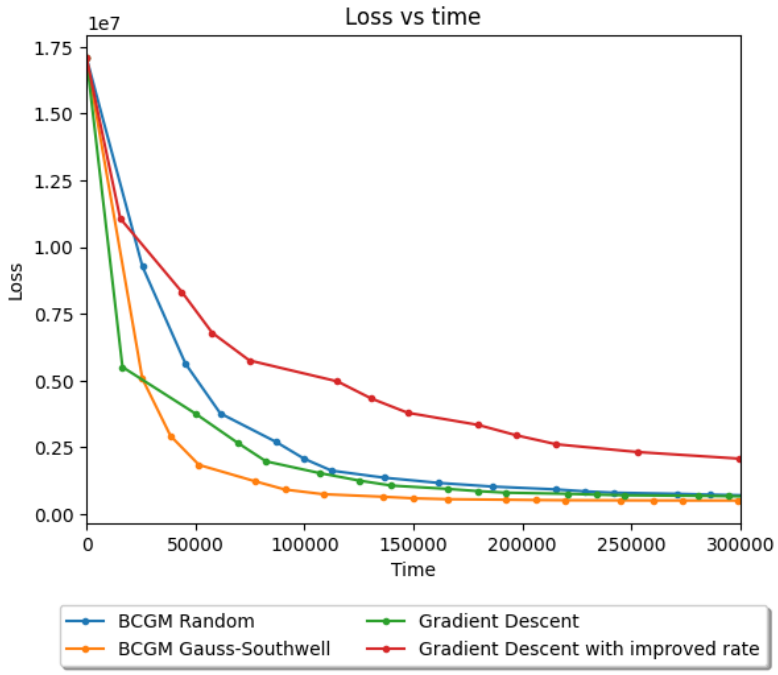


Fig 11: Loss of every method

It is worth mentioning that the longer runtime observed in this study can be attributed to the higher dimensionality of the data. Despite applying PCA to reduce dimensionality, the dataset still retained five dimensions. The increased dimensionality poses computational challenges and requires additional processing time for the optimization algorithms.

However, even with this added complexity, the optimization algorithms demonstrated impressive performance on synthetic data, achieving comparable results in terms of accuracy and loss optimization. However, it is important to note that the Gaussian-Southwell variant of the BCGM algorithm outperformed the others in accuracy, while BCGM Random and Gradient Descent remained the top performers in loss optimization. These findings highlight the algorithms' versatility and effectiveness in handling synthetic data, paving the way for further exploration in real-world scenarios.