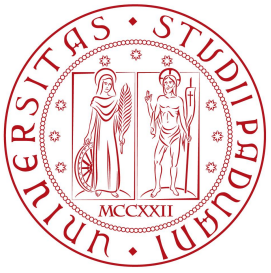
University of Padova
Department of Mathematics

Master Degree in Data Science



Optimization for Data Science: Homework 1

Gioele Ceccon, ID: 2079425
Gaia Nardella, ID: 2091413
Pietro Renna, ID: 2089068
Valerio Rocca, ID: 2094861

Accademic Year 2022/2023

Contents

Chapter 1

Introduction

1.1 Semi-Supervised Learning

Semi-Supervised Learning is a Machine Learning paradigm where the available data comprises labeled and unlabeled examples. This can be formalized as follows:

- Labeled examples: $(\bar{x}_i, \bar{y}_i), i = 1, ..., l$;
- Unlabeled examples: $(x_j, y_j), j = 1, ..., u$
  where the y_j labels are unavailable.

The goal is to find values for the y_j labels.

1.2 Optimization Problem

The optimization problem consists in minimizing the given **objective function**:

$$\min_{y \in R} \sum_{i=1}^{l} \sum_{j=1}^{u} w_{ij}(y_j - \bar{y}_i)^2 + \frac{1}{2} \sum_{i=1}^{u} \sum_{j=1}^{u} \bar{w_{ij}}(y_i - y_j)^2 \qquad (1.1)$$

where:

- weights w_{ij}: similarity between labeled example i and unlabeled example j
- weights $\bar{w}_{ij}$: similarity between unlabeled example i and unlabeled example j

1.3 Overview

To analyze the proposed solutions, they were tested on both a synthetic and a real dataset. The first one is composed of points randomly generated in a two-dimensional space. Both these sets consisted of l labeled examples and u unlabeled ones, where the number of labeled examples l was significantly smaller than the number of unlabeled ones u. Next, three optimization algorithms were developed from scratch, including Classic Gradient Descent, Randomized Block Coordinate Gradient Descent, and Gauss-Southwell Block Coordinate Gradient Descent. These algorithms were evaluated for their performance on both synthetic and real semi-supervised learning tasks.

Chapter 2

Simulated data

2.1 Point Generations

The dataset in total consists of 8000 points. To generate them, 2000 points were sampled from each of the four Bivariate Gaussian Distributions having, respectively, the following parameters:

- $\mu_1 = \begin{bmatrix} -2 & -5 \end{bmatrix}$, $\sigma_1 = \begin{bmatrix} 7 & 0 \\ 0 & 0.5 \end{bmatrix}$
- $\mu_2 = \begin{bmatrix} 2 & -1 \end{bmatrix}$, $\sigma_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 4 \end{bmatrix}$
- $\mu_3 = \begin{bmatrix} 2 & 5 \end{bmatrix}$, $\sigma_3 = \begin{bmatrix} 7 & 0 \\ 0 & 0.5 \end{bmatrix}$
- $\mu_4 = \begin{bmatrix} -2 & 2 \end{bmatrix}$, $\sigma_4 = \begin{bmatrix} 0.5 & 0 \\ 0 & 4 \end{bmatrix}$

First, samples drawn from $N(\mu_1, \sigma_1)$ and $N(\mu_2, \sigma_2)$ were grouped into class 1, while samples drawn from $N(\mu_3, \sigma_3)$ and $N(\mu_4, \sigma_4)$ were grouped into class -1. Then, 95% of the points in each class were labeled as 0 to indicate that they are unlabeled. As a result, a total of 5% of the points were assigned labels as shown in Figure 2.1 .

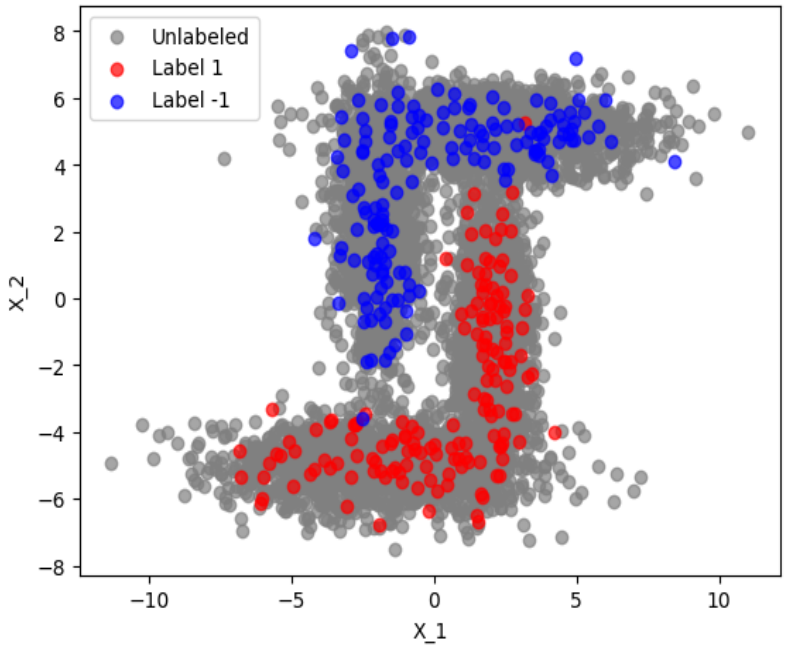


Figure 2.1: Generated points

2.2 Weight Initialization

Weights w_{ij} and $\bar{w}_{ij}$ were obtained using the similarity measure between examples i and j as defined in Section 2.2.1.

2.2.1 Squared Laplacian Kernel

In light of this problem, the nearby points carry more information for determining the class of an unlabeled point than those that are farther away. To address this, a similarity function was chosen that incentivizes the proximity of points and penalizes their distance. The adopted function is he Squared Laplacian Kernel, defined as:

$$K(y, y') = exp\left\{-\frac{\|y-y'\|_2^2}{\sigma}\right\}$$

where:

- $\|y - y'\|_2^2$ is the Euclidean distance between examples y and y';
- σ is a hyperparameter that regulates the smoothness of the distances; after performing a grid search, it was set to 0.2.

2.2.2 Euclidean Distance and Inverse Multiquadratic Kernel

The similarities were computed using the Euclidean Distance and the Inverse Multiquadratic Kernel, which is defined as:

$$K(y, y') = \frac{1}{\sqrt{c^2 + \|y - y'\|_2^2}}$$

The other two measures were not used because of their poor performance over both the synthetic and the real datasets as shown in Figure 2.2.

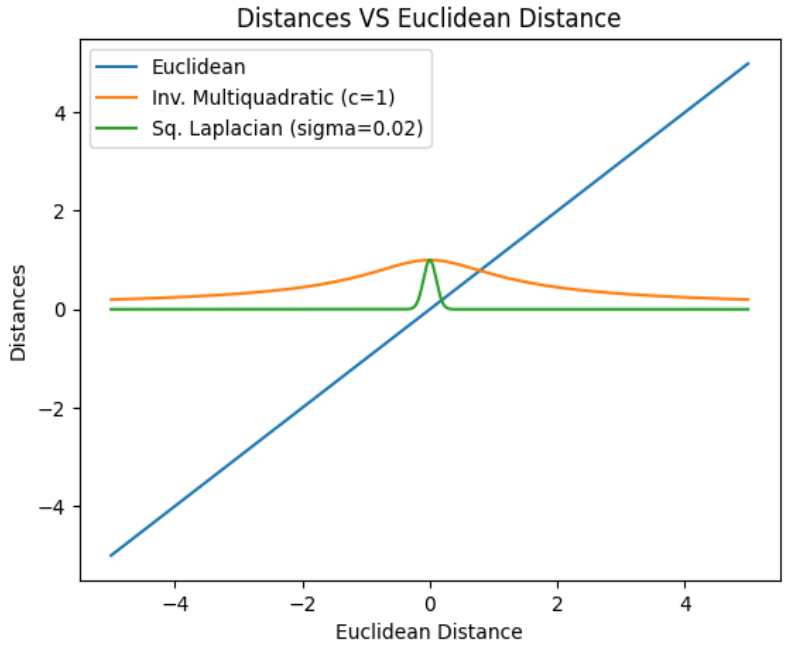


Figure 2.2: Considered distances (y) given the Euclidean distance (x)

Chapter 3

Algorithms

3.1 Fundamental elements

Before presenting the specific algorithms, the shared components used in all the methods will be defined.

3.1.1 Stopping Rule

The algorithms should be executed for a predetermined maximum number of iterations denoted by *iter*. However, in order to prevent the calculation of insignificant updates on the Gradient, the norm of the gradient was assessed to halt the computations based on a given fixed value. The implementation is given in **Algorithm 1**, considering the Gradient in the form $\nabla f(x)$, and a fixed $\epsilon = 10^{-5}$.

Algorithm 1 Stopping Rule

```
Iteration i
norm = ||∇f(x)||^2
if norm < ε then
  print("The norm of the gradient at iteration i is: norm")
  STOP
end if
```

3.1.2 Stepsize

Given that f is twice continuously differentiable and subsequently smooth, the Lipschitz constant was estimated as the largest eigenvalue of the Hessian matrix $\nabla^2 f$. The idea behind this method is that the Lipschitz

constant reflects the function's sensitivity to changes in its input (it's then related to the function's smoothness), while the Hessian matrix provides information about the function's curvature. By using the norm of the matrix (specifically, approximated by its largest eigenvalue), an estimate of the Lipschitz constant can thus be obtained.

Moreover, in this case, the Hessian matrix is composed of the following elements:

- $\nabla_{y_j y_k} f = -2\bar{w}_{kj}$, where $k \neq j$
- $\nabla_{y_j y_j} f = 2\sum_{i=0}^{l} w_{ij} + 2\sum_{i=0}^{u} \bar{w}_{ij} - 2\bar{w}_{jj}$

3.1.3 Label Assignment

Since the algorithms return values in $[0, 1]$, a threshold selector converts the outputs y to -1 or 1.
The function *threshold selector*(y) returns a vector y_{lab}^{out}, which is obtained by applying a thresholding operation to the output y obtained from the algorithms.

$$y_{lab}^{out} = \begin{cases} -1 & \text{if } y < 0.5 \\ +1 & \text{if } y \geq 0.5 \end{cases} \tag{3.1}$$

3.2 Gradient Based Methods

Gradient-based methods are a class of optimization algorithms that iteratively update the variables, minimizing or maximizing the objective function using its gradient.

Gradient of the Objective Function

For the objective function (1.1), the gradient is defined as:

$$\nabla_{y_j} f = 2\sum_{i=1}^{l} w_{ij}(y_j - \bar{y_i}) + \sum_{i=1}^{u} \bar{w}_{ij}(y_j - y_i) \tag{3.2}$$

3.2.1 Gradient Descent

The vector of unknown labels is optimized by moving in the direction of the negative gradient computed for the whole y.

Algorithm 2 Gradient Descent

Choose a vector $y \in R^u$
$y_1 = y$
for $k = 1, ..., iter$ **do**
  if y_k satisfies the stopping rule **then**
    STOP
    return y_k
  end if
  Set $y_{k+1} = y_k - \frac{1}{L}\nabla f(y_k)$, with $L > 0$ Lipschitz Constant
end for
return y_{iter}

3.2.2 Block Coordinate Gradient Descent (BCGD) Methods

BCGD Methods are a sub-family of Gradient Methods capable of efficiently solving large-scale optimization problems. These algorithms use only a subset of variables (blocks) to perform Gradient Updates while keeping the others fixed. The way this subset is selected varies for each BCGD method. From now on, a block of dimension one is considered, which in this context corresponds to one variable.

Random Block Coordinate Gradient Descent

This implementation of BCGD performs the selection of the blocks randomly. At each iteration, a random subset of the variables is selected and the **Algorithm 3** updates only it while keeping the other variables fixed. The randomness could also avoid local optima and speed up the convergence.

Algorithm 3 Random BCGD

Choose a vector $y \in R^u$
$y_1 = y$
for $k = 1, ..., iter$ **do**
  if y_k satisfies the stopping rule **then**
    STOP
    return y_k
  end if
  Randomly pick y_k^j, where $j = 1, ..., u$

  Set $y_{k+1} = y_k - \frac{1}{L}\nabla_j f(y_k)$, with $L > 0$ Lipschit Constant
end for
return y_{iter}

Gauss-Southwell Block Coordinate Gradient Descent

In this implementation of the BCGD, the selected block is the one that minimizes the residual with the current iteration; in simpler terms and focusing on this specific case of study, at each iteration the rule selects the unlabeled unit which is associated with the highest norm of the gradient, i.e.:

$$y_j = argmax \parallel \nabla_i f(x_k) \parallel \forall i \in \{1, ..., u\}$$

Since the classic implementation of the Gauss-Southwell BCGD would be very expensive in terms of computational resources, the provided implementation incorporates a more efficient gradient update method that avoids the computation of the entire gradient at each iteration. In fact, it updates the gradient only with the changes made at the previous iteration, resulting in a faster convergence and reduced computational cost.

Algorithm 4 Gauss-Southwell BCGD

Choose a vector $y \in R^u$
$y_1 = y$
for $k = 1, ..., iter$ **do**
  if y_k satisfies the stopping rule **then**
    STOP
    return y_k
  end if
  Select y_k^j, for which $j = argmax \parallel \nabla_i f(x_k) \parallel$, where $i = 1, ..., u$

  Set $y_{k+1} = y_k - \frac{1}{L}\nabla_j f(y_k)$, with $L > 0$ Lipschitz Constant
end for
return y_{iter}

3.3 Performances on Simulated Data

Looking at Table 3.1, it's easy to notice that the highest score in terms of accuracy is the one relative to the Gradient Descent algorithm. However, the CPU time required for Gradient Descent is significantly longer than the other algorithms. This result does not come unexpected, since this algorithm is the only one computing the full gradient at each iteration. Secondly, Random BCGD had the lowest accuracy, indicating that with the given settings (it was given 100000 iterations) it was less effective at finding the optimal solution than the other algorithms. As expected, it required the shortest CPU time. Furthermore, based on the evidence presented in Figure 3.4, it is likely that with additional iterations, a superior outcome could have been achieved. Finally, Gauss-Southwell BCGD achieved an accuracy lower than the one of the Gradient Descent but higher than the one of the Random BCGD, and the same goes for the CPU time it required. This suggests that Gauss-Southwell BCGD may be a good trade-off between accuracy and computational efficiency, especially for problems of similar complexity to the synthetic data set used in this document.

Algorithm	**Accuracy**	**CPU time**	**Iterations**
Gradient Descent	0.9223	38min 33s	50
Random BCGD	0.7663	10min 11s	100000
Gauss-Southwell BCGD	0.8128	23min 26s	100000

Table 3.1: Performance of the algorithms in the synthetic Data Set

3.3.1 Gradient Descent

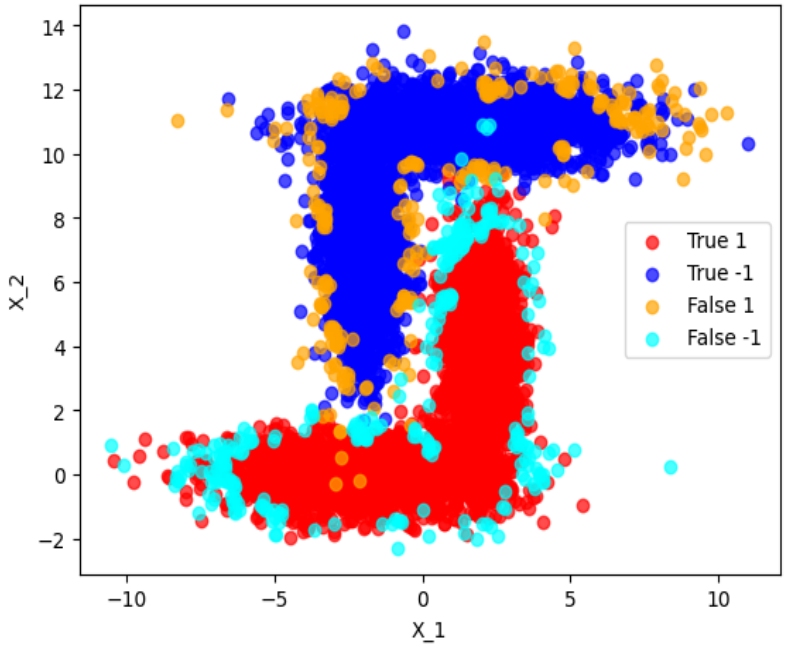


Figure 3.1: Points and their assigned class in Gradient Descent

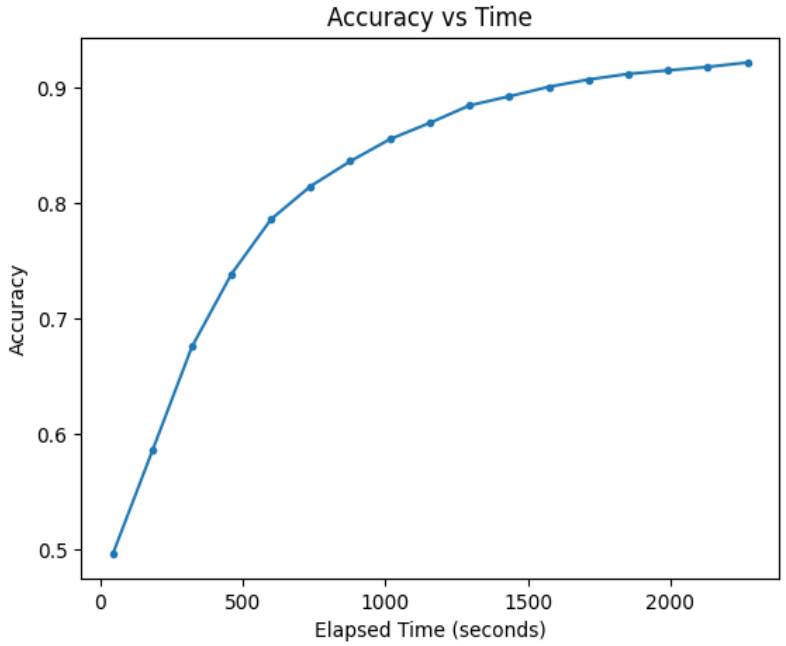


Figure 3.2: Plot of Accuracy vs CPU Time for Gradient Descent

3.3.2 Randomized BCGD

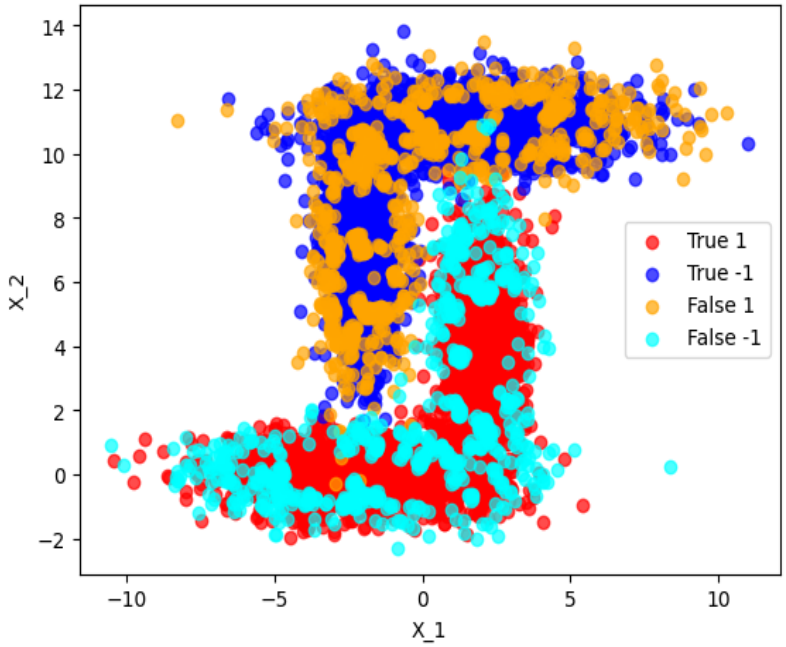


Figure 3.3: Points and their assigned class in Randomized BCGD

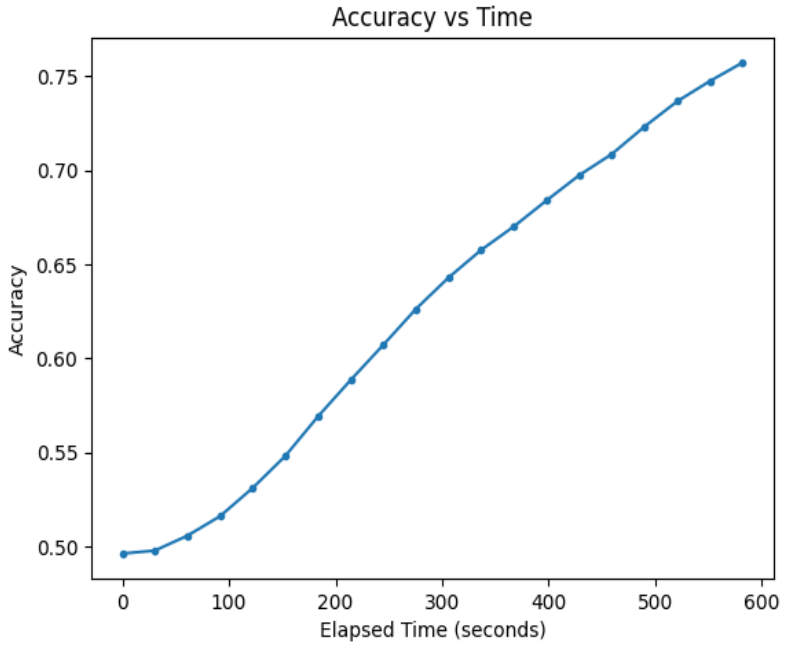


Figure 3.4: Plot of Accuracy vs CPU Time for Randomized BCGD

3.3.3 Gauss-Southwell BCGD

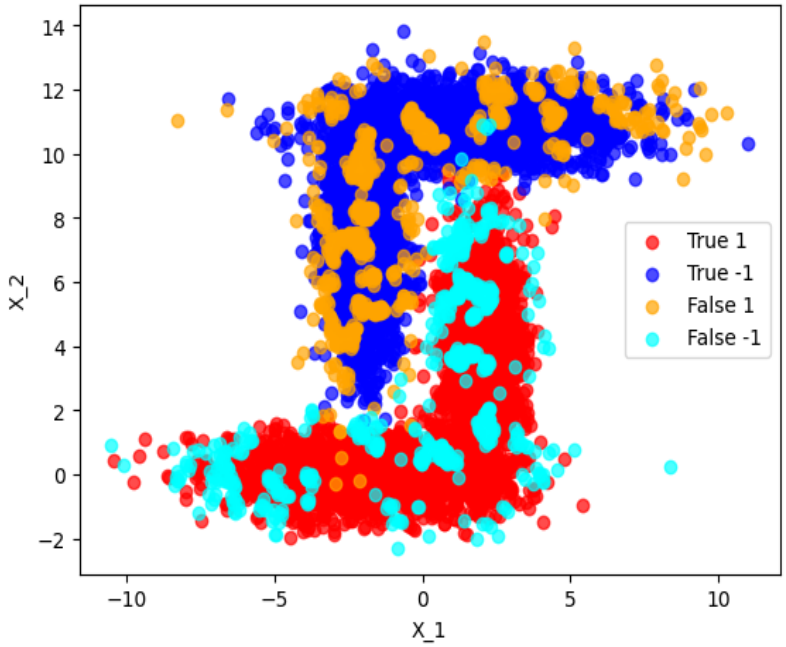


Figure 3.5: Points and their assigned class in Gauss-Southwell BCGD

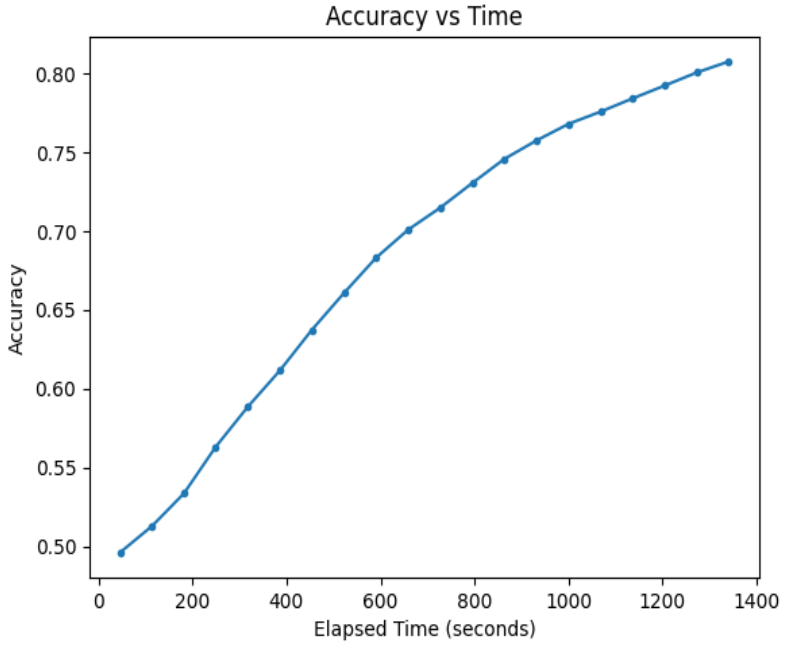


Figure 3.6: Plot of Accuracy vs CPU Time for Gauss-Southwell BCGD

3.3.4 Comparison

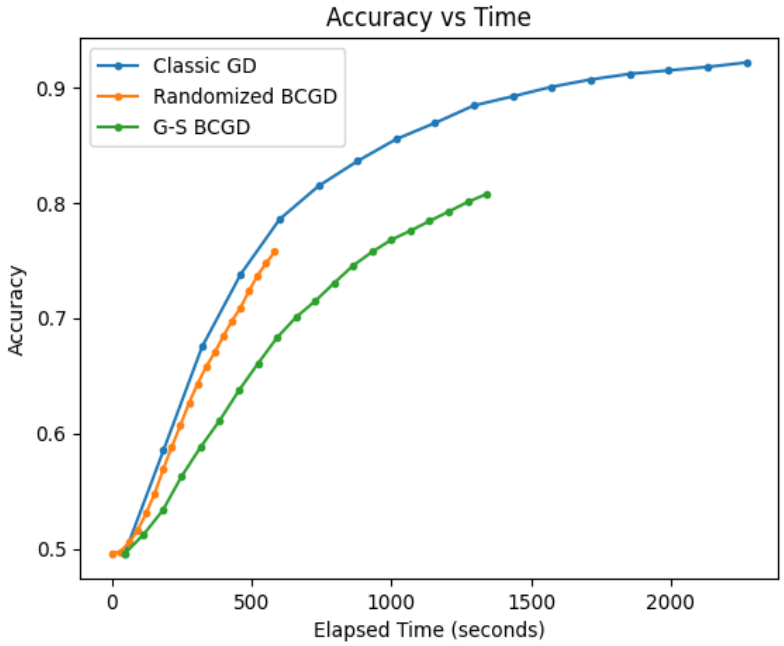


Figure 3.7: Plot of Accuracy vs CPU Time for all the methods

Chapter 4

Application on a real dataset

4.1 Banknote Authentication Data Set

The Banknote Authentication Data Set from UCI[1] contains data about authentic and forged banknotes, which defines a binary classification problem. Data is digitized through an industrial camera usually used for print inspection, while the features are extracted through a Wavelet Transform tool. The dataset has 1372 examples and 4 continuous predictors.

Target

- Class: Integer value (-1 for authentic, 1 for forged).
  There are 961 authentic examples and 411 forged examples.

Predictors

- Variance of the Wavelet Transformed image
- Skewness of the Wavelet Transformed image
- Kurtosis of the Wavelet Transformed image
- Entropy of the image

The features have been scaled to the interval [0,1]; then, 95% of examples were made unlabeled and the remaining 5% has been used to perform the labeling with the algorithms.

[1] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

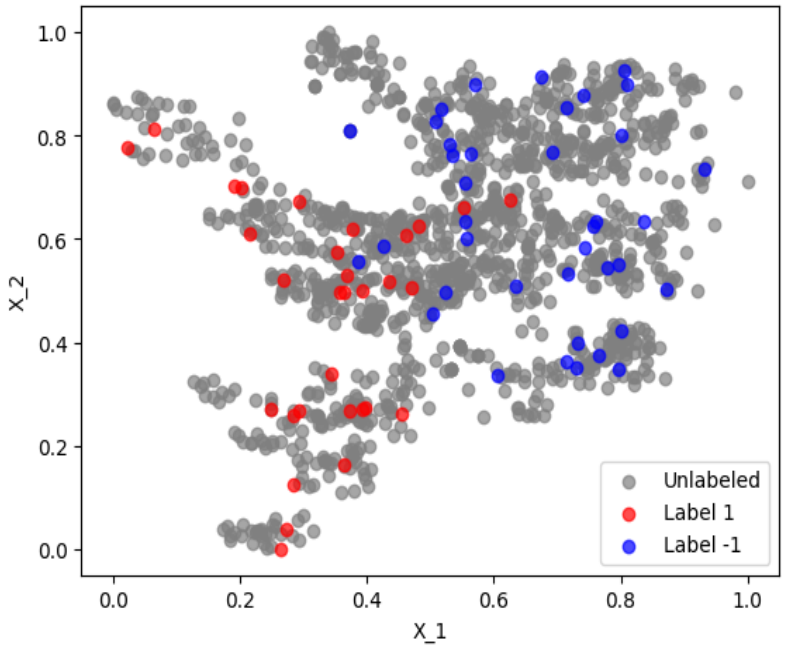


Figure 4.1: Plot of Variance (X_1) vs Skewness (X_2) of the final Banknote authentication Dataset

4.2 Algorithms Performance

The performance trend of the three algorithms appears to be similar, as evidenced by Figure 4.2, with accuracy increasing rapidly within the first 25-30 seconds before becoming stationary. The table below summarizes the results obtained.

Algorithm	**Accuracy**	**CPU time**	**Iterations**
Gradient Descent	0.9716	2min 49s	100
Random BCGD	0.9685	2min 17s	200000
Gauss-Southwell BCGD	0.9700	2min 38s	50000

Table 4.1: Performance of the algorithms in the Banknote Authentication Data Set

The Classic Gradient Descent and the Gauss-Southwell BCGD have iterated for the maximum number of epochs specified, while the Randomized BCGD has stopped earlier as it satisfied the stopping condition.
It can be concluded that all three algorithms, Gradient Descent, Random BCGD, and Gauss-Southwell BCGD, performed similarly well in terms of accuracy in the Banknote Authentication Dataset. Conversely, vari-

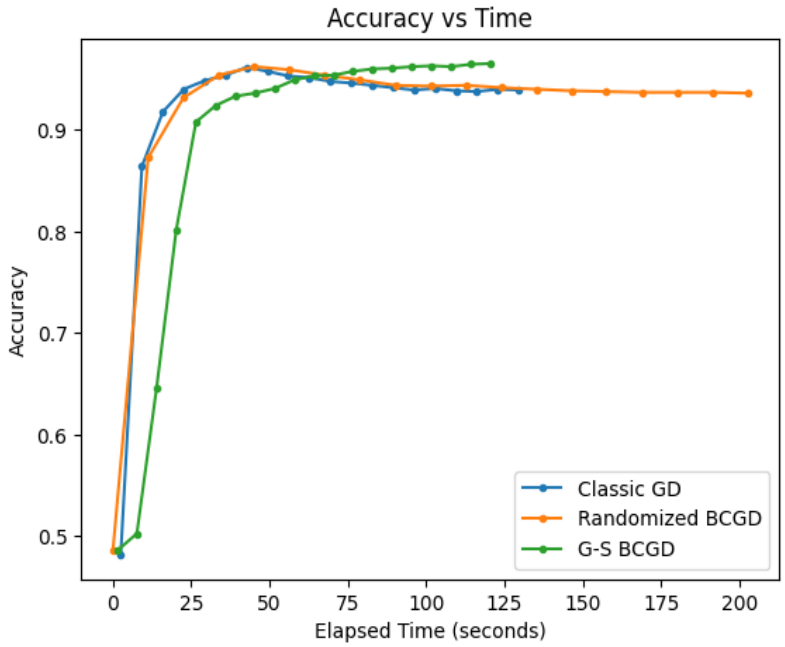


Figure 4.2: Plot of Accuracy VS CPU time of the algorithms

ations were observed in the CPU time and number of iterations needed by each algorithm to reach the desired level of accuracy.

The Gradient Descent algorithm resulted in the lowest number of iterations required to achieve an accuracy of $> 97\%$, but it also had the longest CPU time since it needs to compute the full gradient at each iteration; as discussed in Section 3.3, this is an expected result.

On the other hand, Random BCGD had a significantly larger number of iterations, but a shorter CPU time of 2 minutes and 17 seconds, this is because BCGD updates only a subset of the variables at each iteration, whereas Gradient Descent updates all the variables simultaneously.
One reason why BCGD may require more iterations than Gradient Descent is that BCGD updates only a subset of the variables at each iteration, whereas Gradient Descent updates all the variables simultaneously. BCGD may require more iterations if the choice of hyperparameters, such as the size of the blocks and the step size, is not well-suited to our problem.

Gauss-Southwell BCGD has a good trade-off since it achieved high accuracy of 97% with a moderate amount of computational resources.

4.3 Conclusions

Overall, it is evident that the algorithms could have achieved a similar accuracy by running for just few epochs, indicating that a reduction in the maximum number of iterations could be applied for larger-scale applications. So, in conclusion, it's possible to affirm that the choice of the algorithm for a particular application may depend on factors such as the available computational resources, the desired trade-off between accuracy and computation time, and considering the specific context of analysis.