Gradient Descent Methods for Semi-Supervised Learning: A Comparative Study

TABLE OF CONTENTS

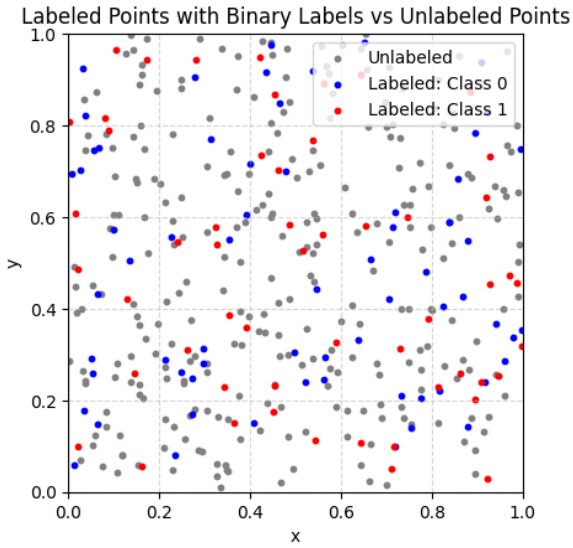
1. GENERAL SCHEME

The following subsection describes the general scheme that will be the basis for all the following both theoretical and on-real-data analysis, that will comprehend the implementations of classical Gradient Descent (with both fixed and Armijo-based step sizes), Block Coordinate Gradient Descent (with Gauss-Southwell rule), and Coordinate Minimization.

1.1. DATA GENERATION

To perform our analysis on some optimization semi-supervised learning algorithms 400 random points in a [0,1]^2 space were randomly and uniformly generated. These data points have been later split in two subsets:

- A subset, ‘labeled_points’, consisting in the 30% (L = 120 data points) of the original one, which has been randomly binary classified
- A subset, ‘unlabeled_points’, consisting in the remaining 70% of the original one, which remained unlabeled



1.2. WEIGHT COMPUTATION

Then, two similarity matrices have been computed:

- W (120 x 280) : in which each entry, w_{ij}, corresponds to 1 - the normalized euclidean distance between the i-th labeled data point (i = 1, …, 120) and the j-th unlabeled data point (j = 1, …, 280), with w_{ij} that maps in [0,1] and grows as the i-th labeled point and j-th unlabeled point are closer

Shape of W: (120, 280)
Sample of W:
[[0.4897414 0.85813911 0.50733474 0.35147821 0.38314473]
[0.45663731 0.49207692 0.82225578 0.21606731 0.50556549]
[0.835458 0.43869968 0.40204002 0.86782865 0.68763487]
[0.90193752 0.57140185 0.65561419 0.6491001 0.8203482]
[0.92473529 0.51391701 0.62017968 0.67516139 0.85874313]]

- $\bar{W}$(280 x 280): in which each entry, $\bar{w}_{ij}$, corresponds to 1 - the normalized euclidean distance between the i-th unlabeled data point (i = 1, ..., 280) and the j-th unlabeled data point (j = 1, …, 280). The $\bar{W}$ matrix results in a symmetric matrix (the similarity between the i-th and j-th unlabeled and the one between the j-th and the i-th data points are equal), with 1s diagonal values (the similarity between a data point and itself will always be maximum, as per definition)

Shape of W_hat: (280, 280)
Sample of W_hat:
[[1. 0.53946988 0.55361013 0.74127991 0.78504723]
[0.53946988 1. 0.43244784 0.44957461 0.38823593]
[0.55361013 0.43244784 1. 0.29603248 0.65644577]
[0.74127991 0.44957461 0.29603248 1. 0.55695541]
[0.78504723 0.38823593 0.65644577 0.55695541 1.]]

1.3. LOSS FUNCTION

The optimization problem we want to solve is the following one:

$$\min_{y \in R^U} \sum_{i=1}^{L} \sum_{j}^{U} w_{ij} \left(y^j - \bar{y}^i\right)^2 + \frac{1}{2} \sum_{i=1}^{U} \sum_{j}^{U} \bar{w}_{ij} \left(y^i - y^j\right)^2$$

To effectively visualize the trend of the loss function over the iterations, for each algorithm has been provided a loss function - number of iteration plot, that clearly shows the improvements gained in terms of the objective function value.

1.4. FIRST DERIVATIVE OF THE LOSS FUNCTION

And the first derivative with respect y_j :

$$2\sum_{i=1}^{L} w_{ij}(y_j - \bar{y}_i) + 2\sum_{i=1}^{U} \bar{w}_{ij}(y_j - y_i)$$

1.5. THRESHOLDING

In machine learning classification tasks, the conversion of continuous prediction values into discrete labels plays a crucial role in classifying data points accurately. This process is commonly achieved by applying a threshold to continuous output values, typically probabilities, to categorize data into predefined classes. This report outlines the application of thresholding in a classification model, followed by the visualization of labeled and newly predicted data points.

In this scenario, we aim to classify a set of points into two classes based on their feature vectors. Initially, we have labeled points (points that have already been assigned a class) and unlabeled points (points that need to be predicted). The labels for the labeled points are binary, representing two distinct classes (Class 0 and Class 1). The model, having made predictions for the unlabeled data points, generates continuous output values. These values represent probabilities or confidence levels that each point belongs to a particular class.

Thresholding is the process of converting continuous predictions (ranging from 0 to 1) into binary labels. A threshold value, in this case, 0.45, is selected to determine the class label for each predicted point. If a point's predicted value is greater than or equal to the threshold (0.45), it is classified as Class 1. If the predicted value is less than the threshold, the point is classified as Class 0.
In our case, we applied a threshold of **0.45**, meaning:

- Predictions ≥ 0.45 are assigned to Class 1.
- Predictions < 0.45 are assigned to Class 0.

After applying the thresholding, the unlabeled points are assigned predicted class labels. These newly labeled points are then combined with the already labeled data. This ensures that the dataset now contains both labeled and newly predicted data points, which are crucial for evaluating the accuracy of the classification model and providing more comprehensive insights.

1.6. DATA VISUALIZATION

Data visualization plays an essential role in understanding how well the classification model performs and how the threshold has influenced the prediction. For the gradient descent with a fixed step size has been provided a scatter plot that clearly demonstrates the classification of both labeled and unlabeled data. It provides a visual representation of how the thresholding process has grouped the unlabeled points into their respective classes. The graphs contain four distinct groups of points, with the following characteristics:

- **Blue points** represent the labeled points belonging to Class 0.
- **Red points** represent the labeled points belonging to Class 1.
- **Green points** represent the newly predicted Class 0 points from the unlabeled set.
- **Pink points** represent the newly predicted Class 1 points from the unlabeled set.

2. GRADIENT DESCENT

Gradient descent is an iterative optimization algorithm that adjusts model parameters by moving in the direction of the negative gradient of the loss function. The step size determines the magnitude of each update, balancing the trade-off between convergence speed and stability. If the step size is too large, the algorithm may overshoot the minimum and fail to converge. Conversely, if the step size is too small, the optimization process becomes inefficient, requiring numerous iterations to reach a satisfactory solution.

2.1. FIXED STEP SIZE GRADIENT DESCENT

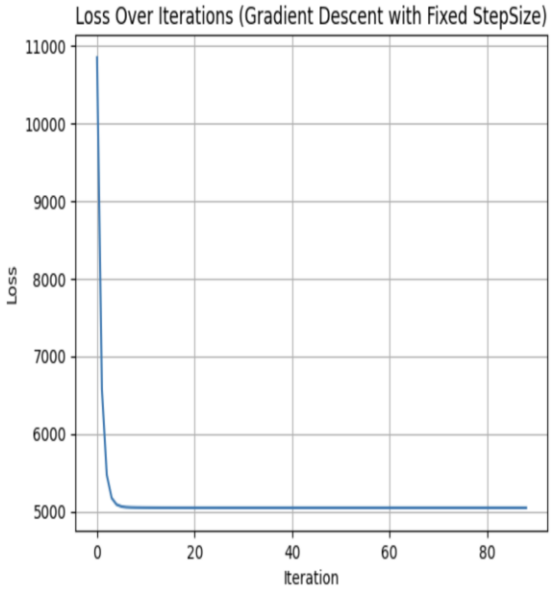
The Fixed Step Size method employs a constant learning rate throughout the optimization process. This approach is straightforward to implement, as it requires only a single hyperparameter—the step size—to be specified in advance. However, its simplicity comes with notable limitations. A poorly chosen step size can lead to suboptimal performance: excessively large steps may cause the algorithm to diverge, while overly small steps result in slow convergence.

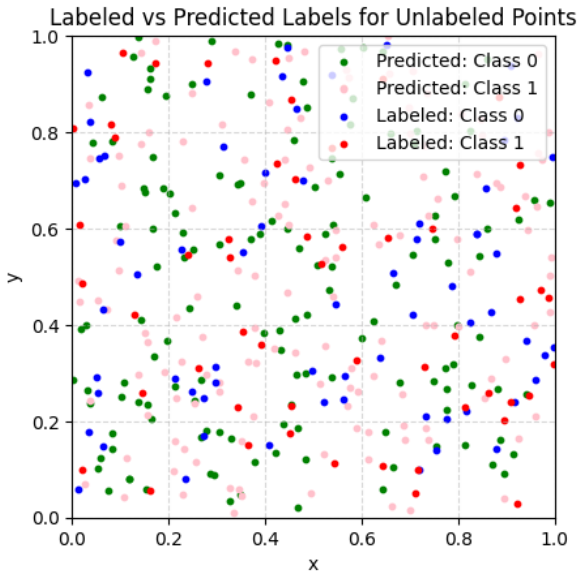
2.1.1. ARBITRARY FIXED STEP SIZE

In our experiments on the synthetic dataset, the Fixed Step Size method was arbitrarily applied with a learning rate of 0.001 over 500 iterations. While the algorithm succeeded in reducing the loss, the convergence was sluggish. The gradient norm decreased gradually but remained relatively high even after many iterations, indicating that the fixed step size hindered the algorithm's ability to fine-tune the parameters effectively. Additionally, the loss plateaued prematurely, suggesting that the constant step size was unable to adapt to the varying curvature of the loss landscape.

Iteration 0, Loss: 10848.0940, Gradient Norm: 2385.011548
Iteration 50, Loss: 5045.5886, Gradient Norm: 0.038270
Converged at iteration 88

The gradient descent optimization process appears to have been successful and efficient. Initially, at iteration 0, the loss was quite high at 10848.0940, with a large gradient norm of 2385.0115, indicating the model was far from an optimal solution and the slope of the loss function was steep. By iteration 50, the loss had significantly decreased to 5045.5886, and the gradient norm had dropped to 0.03827, suggesting the model was approaching a minimum and the updates were becoming smaller. The algorithm ultimately converged at iteration 88, meaning the changes in the parameters became negligible and the loss function had reached a relatively stable value. This pattern shows effective convergence behavior, where gradient descent steadily minimized the loss function and reached a solution efficiently.





2.1.2. LIPSCHITZ CONSTANT BASED FIXED STEP SIZE

Afterwards, instead of choosing arbitrarily the step size, we decided to study the algorithm performance with Lipschitz constant based step size.

Since our loss function is quadratic and its first derivative is linear (e.g. $DL(y) = Ay + b$), the rate at which the gradient changes is just determined by the matrix multiplying y (A).

So, how fast does the gradient change? It depends on the largest eigenvalue of A, because eigenvalues tell how much the matrix can 'stretch' a vector and in gradient descent we want to avoid taking steps that overshoot the minimum. If the function has directions where it changes very quickly (big eigenvalues) we need to take smaller steps in those directions (and therefore we use the inverse of Lipschitz constant).

So, our loss functions has a Lipschitz constant equal to the largest eigenvalue of the matrix multiplying y (A), that, in our case, is $W^T W + \bar{L}$, with:

W the already computed weights similarity matrix (between labeled and unlabeled data)

$\bar{L} = \bar{D} - \bar{W}$, where

$\bar{W}$ is the already computed similarity matrix (between unlabeled data)

$\bar{D}$ is a diagonal matrix in which each entry is the row sum of $\bar{W}$

This term appears in the second term of the loss function, which, in fact, can also be written as $y^T \bar{L}\, y$, with the goal of penalizing differences between connected nodes, enforcing smoothness.

Putting everything together, the loss function can also be written as:

$L(y) = y^T (W^T W + \bar{L})y + (linear\ terms)$

$DL(y) = 2(W^T W + \bar{L})y + (constant\ vector)$

And, therefore:

$Lipschitz\ constant = L = 2\,\lambda_{max}(W^T W + \bar{L})$

This formulation reflects an important principle in graph-based semi-supervised learning: smoothness. The term $y^T \bar{L}\ y$ in the loss function penalizes differences between the predicted labels of similar (connected) unlabeled nodes, thereby encouraging label consistency in regions of the graph with high similarity.

The reciprocal of this maximum eigenvalue is then used as the step size alpha_lipschitz, guaranteeing that the gradient descent will not overshoot the minimum, which could cause divergence or oscillation.

Estimated Lipschitz constant: 25280.0561, Step size (1/L): 0.000040

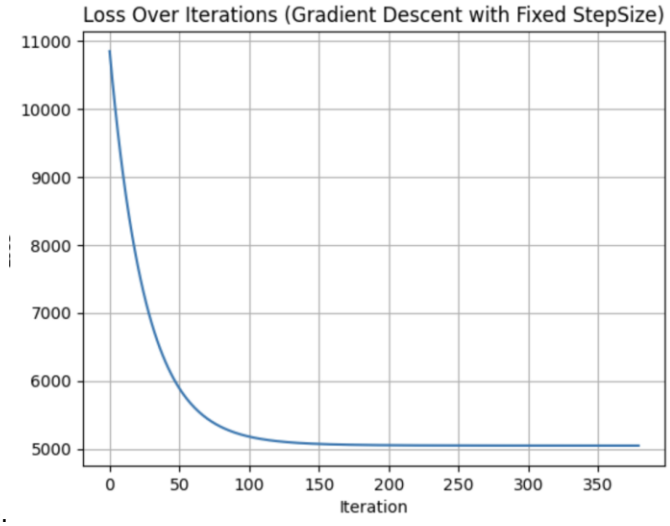
Gradient descent has then been built using the computed $\alpha_{LIPSCHITZ} = 1 / L$ as the learning rate. This is a more principled approach than choosing an arbitrary value for alpha, as it is grounded in the theoretical behavior of the loss function.
The code initializes the predicted labels for the unlabeled data points (y_unlabeled) randomly and iteratively updates them by subtracting the gradient of the loss function with respect to each component of y_unlabeled. These gradients are calculated using a helper function, gradient_yj, for each index j. The gradients are clipped between -10 and 10 to prevent instability due to potential outliers or extreme values, which is a common regularization trick in numerical optimization.
The convergence is monitored through the norm of the gradient vector — when it drops below a small threshold (tolerance), the algorithm halts, assuming it has reached a minimum or a point sufficiently close to one. Additionally, the loss value is computed and stored at each iteration to track how the optimization progresses over time. The output shows a steady decrease in the loss and the gradient norm over 1450 iterations, ending with a gradient norm as low as 0.006, which indicates excellent convergence.
By plotting the loss_history, the algorithm visualizes how the optimization stabilizes. The convergence curve typically flattens out as the model reaches an optimal or near-optimal configuration for the unlabeled data predictions.

This method of using a Lipschitz-based fixed step size is especially suitable for problems with a quadratic loss function, where eigenvalue analysis is feasible. It ensures a balance between speed and stability, avoids trial-and-error for choosing alpha, and provides convergence guarantees under convexity. This implementation is a strong example of combining theory and practice in optimization, particularly in semi-supervised graph-based learning scenarios.

Iteration 0, Loss: 10848.0940, Gradient Norm: 2385.011548
Iteration 50, Loss: 5887.4005, Gradient Norm: 894.450088
Iteration 100, Loss: 5178.1169, Gradient Norm: 344.329902
Iteration 150, Loss: 5070.2280, Gradient Norm: 138.440343
Iteration 200, Loss: 5051.8427, Gradient Norm: 60.757979
Iteration 250, Loss: 5047.8997, Gradient Norm: 31.232418
Iteration 300, Loss: 5046.6896, Gradient Norm: 19.255527
Iteration 350, Loss: 5046.1740, Gradient Norm: 13.368512
Converged at iteration 379

This gradient descent run, using a Lipschitz-based step size, demonstrates a stable and controlled convergence process. Initially, at iteration 0, the loss was high at 10848.0940, with a steep gradient norm of 2385.0115, indicating a poor starting point and a steep slope in the loss landscape. Unlike aggressive step size strategies, the Lipschitz-based approach results in more cautious updates, as seen in the

slower but steady decrease in both loss and gradient norm over time. By iteration 50, the loss had dropped to 5887.4005 and the gradient norm to 894.4501, and this gradual descent continued until convergence at iteration 379. At that point, the loss had stabilized around 5046.1740 with a much smaller gradient norm of 13.3685. This behavior reflects the conservative nature of Lipschitz-based step sizing, which ensures stability and avoids overshooting, making it especially useful when the curvature of the objective function is highly variable.



2.1.3. COMPARISON BETWEEN THE TWO FIXED STEP SIZE STRATEGIES

The comparison between the two gradient descent strategies highlights key differences in convergence speed and stability. With a fixed step size, the algorithm converged rapidly, reaching a loss of approximately 5045.6 in just 88 iterations, with the gradient norm dropping sharply to near zero by iteration 50. This indicates fast progress toward a local minimum, but such aggressive updates can risk instability or overshooting in more complex problems. In contrast, the Lipschitz-based step size led to a much more gradual descent, requiring 379 iterations to converge. The loss decreased steadily from the same starting point but did so more cautiously, with the gradient norm remaining relatively large until much later in the process. While slower, this approach ensures more stable and controlled updates, especially beneficial in cases where the loss surface has high curvature or is not well-behaved. Thus, the fixed step size offers faster convergence when well-tuned, whereas the Lipschitz-based step size prioritizes reliability and stability.

2.2. ARMIJO RULE GRADIENT DESCENT

Unlike the Fixed Step Size method, the Armijo Rule dynamically adjusts the learning rate at each iteration based on the local behavior of the loss function. This adaptive approach ensures that the step size is neither too large nor too small by enforcing a sufficient decrease condition. Specifically, the Armijo Rule checks whether a proposed step size leads to an adequate reduction in the loss before accepting it. If the condition is not met, the step size is progressively reduced until the criterion is satisfied.

The behaviour of the algorithm at each iteration k=1, 2, ... is the following:

$\alpha_{init} = 1.0$

$m = 0$

until $f(x_k + \alpha d_k) \leq f(x_k) + \gamma\alpha \nabla f(x_k)^T d_k$, with $\gamma \in (0, 0.5)$, and d_k a descent direction, is satisfied:

$\alpha = \delta^m \alpha_{init}$, with $\delta \in (0,1)$

$m = m + 1$

$\alpha_k = \alpha$

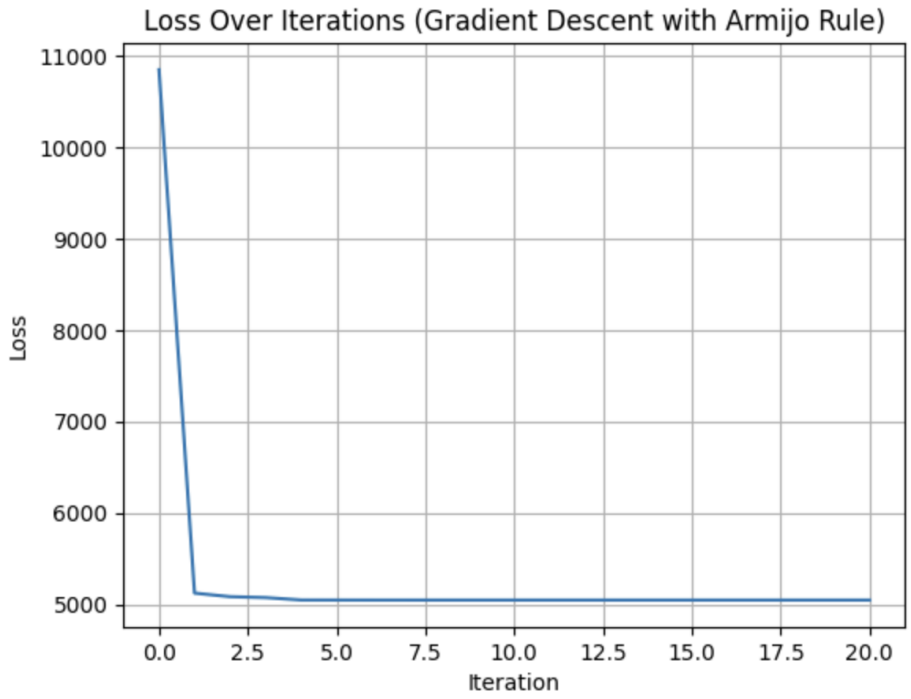
2.2.1. ARMIJO RULE HYPERPARAMETERS FINE TUNING

The Armijo Rule based step size expects gamma and delta, two hyperparameters to be fine tuned.
To do so we run a grid search algorithm and we found out that the best hyperparameters in terms of number of oracle calls to reach convergence are a delta value of 0.5 and a gamma value of 0.05

Delta: 0.50, Gamma: 0.0500, Final Loss: 5045.5886, Iterations: 20, Converged: True
Delta: 0.50, Gamma: 0.0010, Final Loss: 5045.5886, Iterations: 22, Converged: True
Delta: 0.50, Gamma: 0.0100, Final Loss: 5045.5886, Iterations: 22, Converged: True
Delta: 0.50, Gamma: 0.1000, Final Loss: 5045.5886, Iterations: 22, Converged: True
Delta: 0.30, Gamma: 0.0010, Final Loss: 5045.5886, Iterations: 26, Converged: True

We then applied the method on the whole dataset using the tune hyperparameters , getting the following results:

Iteration 0, Loss: 10848.0940, Gradient Norm: 2385.011548, Step Size: 0.00195, m: 9
Iteration 10, Loss: 5045.5890, Gradient Norm: 0.687744, Step Size: 0.00195, m: 9
Stopping: exceeded max_m
Converged at iteration 20, Loss: 5045.5886



2.3. BLOCK COORDINATE GRADIENT DESCENT WITH GS RULE

Block Coordinate Gradient Descent is an extension of the classic gradient descent method that operates on one parameter at a time. The main difference from it is that, instead of updating all the variables simultaneously, it updates only a block (group) of them, computing the partial gradient with respect to that particular block of variables . This is particularly useful in high-dimensional problems where updating all parameters simultaneously may lead to computational inefficiencies or convergence difficulties and where the variable space can be naturally partitioned into blocks.

There are many variants of the BCGD algorithm. In this experiment we built a BCGD following the Gauss-Southwell rule for block selections. This rule aims to choose the block that violates the most the optimality conditions, hence the block whose improvement will cause the greatest decrease in the loss function.

The behaviour of the algorithm at each iteration k=1, 2, ... is the following:

If x_k satisfies some conditions:

STOP

Else:

Pick block $j : \max_{j \in \{1,\dots,b\}} ||\nabla_j f(x_k)||$

Set $l = | j |$

For l = 1, ..., l:

$$y_i = y_{i-1} - \alpha_i U_{ji} \nabla_{ji} f(y_{i-1}),$$

With $\alpha_i > 0$

Set $x_{k+1} = y_l$

End for

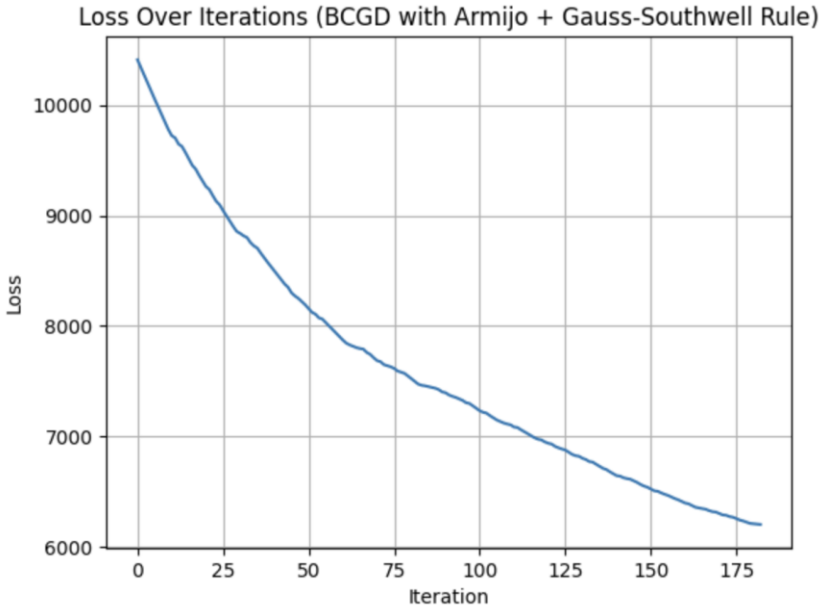
The convergence of the algorithm is monitored by calculating the norm of the gradient after each iteration. The algorithm stops if the gradient norm is smaller than the specified tolerance, which indicates that the algorithm has reached a point where further updates are unlikely to significantly reduce the loss function.

The results obtained are:

Iteration 0, Loss: 10411.5510, Gradient Norm: 2302.456413
Iteration 10, Loss: 9723.9601, Gradient Norm: 2137.731803
Iteration 20, Loss: 9263.5469, Gradient Norm: 2020.331711
Iteration 30, Loss: 8838.1805, Gradient Norm: 1903.409221
Iteration 40, Loss: 8496.6020, Gradient Norm: 1803.429633
Iteration 50, Loss: 8157.5115, Gradient Norm: 1705.407154
Iteration 60, Loss: 7871.8373, Gradient Norm: 1621.042209
Iteration 70, Loss: 7685.1496, Gradient Norm: 1547.475341
Iteration 80, Loss: 7522.2222, Gradient Norm: 1497.539957
Iteration 90, Loss: 7394.5969, Gradient Norm: 1461.218167
Iteration 100, Loss: 7230.8238, Gradient Norm: 1415.551627
Iteration 110, Loss: 7083.8891, Gradient Norm: 1375.407784
Iteration 120, Loss: 6936.7114, Gradient Norm: 1326.920792
Iteration 130, Loss: 6800.4998, Gradient Norm: 1281.177237
Iteration 140, Loss: 6642.7248, Gradient Norm: 1219.211454
Iteration 150, Loss: 6521.6562, Gradient Norm: 1174.329836
Iteration 160, Loss: 6395.0242, Gradient Norm: 1122.371361
Iteration 170, Loss: 6300.2891, Gradient Norm: 1078.443758
Iteration 180, Loss: 6206.1566, Gradient Norm: 1031.149151
Converged at iteration 182

The block coordinate gradient descent (BCGD) with Gauss–Southwell (GS) update rule shows a gradual and consistent convergence pattern over a longer trajectory. Starting with a loss of 10411.5510 and a large gradient norm of 2302.4564, the method steadily reduces both metrics across iterations. At every 10-iteration mark, we observe a significant but decelerating drop in loss and gradient norm, indicating controlled and stable progress. By iteration 180, the loss drops to 6206.1566 with a gradient norm of 1031.1492, and the method converges shortly after at iteration 182. Compared to previous methods: standard gradient descent with fixed or adaptive step sizes, This approach is slower in terms of iteration count but demonstrates consistent and reliable descent. The Gs rule, which updates variables sequentially and immediately uses the most recent values, promotes numerical stability and is particularly

useful in problems with block-structured variables or where a full gradient update is computationally expensive.



2.4. COORDINATE MINIMIZATION

Coordinate minimization is a powerful optimization strategy for semi-supervised learning, especially in label propagation tasks. It operates by updating one variable at a time while holding others constant, which simplifies high-dimensional problems into manageable steps. In our implementation, we observed rapid and effective convergence: starting from an initial loss of 10,155.94 and a gradient norm of 165.75, the method reduced the loss to 5,045.59 and the gradient norm to just 0.000342 within 50 iterations. The algorithm converged fully by iteration 57, showing its efficiency in minimizing the objective function through targeted, one-dimensional updates.

This approach offers notable advantages, including low memory requirements, computational efficiency, and the ability to exploit sparsity in graph-based learning tasks. It is especially useful when labeled data is limited but unlabeled data is abundant, such as in text classification or biological sequence analysis. While its fixed learning rate may slow convergence in later stages and its sequential updates limit parallelization, future work could explore adaptive learning rates or hybrid strategies to enhance its speed and scalability.

From a practical perspective we implemented the Coordinate minimization algorithm as:

```
# --- Cycle through each unlabeled coordinate (y_unlabeled[j]) ---
for j in range(U):
    sum_W_hat_ju_except_j = np.sum(W_hat[j, :] * y_unlabeled) - W_hat[j, j] * y_unlabeled[j]
    sum_W_hat_j_except_j = np.sum(W_hat[j, :]) - W_hat[j, j]

    numerator = 2*np.sum(W[:, j] * labels.flatten()) + sum_W_hat_ju_except_j
    denominator = 2*np.sum(W[:, j]) + sum_W_hat_j_except_j

    if denominator > epsilon:
         y_unlabeled[j] = numerator / denominator

# Calculate the maximum absolute change in y_unlabeled across this iteration's cycle
max_change = np.max(np.abs(y_unlabeled - y_unlabeled_old))
y_unlabeled_history.append(y_unlabeled.copy()) # Store updated y_unlabeled

if max_change < tolerance:
    print(f"Converged at iteration {iteration} with max change {max_change:.6f}")
    break
```

In details, we minimized the loss function, setting its derivative, w.r.t. Unlabeled variables, to zero. In this way we obtained:

$$0 = \frac{\partial L}{\partial y_{U_j}} = \frac{\partial}{\partial y_{U_j}} \left(\sum_{i}^{L} \sum_{j}^{U} W_{ij} (y_{L_i} - y_{U_j})^2 + 1/2 \sum_{k}^{U} \sum_{j}^{U} \hat{W}_{kj} (y_{U_k} - y_{U_j})^2 \right)$$

Further computations give us:

$$0 = -2 \sum_{i}^{L} W_{ij} (y_{L_i} - y_{U_j}) - \sum_{k,k \neq j}^{U} \hat{W}_{kj} (y_{U_k} - y_{U_j})$$

And then an explicit formulation for the variable that minimizes the loss function as:

$$y_{U_j} = \frac{2 \sum_{i}^{L} W_{ij} y_{L_i} + \sum_{k,k \neq j}^{U} \hat{W}_{kj} y_{U_k}}{2 \sum_{i}^{L} W_{ij} + \sum_{k,k \neq j}^{U} \hat{W}_{kj}}$$

In our implementation we iterate over j and find the value of j for which we get the best improvement in the loss function. Then we update y_unlabeled.

Iteration 0, Loss: 10155.9375, Gradient Norm: 165.753648
Iteration 50, Loss: 5045.5886, Gradient Norm: 0.000342
Converged at iteration 57

3. COMPARISONS AND THEORETICAL CONCLUSIONS

Method	Iterations	Loss (Initial → Final)	Final Gradient Norm
Fixed Step Size GD	88	10,848.09 → 5,045.59	0.03827
Lipschitz-based GD	379	10,848.09 → 5,046.17	13.37
Armijo Rule GD	20	10,848.09 → 5,045.59	0.68774
BCGD (Gauss-Southwell)	182	10,411.55 → 6,206.16	1,031.15
Coordinate Minimization	57	10,155.94 → 5,045.59	0.000342

The experiment demonstrated how each different optimization method provides distinct results in convergence oracle calls, robustness and computational efficiency. Among all the tested strategies, the Armijo Rule Gradient Descent demonstrated to be the most fitted for the problem. Thanks to its adaptive step size mechanism, it reached the optimal loss in 20 iterations - way less than the other methods. Its ability to dynamically adjust to the curvature of the surface of the loss function enables rapid and stable convergence.

Fixed Step Size Gradient Descent - while also being relatively fast - depended mainly on the choice of the step size. When the step size was well tuned, as in the first experiment, it proved efficient learning behavior, converging in 88 iterations. However, this speed and its easy implementation from a coding point of view, comes with a price: without proper tuning, the method may either overshoot the minimum or converge slowly, making the method not robust. On the other hand, with a fixed step size equal to the Lipschitz Constant of the objective function, that prioritized theoretical guarantees and caution, it turned out to be the slowest method, requiring 379 iterations to converge.

Block Coordinate Gradient Descent with Gauss-Southwell Rule presented a different advantage: structured and scalable optimization. In our experiment it required way more iterations to converge (182) and showed a slower reduction in loss and gradient norm than the previous methods. BCGD is in fact

known for its significant practical benefits in large-scale problems where full gradient evaluation is expensive, which was not our case.
Lastly, Coordinate Minimization achieved significant reduction in both the loss and gradient norm in the 57 iterations it took to converge.

In conclusion, the choice of optimization method should align closely with the problem structure and computational constraints. Armijo Rule GD is the best all-around performer; Fixed Step Size GD works well with careful tuning (if Lipschitz constant based offers cautious reliability); BCGD and Coordinate Minimization don’t show their full potential with low-dimensional optimization problems.

4. TEST ON BREAST CANCER WISCONSIN (DIAGNOSTIC) DATASET

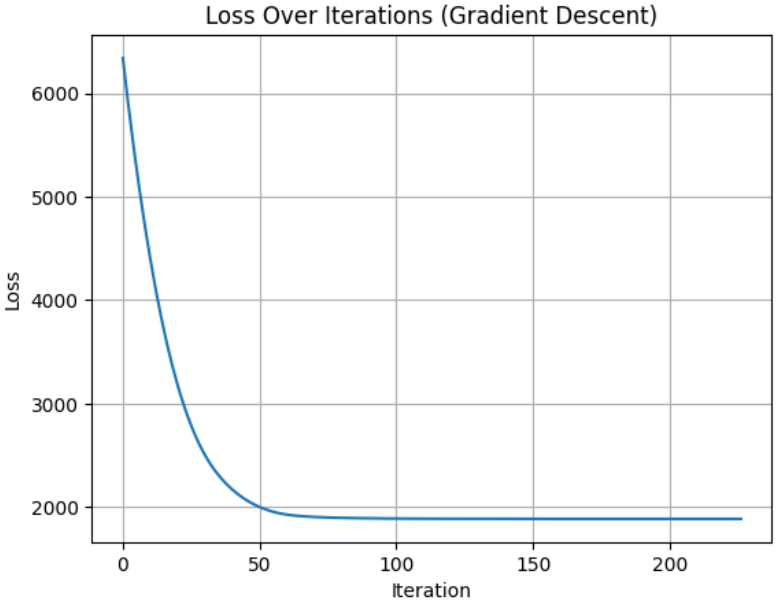
The Breast Cancer Wisconsin (Diagnostic) Dataset was used to evaluate different optimization methods in a semi-supervised learning context. The dataset consists of 569 samples, each containing 30 numerical features that describe cell nuclei characteristics extracted from digitized images of breast masses. The target variable is binary, indicating whether the tumor is malignant (0) or benign (1). Given the structure of the dataset, a higher threshold of 0.7 was chosen to balance predictions.

In this study, 20% of the data was treated as unlabeled, with the remaining 80% serving as labeled data. The objective was to predict labels for the unlabeled data based on inter-data relationships while minimizing a loss function that combines a label fitting term and a smoothness regularization term. To achieve this, the data was preprocessed using StandardScaler, and similarity weights were calculated using inverse Euclidean distances. Various optimization strategies were then implemented and compared in terms of convergence behavior, classification accuracy, and CPU time.

4.1. FIXED STEP SIZE GD - BREAST CANCER DATASET

The first method implemented was basic gradient descent (GD) using a fixed step size of 0.001. It converged after 226 iterations with a final gradient norm of 0.122628.

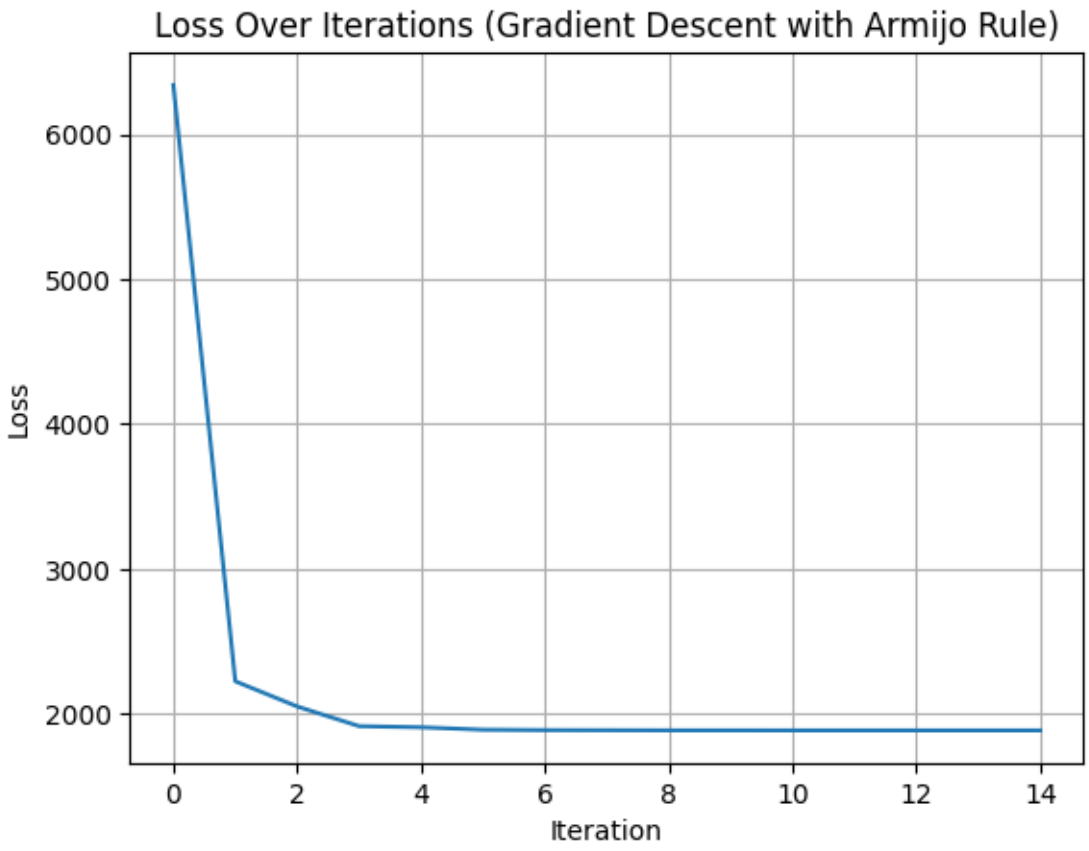
- *Accuracy on unlabeled data*: 88.38%
- *CPU time*: 80.38 seconds
- *Observation*: This approach is straightforward but suffers from slow convergence and sensitivity to the chosen step size.



4.2. ARMIJO RULE GD - BREAST CANCER DATASET

To improve robustness and avoid manual step-size tuning, the Armijo backtracking line search was applied. Starting with an initial step size of 1.0 and using a shrinkage factor of 0.5, the algorithm converged at iteration 14 with a final loss of 1885.1195.

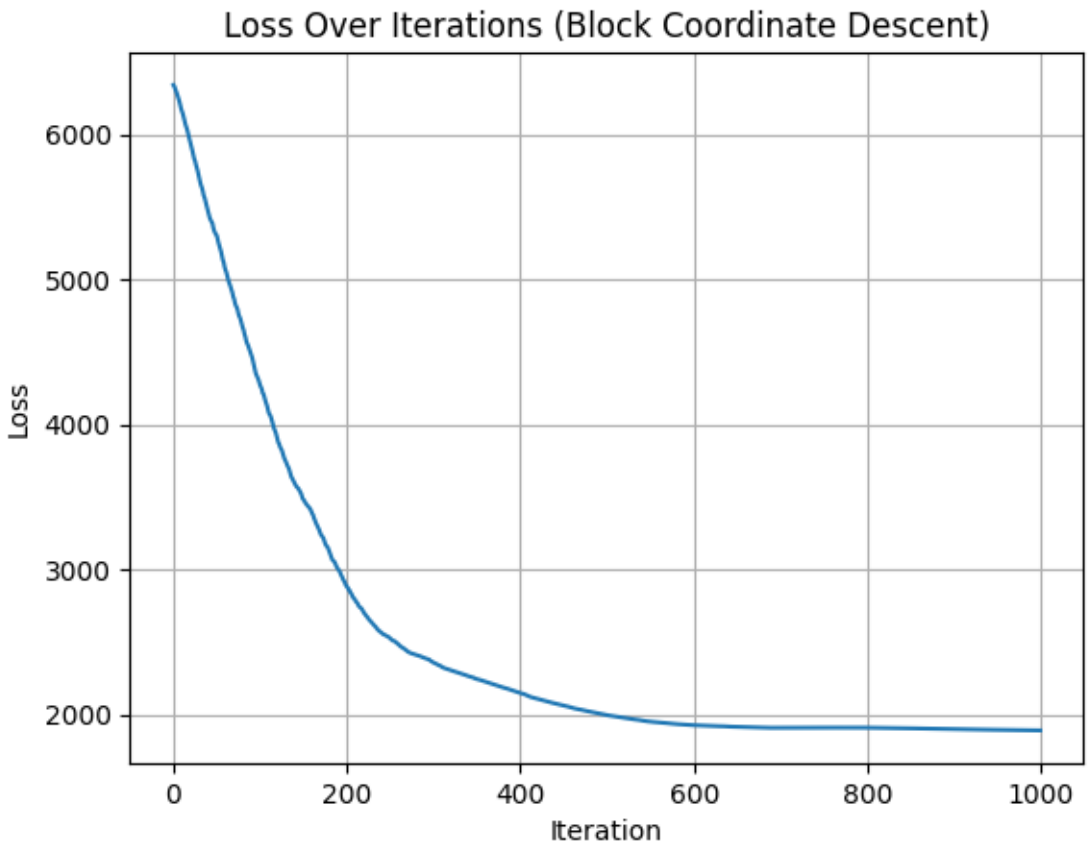
- *Accuracy on unlabeled data*: 88.38%
- *CPU time*: 27.25 seconds
- *Observation*: Adaptive step sizing leads to faster convergence and higher accuracy compared to fixed GD, at moderate computational cost.



4.3. BCGD WITH GS RULE - BREAST CANCER DATASET

This method updates one label at a time using the Gauss-Southwell rule to select the coordinate with the largest gradient magnitude. It converged after reaching the maximum iterations of 1000 total coordinate updates with loss 1892.5784and gradient norm 21.007566.

- *Accuracy on unlabeled data*: 85.75%
- *CPU time*: 348.81 seconds
- *Observation*: Offers precise, targeted updates but incurs significant computational overhead due to per-coordinate line searches.

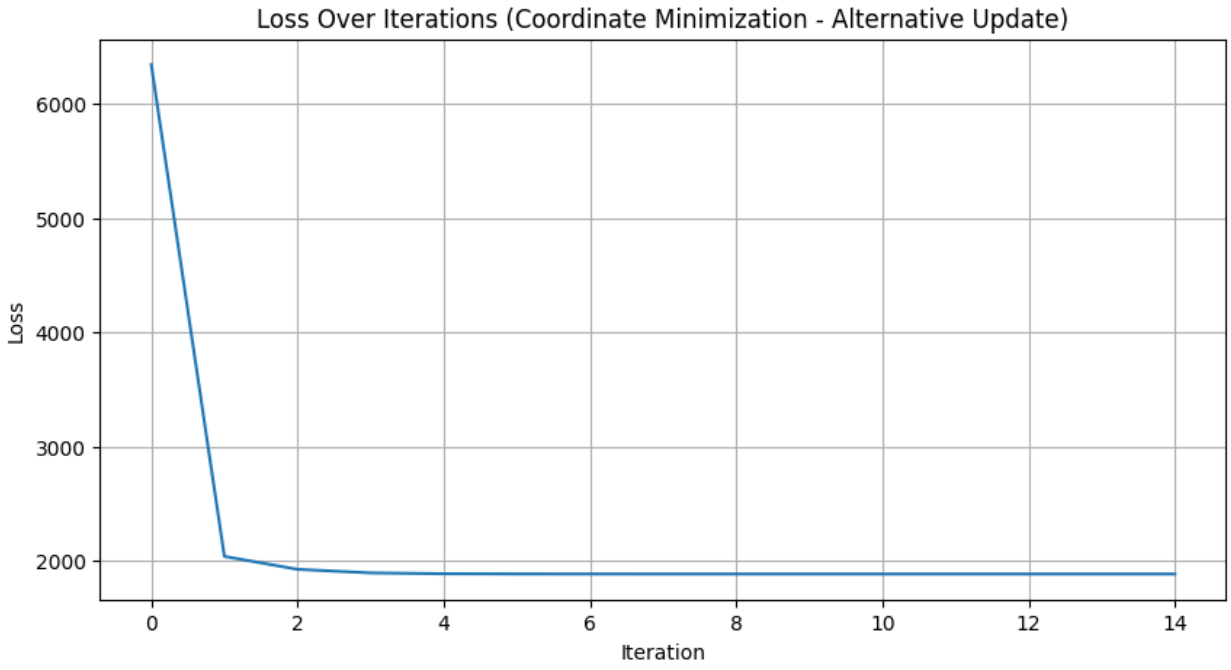


4.4. COORDINATE MINIMIZATION - BREAST CANCER DATASET

Coordinate Minimization (CM) was implemented, solving a 1D minimization subproblem for each label sequentially. The algorithm converged in 14 iterations with a maximum change below 1e-5.

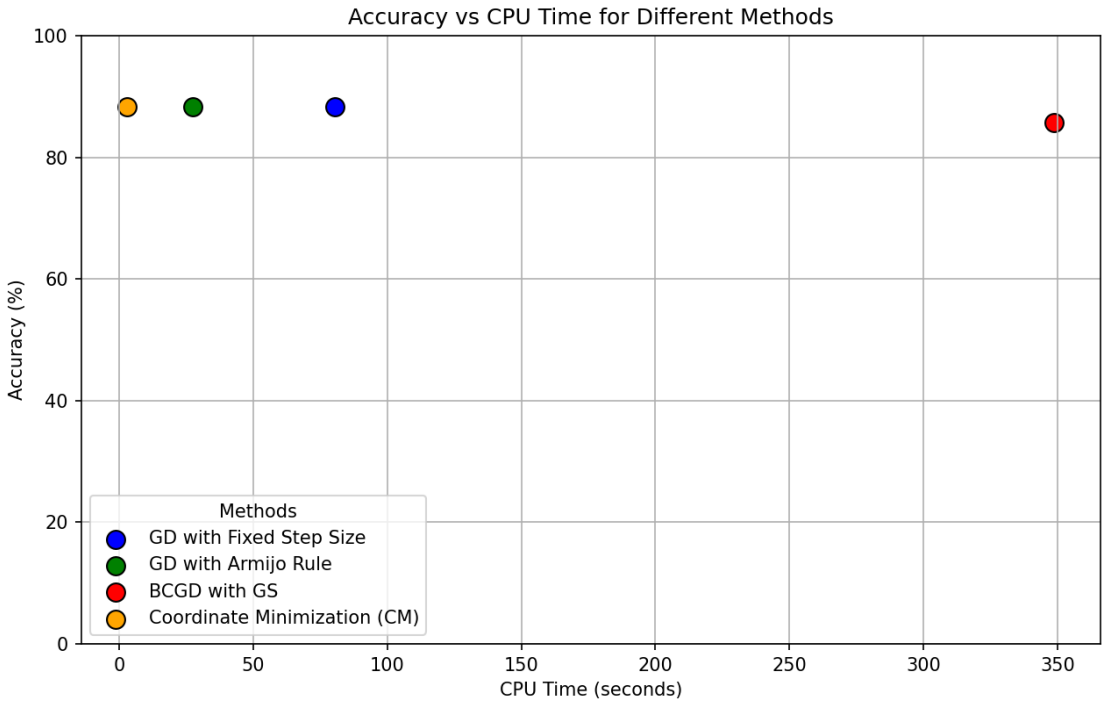
- Converged at iteration 14 with Max Change: 0.000005
- *Accuracy on unlabeled data*: 88.38%
- *CPU time*: 2.98 seconds

Observation: Quick per-iteration convergence due to closed-form updates, resulting in the fastest runtime but lower predictive performance.



4.5. COMPARATIVE ANALYSIS - BREAST CANCER DATASET

Method	Accuracy	CPU Time (s)	Iterations
GD with Fixed Step Size	88.38%	80.38	226
GD with Armijo Rule	88.38%	27.25	14
BCGD with GS	85.75%	348.81	1000+
Coordinate Minimization (CM)	88.38%	2.98	14



4.6. CONCLUSION - BREAST CANCER DATASET

The comparative analysis of the four different optimization methods provided in the project and adapted to the Breast Cancer dataset highlights a clear contrast in both accuracy and computational efficiency. Gradient Descent with Fixed Step Size, Gradient Descent with Armijo Rule, and Coordinate Minimization all achieved the highest observed accuracy of 88.38%, but with varying levels of efficiency. Coordinate Minimization stands out as the most computationally efficient, reaching optimal accuracy in just 2.98 seconds and 14 iterations, thanks to its closed-form coordinate updates. Gradient Descent with Armijo Rule matches this accuracy while being more computationally intensive than Coordinate Minimization (27.25 seconds), yet significantly faster than Gradient Descent with Fixed Step Size, which required 80.38 seconds and 226 iterations due to its slower convergence.

In contrast, Block Coordinate Gradient Descent with Gauss-Southwell was unable to reach the same accuracy, terminating early after hitting the preset maximum of 1000 iterations. It achieved a lower accuracy of 85.75% despite the highest runtime (348.81 seconds), indicating that with more iterations, it might have matched the others in accuracy but at a substantial computational cost. Overall, Coordinate Minimization and Gradient Descent with Armijo Rule emerge as the most favorable methods, offering both high accuracy and strong computational efficiency, while Block Coordinate Gradient Descent with Gauss-Southwell appears limited under iteration-constrained conditions.