SP25 - Optimization HW1 Report (Semi-supervised Learning Model)

Authors: Natalya Lavrenchuk (ID: 2141882), Christina Caporale (ID: 2141881)

Overview

Semi-supervised learning is a practical, widely-applicable approach for real-word scenarios where labeled data is scarce yet unlabeled data is often easy to obtain. Graph-based semi-supervised learning is an especially useful way to represent data and assumptions made about the data, where weighted edges reflect similarity between nodes or data points.

This report details the steps and logic behind the construction of a semi-supervised learning model using various optimization methods taught throughout the course. The goal for this model is to address a binary classification task and minimize the given loss function to accurately assign class labels to unlabeled points. In order to work with the model, first, random points are generated around two clusters and only a small portion of the points are labeled. The loss function, which relies on a similarity metric, is then minimized, thereby assigning classes to the unlabeled points. In our case, Euclidean distance was selected to define the proximity distance between labeled and unlabeled points, as well as unlabeled points to other unlabeled points. Three models are constructed, a) Gradient Descent (GD) with a fixed step size, b) Gradient Descent with the Armijo step size, and c) Block Coordinate Gradient Descent (BCGD) with the Gauss-Southwell (GS) update scheme and fixed step size. The model was then applied to a real-world dataset to classify breast cancer tumor malignancy.

Part 1 - Point generation

Five hundred points were randomly generated in a 2D space aggregated in two clusters. 20% of these points were labeled (-1 or 1) depending on cluster assignment, while the rest remained unlabeled. These points are plotted below, and provide the dataset for the following model constructions.

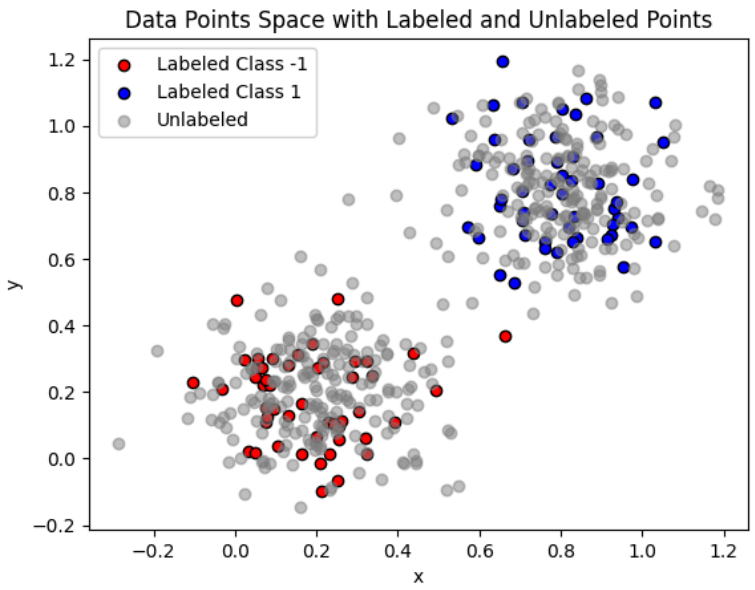

Figure 1. Generated plot of labeled and unlabeled data points grouped in two clusters. This provides the evaluation framework for the following semi-supervised learning models.

Part 2 - Similarity measure

Euclidean distance was chosen as a measure of similarity, which is the mathematical representation of the length of the line segment between two points. Two independent weight (length) matrices were produced. The first matrix (w_ij) contains the distances between labeled - unlabeled points, necessary values to include in calculations to accurately assign labels to unlabeled points and capture the correct cluster group based on known labels. The second vector (w_ij_bar) contains distances between unlabeled to other unlabeled points, and acts as a regularization or smoothing term. Minimizing distances between unlabeled points aids in maintaining cluster purity and keeping similar labels grouped together. A section of each vector is shown below.

```
Example of term 1 weight matrix:
[[0.5911015  0.47168058 0.5652034  0.65660175 0.90455216]
 [0.77665882 0.39306182 0.74850003 0.86564688 1.06836155]
 [0.3960064  0.5629214  0.37449115 0.43637443 0.71634324]
 [0.79416213 0.20885109 0.76576234 0.90929234 1.03822761]
 [0.8539729  0.42180128 0.82566116 0.94527795 1.1413614 ]]
Shape of term 1 weight matrix: (100, 400)


Example of term 2 weight matrix:
[[0.         0.64731666 0.02867381 0.13503711 0.32034125]
 [0.64731666 0.         0.62039156 0.77411791 0.85652607]
 [0.02867381 0.62039156 0.         0.15759034 0.34292668]
 [0.13503711 0.77411791 0.15759034 0.         0.32835871]
 [0.32034125 0.85652607 0.34292668 0.32835871 0.        ]]
Shape of term 1 weight matrix: (400, 400)
```

Figure 2. Example calculations for the two distance matrices. Note the difference in shape for each matrix, reflecting the two different subsets of data: 100 labeled points and 400 unlabeled points.

The second matrix is symmetric and has duplicate values (i.e. ij = ji), which will be taken into account with the loss function.

Part 3 - Loss function

The loss function (Figure 3a) and the first derivative of the loss function (Figure 3b) are shown below. The loss function incorporates the Euclidean distance calculations for both labeled/unlabeled (term 1) and unlabeled/unlabeled points (term 2).

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

b) $$2 * \sum_{i=1}^{l} w_{ij}(y^j - \bar{y}^i) + 2 * \sum_{i=1}^{\mu} \bar{w}_{ij}(y^j - y^i)$$

Figure 3. The loss function (a) and its first derivative (b) with respect to y^j used within the semi supervised learning model is defined above.

The methods for point generation, similarity measure, and loss function as shown above, provide the basis for the following algorithms.

Part 4 - Gradient descent implementations

Gradient descent is an optimization algorithm that utilizes a loss function to measure how far the predicted labels are from the actual labels, and additionally in this case, how similar unlabeled

data point predictions are. Gradient descent works to minimize this loss function by adjusting weights and therefore driving improved label predictions.

4a - Fixed step-size gradient descent

First, we start with the simplest implementation of gradient descent with a fixed step size, α, set to 5e-4. At each iteration, the current loss and the gradient across each unlabeled point is calculated. The label predictions, initially randomly assigned with uniform probability between -1 and 1, are updated at each step. Figure 4a shows the initial randomized label assignments in the 2D space. Successful convergence, or a stopping criteria, was defined as the gradient norm falling below 0.1, indicating a stable minimum was reached and there are not severe fluctuations in variables. The gradient norm is the metric that characterizes the “flatness” of the loss function surface and is calculated with the `np.linalg.norm` function applied to the gradient. The results of label assignment are shown in Figure 4b and the minimization of loss function is shown in Figure 5.

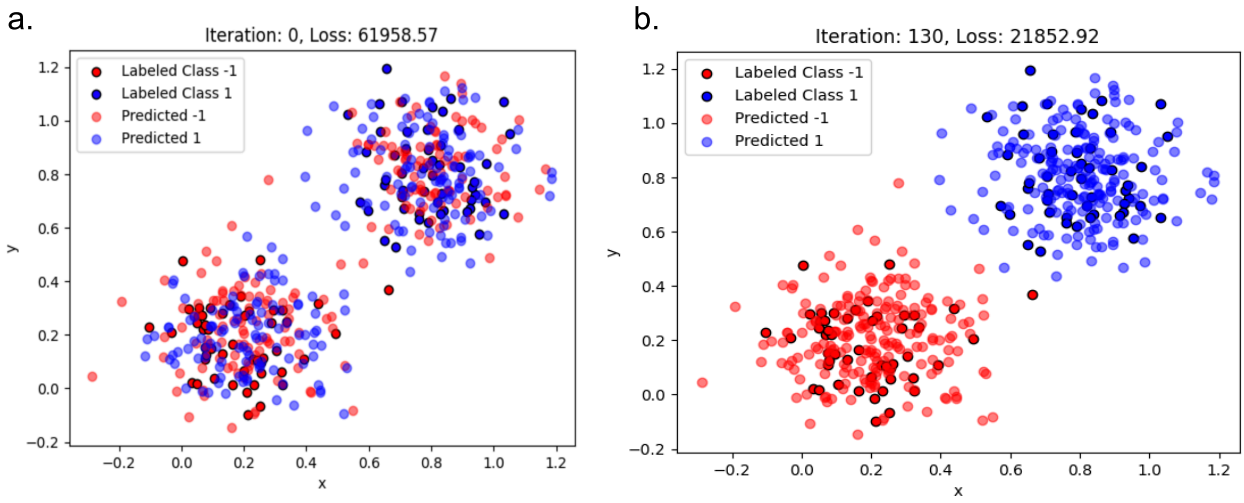

Figure 4. Gradient descent with a fixed step size took 130 iterations to reach an acceptable gradient norm, with correctly classified points as shown above.

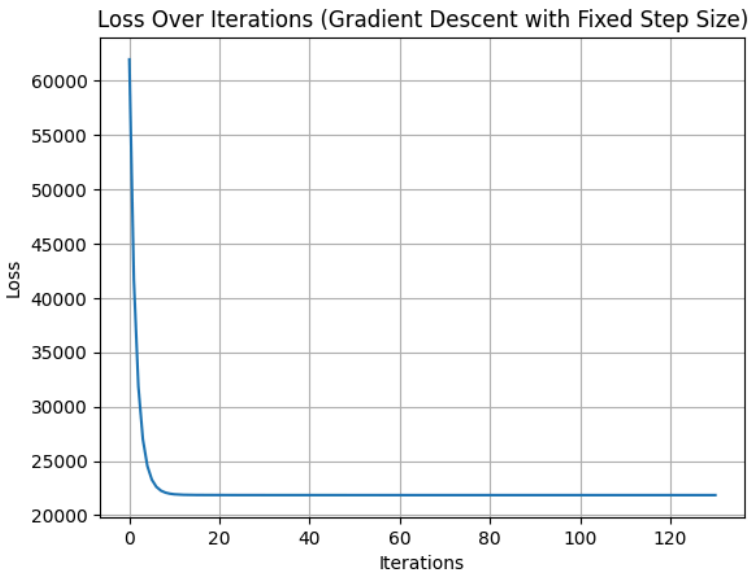

Figure 5. This graph displays the loss value at each iteration. As iteration increases, loss decreases until it reaches a plateau. Even though loss stabilized around iteration 10, a stable gradient norm was not reached until iteration 130. This indicates a flat but non-optimal region in the loss function, where variables are still being adjusted even though the label predictions are not impacted.

Gradient descent with a fixed step size is a very simple implementation of an optimization algorithm, although most likely inefficient. The step size is chosen in advance and it is not specifically tailored to the problem, often leading to too small or too large of step sizes. With an undersized step size, convergence can take a long time and use a lot of computational power. Conversely, oversized step sizes can overshoot the optimum or never converge.

This fixed step size implementation took a higher number of iterations compared to Gradient Descent with Armijo rule step-size, which can be attributed to a non-optimal step size, converging slower than necessary. Iterations to convergence could be improved with tuned step sizes or a different optimization paradigm like block coordinate descent.

4b - Gradient descent with Armijo rule step-size

Grid search for delta and gamma parameters

Gradient descent has many variations suited to certain applications and selecting a step size is an important decision in regards to convergence and convergence rate. The Armijo rule is a strategy for dynamic step size, where the step size updates as the optimization progresses. First, a grid search is performed to find ideal delta and gamma values for the Armijo rule equation. A range of prospective delta and gamma values were chosen [0.3, 0.5, 0.8] and [0.0001, 0.1, 0.5], respectively. Each combination of these values is tested with Gradient Descent and the resulting loss and iteration count is compiled in the table below.

Within this grid search, the reciprocal of the Lipschitz constant is used as a starting point for alpha. The Lipschitz is estimated by the largest eigenvalue of the Laplacian matrix of the similarity matrix, and was calculated to be 769 in this case. The Lipschitz constant is a useful optimization concept that provides an upper boundary about how fast a function's output changes. It describes the “steepness” of the function and can be applied to make reasonable step size estimations that allow convergence. Therefore, the initial step size is set to 1/L or 0.0013. Additionally, a more lenient stopping criteria was used for this particular exercise, as it was a rough search for ideal parameters. Instead of the gradient-norm based stopping criteria used in all other instances of gradient descent here, a loss-based stopping threshold of 1e-2 was implemented instead to speed up convergence.

	Delta	Gamma	Loss	Iterations
0	0.3	0.0001	21852.938648	19
1	0.3	0.1000	21852.938648	19
2	0.3	0.5000	21852.938648	19
3	0.5	0.0001	21852.938648	19
4	0.5	0.1000	21852.938648	19
5	0.5	0.5000	21852.938648	19
6	0.8	0.0001	60280.654653	100
7	0.8	0.1000	21852.938617	97
8	0.8	0.5000	21852.937575	27

Table 1. This table compiles grid search results to the purpose of selecting appropriate delta and gamma values for the Armijo rule. The combination that leads to the lowest number of iterations is selected, and in this case multiple values lead to the same number of iterations.

Delta represents the loss decrease that is acceptable. A low delta indicates a lower threshold for a step to be implemented and therefore more steps are accepted. The other variable in the grid search, gamma, represents how fast the step size is shrunk. A low gamma means that the step size shrinks very quickly and progress is slowed, representing a more cautious approach. As Table 1 shows, multiple combinations of delta and gamma result in the minimum number of iterations (19). As such, the middle value for delta was chosen (0.5) and the largest value for gamma was chosen (0.1) to allow for a more aggressive approach while still within the set of best performing pairs.

Implementation of optimized delta and gamma

This implementation of Gradient Descent with the Armijo rule step-size is very similar to Gradient Descent with a fixed step-size, with the key difference in the addition of the following

block of code, where you can see how delta and gamma are used to dynamically update step size.

```
while loss_function(n_labeled, n_unlabeled, w_ij, w_ij_bar, y_labeled, y -
alpha * grad) > current_loss - delta * alpha * (grad_norm ** 2):
                alpha *= gamma
```

This inequality compares a new loss versus the previous loss minus the minimum expected loss (proportional to delta). When the inequality fails, the algorithm has moved too far or moved in the wrong direction, and so the step size is shrunk (proportional to gamma). This is where you can see the dynamic progression of step size, beginning with a large alpha, and shrinking at every inequality failure.

The randomized set of points used to test this method is equal to the one used in the gradient descent, fixed step size case, seen in Figure 4a. The results from this algorithm using optimized Armijo parameters, are shown in Figure 6, where it is demonstrated that the data points are accurately classified, and in Figure 7, where the loss function over iteration is shown. The algorithm converged in 40 iterations, faster than the Gradient Descent with a fixed step size.

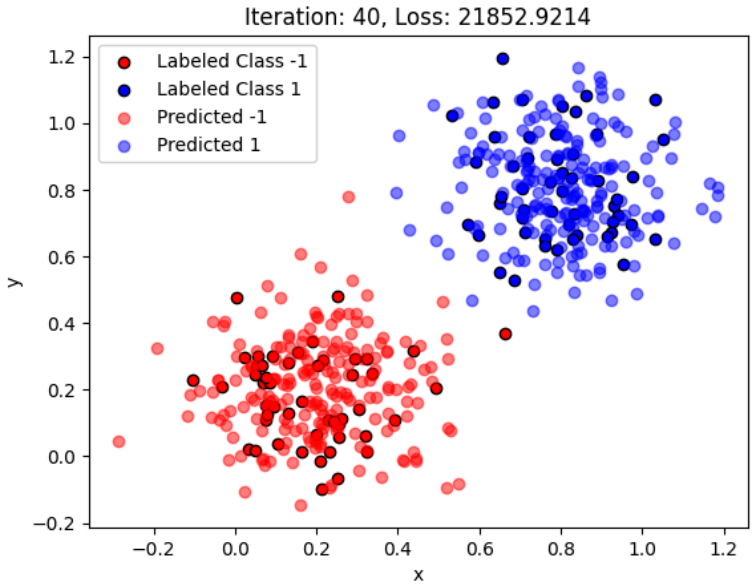

Figure 6. Classification results from Gradient Descent with an Armijo rule step size. The algorithm converged in 40 iterations and correctly classified points, as shown above.

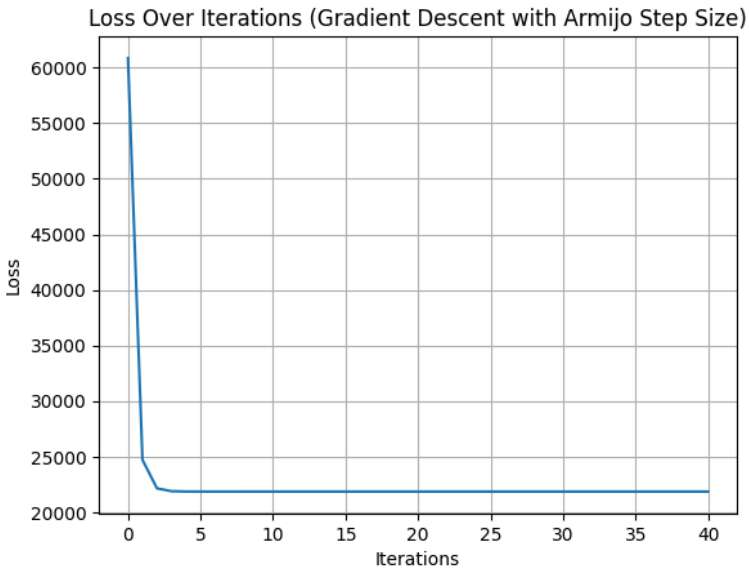

Figure 7. Loss over iteration plot for the Gradient Descent with Armijo step size. This algorithm converged faster than Gradient Descent with a fixed step size.

This Gradient Descent algorithm with the Armijo step size rule demonstrates how dynamic, optimized step sizes can improve convergence rate.

4c - Block coordinate gradient descent

Block coordinate gradient descent (BCGD) with Gauss-Southwell (GS) selection represents a fundamental difference in paradigm from the two previous gradient descent implementations. Instead of updating all weights at once, this algorithm aims to address the “worst” points first. The Gauss-Southwell strategy selects blocks of data points with the largest gradients to update, resulting in a targeted and efficient approach to optimization. This strategy is typically the most effective when the problem being optimized has high dimensions, i.e. a large number of parameters. Block sizes can be chosen as unidimensional or multidimensional depending on the nature of the data. In our case the data points are loosely correlated thus a block size with dimension 1 is used simply meaning each data point is individually considered when updating the loss function.

The code block below reveals the key paradigm differences between a block coordinate descent strategy and standard gradient descent. The gradient value at each variable is calculated, however only the maximum value is flagged and updated at each iteration.

```
    for k in range(n_unlabeled):
      # Calculate the gradient for the current variable
      max_gradient = 0
      gradient_k = derivative(n_labeled, n_unlabeled, w_ij, w_ij_bar,
y_labeled, y, k)
```

```
    # Gauss-Southwell selection (within the block of size 1)
    # Check if the current gradient is the largest within the block
    if k == 0 or abs(gradient_k) > abs(max_gradient):
      max_gradient = gradient_k
      max_gradient_index = k

    # Update only the selected coordinate within the dimension/block
    y[max_gradient_index] -= alpha * max_gradient
```

The results from implementing the BCGD with GS scheme are shown below.

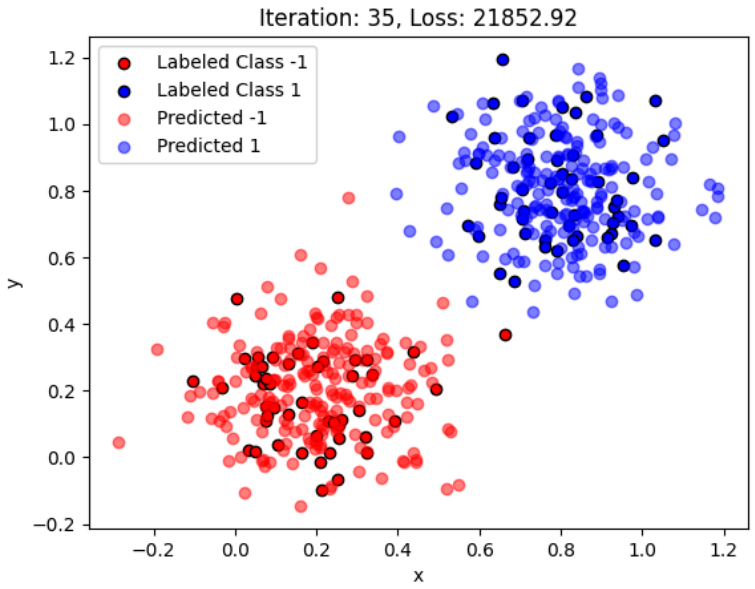

Figure 8. The final point classifications from the block coordinate gradient descent are shown above, with convergence reached in 35 iterations.

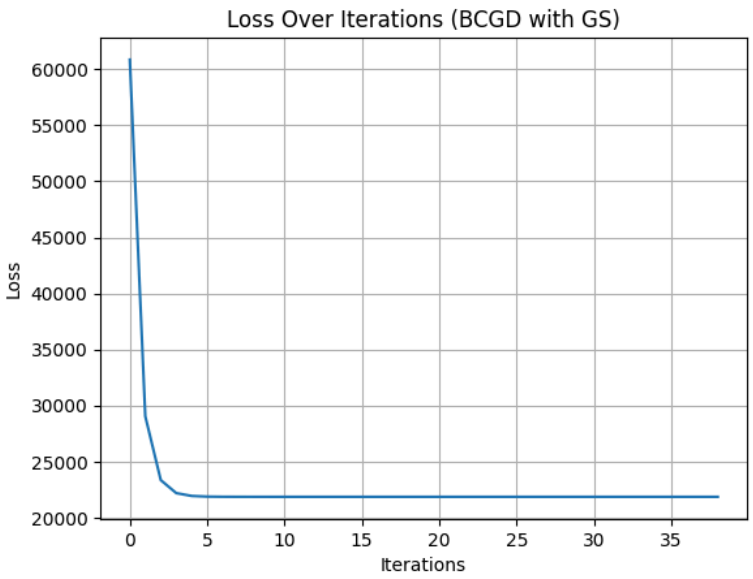

Figure 9. The loss calculations over iteration for the BCGD approach is shown above. Convergence is reached very quickly, compared to standard gradient descent or gradient descent with the Armijo rule.

4d - Comparison between gradient descent implementations

While all three gradient descent algorithms successfully converged and provided accurate label classifications, block coordinate gradient descent took the least amount of iterations to converge, followed by gradient descent with the Armijo step size, followed by gradient descent with a fixed step size. This is due to the fundamental differences of the algorithms, where dynamic step sizes (Armijo), and an efficient strategy of targeting the data points with the most error (BCGD) help to improve convergence time.

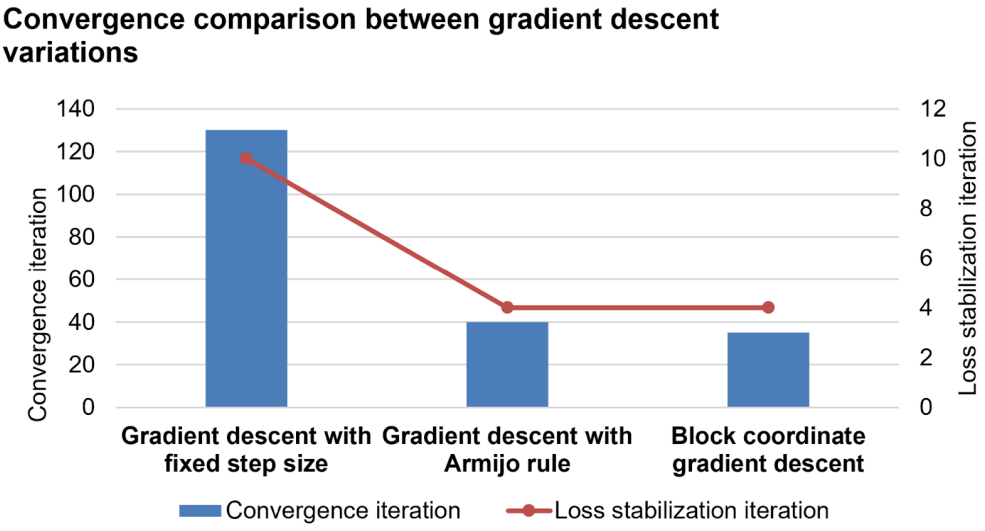

Figure 11. A comparison of the convergence rates is shown above. Convergence iteration, as defined by gradient loss norm, is coded in blue, while the approximate iteration count at which loss stabilizes is shown in red.

Part 5 - Algorithms on publicly available datasets

We chose a dataset on breast cancer, with physical descriptions of tumor size, shape, and other features linked to malignancy (-1) or benignness (1), to test our gradient descent models. The data is represented by 569 rows of tumor samples by 32 features of the samples (mean radius, mean texture, etc.). First, the data was pre-treated, cleaned, split into a training and test set, and a fraction of the data was randomly unlabeled to work with the model. The actual labels from the dataset for these points were stored in a vector to be used for the purpose of calculating accuracy in the model predictions later. The three gradient descent models as previously described, were applied and tested. The best performing model was then applied to the test set.

Preparation of data and model foundations

The dataset was loaded into the workspace into a dataframe, and a preview is shown below.

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	target	target_name
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	-1	malignant
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	-1	malignant
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	-1	malignant
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	-1	malignant
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	-1	malignant

5 rows × 32 columns

Table 2. A preview of the dataset we are working with is shown above. Thirty tumor characteristics are listed in each column, linked to breast cancer malignancy or benignness.

First, the data was organized and standardized. This consists of switching labels to -1 or 1, indicating malignant or benign tumors, respectively. MinMaxScaler was implemented to normalize all data within the feature range. The data was then split into a training and test set, with 20% of the data reserved for later validation once a gradient descent model was chosen based on the training set. The training and test datasets were each split into labeled points and unlabeled points.

Next, distance matrices were calculated for both the training and test sets. The same strategy from before was used, with a matrix for Euclidean distances between labeled and unlabeled, as well as a matrix for Euclidean distances between unlabeled and unlabeled points was created. A preview is shown below.

```
[[0.64326482 1.38395574 1.41823837 ... 1.1901971  0.82526143 0.8741139 ]
 [1.06954297 0.55987055 0.64898619 ... 0.86835653 1.00737855 1.04204537]
 [0.47818418 0.6984836  0.76403192 ... 0.68166827 0.42414451 0.47179619]
 [0.91195259 0.47623034 0.51433298 ... 0.56563042 0.99040126 1.05181038]
 [0.99570254 0.34480219 0.23850802 ... 0.52671191 1.02750194 1.08800868]]
(91, 364)
[[0.         0.87160979 0.96443908 ... 0.74381584 0.41034764 0.44087653]
 [0.87160979 0.         0.3118808  ... 0.52661225 0.91962938 0.94498459]
 [0.96443908 0.3118808  0.         ... 0.43969174 1.03919834 1.07460536]
 [0.90214474 0.2299196  0.3135799  ... 0.57186702 0.92563628 0.94573491]
 [0.58129132 0.51228269 0.56540148 ... 0.47973623 0.56758759 0.59664902]]
(364, 364)
```

Figure 10. A preview of the training set Euclidean distance matrices for I) labeled - unlabeled and II) unlabeled - unlabeled points.

The same loss function scheme was used as previously defined in Part 3 - Loss function. Additionally, a measure for accuracy was defined with the accuracy_score function from the sklearn.metrics toolkit, comparing the actual tumor labels with model predictions.

Each gradient descent strategy was tested with the training dataset and the results will be shown below. It can be noted for each model that accuracy takes a dip down initially as the machine is testing the assignment of labels, but quickly readjusts and learns by minimizing the loss function. The best performing algorithm, in terms of convergence rate, was then applied to the test set.

Gradient descent with fixed step size

The same scheme as described in 4a - Fixed step-size gradient descent was used to generate a model that produced the following results. The fixed step-size gradient descent model converged at iteration 483 with a final accuracy of 70% on the training set in 108 seconds of CPU time.

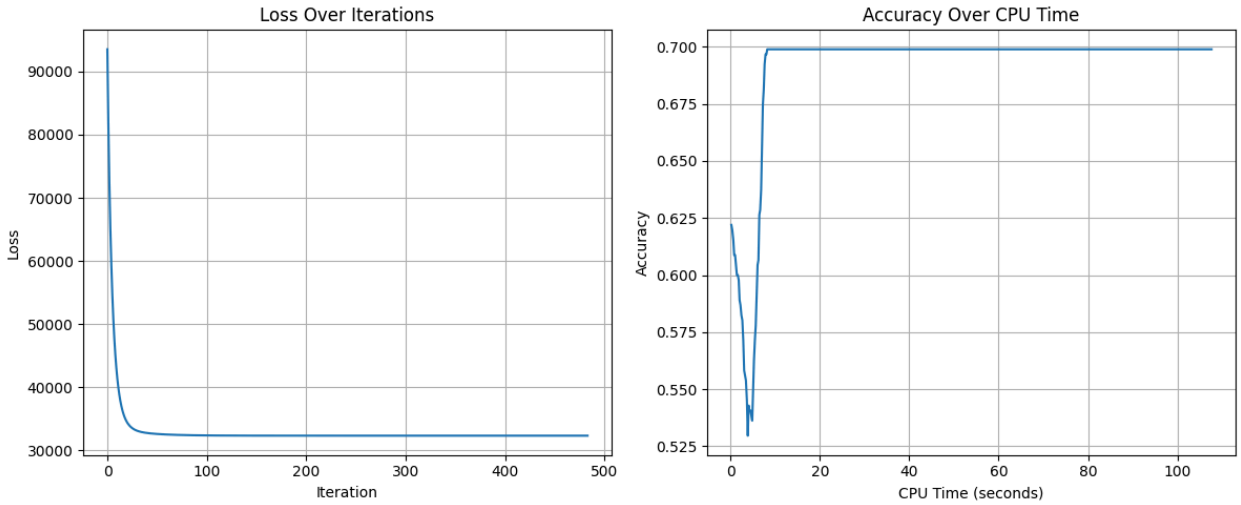

Figure 12. Loss over iteration is shown above, which stabilizes around 20 iterations. Accuracy, as shown in the second graph over CPU time, stabilizes around 10 seconds.

Gradient descent with Armijo rule step size

The same scheme as described in 4b - Gradient descent with Armijo rule step-size was used to generate a model that produced the following results. The new Lipschitz constant was calculated to be 1,976, setting the initial step size at 0.0005. The gradient descent with the dynamic step size model converged at iteration 96 with a final accuracy of 70% on the training set in 45 seconds of CPU time.

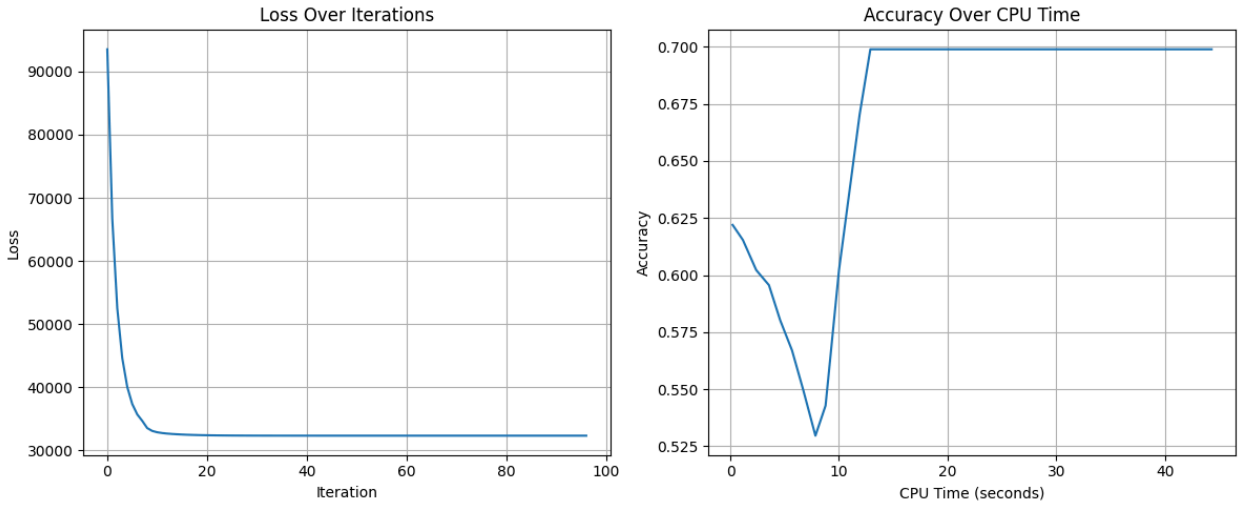

Figure 13. Loss over iteration is shown above, which stabilizes around 10 iterations. Accuracy, as shown in the second graph over CPU time, stabilizes around 13 seconds.

It is interesting to note that while the Armijo step size converges faster by most metrics as compared to the fixed step size model, the accuracy stabilizes slightly faster in the fixed step size case. This may indicate the slower overhead of the Armijo calculations and update rule for the step size as opposed to a fixed step size that requires no additional calculations or computational time.

Block coordinate gradient descent with Gauss-Southwell selection

The same scheme as described in 4c - Block coordinate gradient descent was used to generate a model that produced the following results. The block coordinate gradient descent model converged at iteration 75 with a final accuracy of 70% on the training set in 29 seconds of CPU time.

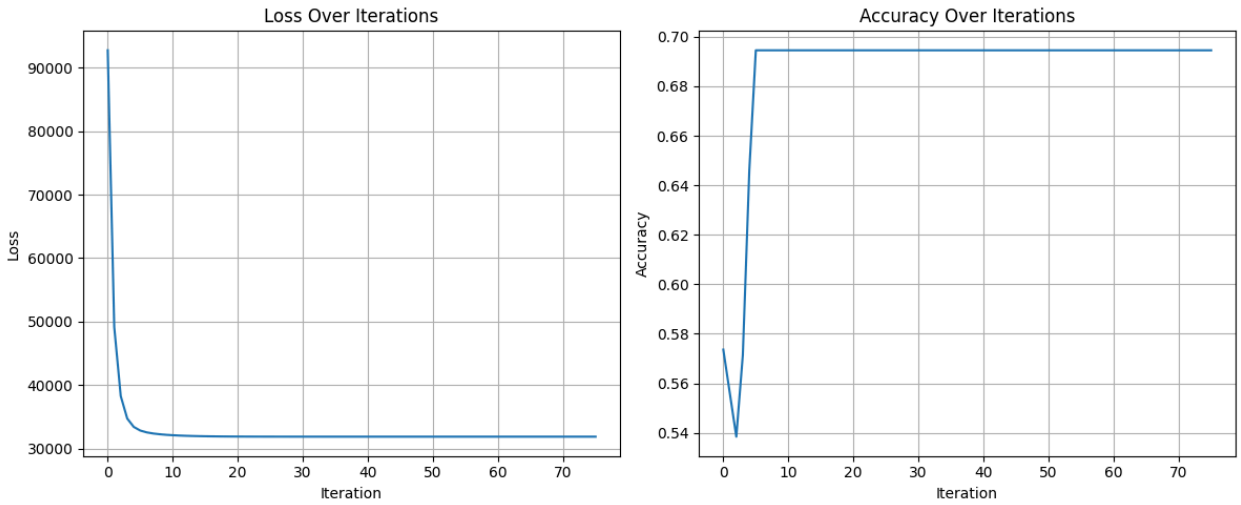

Figure 14. Loss over iteration is shown above, which stabilizes around 5 iterations. Accuracy, as shown in the second graph over CPU time, stabilizes around 5 seconds as well.

Application to test set

The three gradient descent variations applied to the training set were compared in terms of convergence iteration as well as stabilization time below.

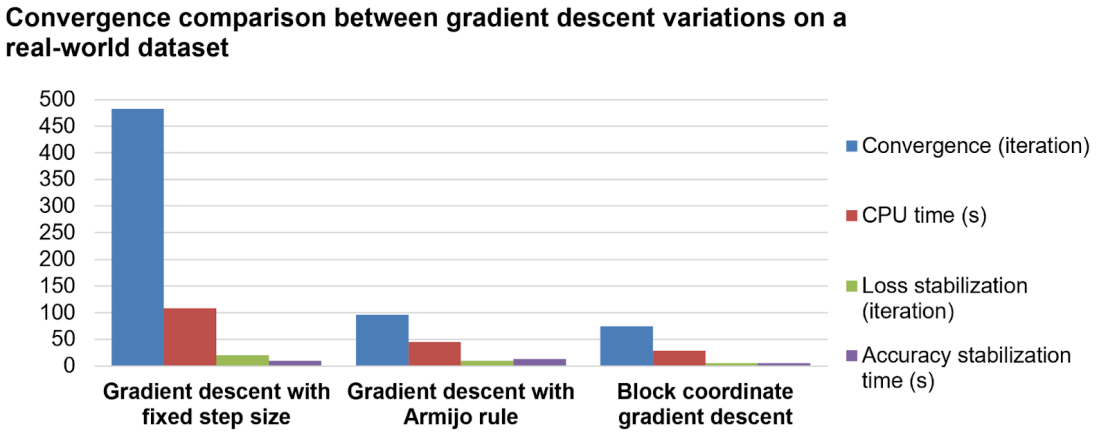

Figure 15. Convergence iteration and stabilization time is directly compared for each gradient descent strategy. Block coordinate gradient descent with Gauss-Southwell selection is the clear winner across every metric. As such, it was chosen for the test set model.

Block coordinate gradient descent proved to be the most efficient algorithm for this particular application, with the lowest convergence iteration, CPU time, loss stabilization iteration, and accuracy stabilization time. This matches our early findings on the synthetic data exercise, where block coordinate gradient descent was also the most efficient. BCGD is also likely the best model as the dataset we are working with has a high number of features (32 in total), making coordinate-wise updates particularly effective, as optimizing one feature at a time allows for more targeted and efficient convergence. For this reason, this algorithm was chosen for application to the test set.

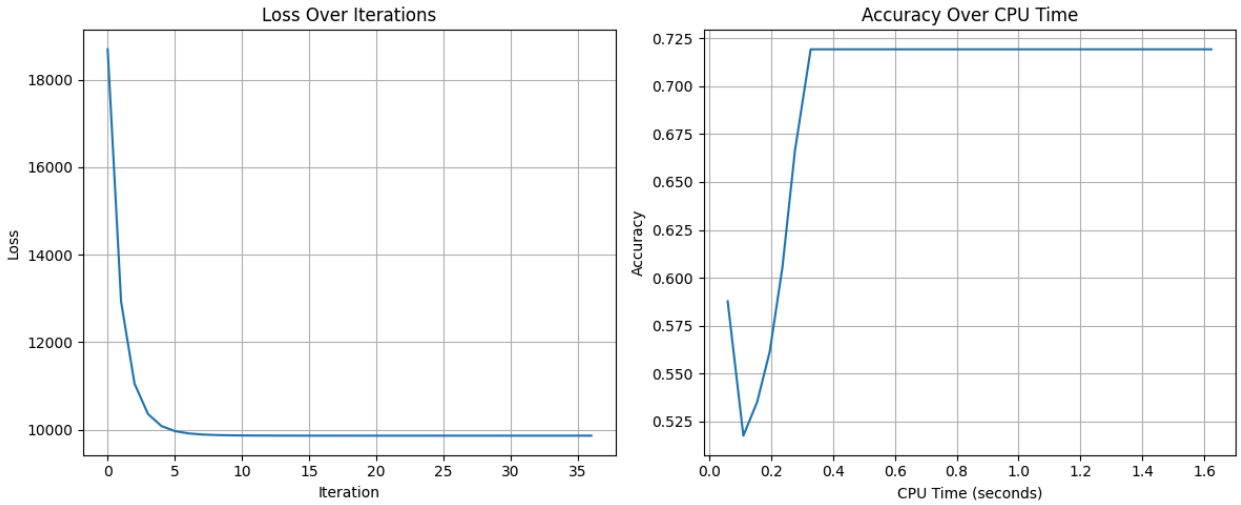

Figure 16. Loss over iteration is shown above, which stabilizes around 6 iterations. Accuracy, as shown in the second graph over CPU time, stabilizes around 0.3 seconds.

The block coordinate gradient descent model applied to the test set converged at iteration 36 with a final accuracy of 72% in only about 2 seconds of CPU time. This quick convergence is attributed to the small test size, having only 114 rows, and therefore optimization is speedy. This machine learning model proves to be a predictive and optimized tool in forecasting the malignancy of breast cancer tumors given certain physical characteristics.

Conclusions

An exercise in gradient descent algorithms, including fixed step size gradient descent, gradient descent with the Armijo rule, and block coordinate gradient descent with Gauss-Southwell selection, was conducted and applied to a synthetic dataset as well as publicly available breast cancer data. For the synthetic dataset, points were generated, a similarity measure was defined, and a loss function was implemented. Each gradient descent algorithm was applied and coordinate gradient descent was found to be the most efficient, converging in approximately 35 iterations. For the breast cancer dataset, the data points were imported and standardized. Again, the three algorithms were applied and block coordinate gradient descent was found to be the most efficient with convergence at 75 iterations. The final model evaluated on the test set had an accuracy of 72% and a convergence time of 2 seconds.