Coordinate Descent Methods for Logistic Regression: A Comparative Study on the Wine Dataset

Allen Zou

February 26, 2024

Abstract

Logistic regression is a widely employed method for solving binary classification problems, often necessitating iterative optimization techniques. This study explores the application of coordinate descent methods to logistic regression, with a specific focus on the Momentum Coordinate Descent algorithm. A comparative analysis is conducted with a Random-Feature Coordinate Descent approach, examining convergence properties. Experimental results are presented to showcase the efficiency and behavior of the Momentum Coordinate Descent algorithm in comparison to Random-Feature Coordinate Descent. Additionally, we extend the Momentum Coordinate Descent algorithm into a k-sparse setting and assess its performance across varying k values.

1 Introduction

Logistic regression serves as a foundational technique for binary classification, often requiring iterative optimization strategies for model refinement. This study delves into coordinate descent methods, with a specific focus on the application of the Momentum Coordinate Descent algorithm to logistic regression. As an alternative to traditional optimization techniques, the simplicity of coordinate descent warrant a comprehensive investigation.

2 Coordinate Descent Strategies

Coordinate descent involves updating one variable of the weight vector at a time while holding others fixed. In the context of logistic regression, the goal is to iteratively adjust the coefficients to minimize the cost function. This study focuses on two coordinate descent algorithms: Random-Feature Coordinate Descent and Momentum Coordinate Descent.

Random-Feature Coordinate Descent merely updates a randomly chosen feature in each iteration with the negative gradient. Momentum Coordinate Descent, however, not only randomly selects a feature but also updates the weight vector by incorporating a momentum term. The introduction of momentum allows the algorithm to accumulate information from previous iterations. This choice is particularly relevant when considering the history of gradient updates to overcome potential oscillations or slow convergence.

When determining the step size (α), both algorithms integrate backtracking line search with the Armijo condition. The purpose of the line search is to dynamically adjust the step size in each iteration, facilitating an efficient convergence towards the optimal solution.

The Armijo condition ensures the chosen step size meets the desired reduction in the objective function. Here, w is the weight vector, d is the search direction, and c is a constant. This condition guides the algorithm in making step size decisions that contribute to effective convergence. It is defined as:

$$f(w + \alpha \cdot d) \leq f(w) + c \cdot \alpha \cdot \nabla f(w)^T d.$$

2.1 Pseudocode

Algorithm 1 Random-Feature Coordinate Descent

```
Require: Initial weights $w$, Dataset $X$, Labels $y$,
    Learning rate $\gamma$, Number of iterations $T$
1: for $t = 1$ to $T$ do
2:     $dw \leftarrow$ Compute $\frac{\partial L}{\partial w}$ using $w$
3:     $i \leftarrow$ Randomly choose from $[0, d)$
4:     $\alpha \leftarrow$ line_search($w$, $dw$, $i$)        ▷ Step size
5:     $w_i \leftarrow w_i - \alpha \cdot dw_i$
6: end for
7: return $w$
```

Algorithm 2 Momentum Coordinate Descent

```
Require: Initial weights $w$, Dataset $X$, Labels $y$,
    Learning rate $\gamma$, Number of iterations $T$
 1: Initialize $m$ to all $0$                    ▷ Momentum term
 2: $\beta \leftarrow 0.95$    ▷ Constant for momentum update
 3: for $t = 1$ to $T$ do
 4:     $i \leftarrow$ Randomly choose from $[0, d)$
 5:     $dw_i \leftarrow$ Compute $\frac{\partial L}{\partial w_i}$
 6:     $m = \beta * m + (1 - \beta) * dw_i$
 7:     $\alpha \leftarrow$ line_search($w$, $dw$, $i$)        ▷ Step size
 8:     $w_i \leftarrow w_i - \alpha \cdot m_i$
 9: end for
10: return $w$
```

Algorithm 3 Backtracking Line Search

```
Require: Weights $w$, Gradients $dw$, Coordinate $i$
 1: $\alpha \leftarrow 1.0$
 2: $c \leftarrow 10^{-4}$      ▷ constant for Armijo condition
 3: while True do
 4:     $old\_loss \leftarrow$ Compute log loss using $w$
 5:     $w' \leftarrow$ deep copy $w$
 6:     $w_i^{'} \leftarrow w_i^{'} - \alpha \cdot dw_i$
 7:     $new\_loss \leftarrow$ Compute log loss using $w'$
 8:     $armijo \leftarrow c \cdot \alpha \cdot ||dw||^2$
 9:     if $new\_loss \leq old\_loss - armijo$ then
10:         break
11:     end if
12: end while
13: return $\alpha$
```

2.2 Assumptions About Convergence

The coordinate descent algorithms presented here assume the differentiability and convexity of the loss function $L(w)$. The differentiability of the loss function is essential for computing gradients, which are necessary for weight updates in each iteration. Additionally, the convexity property ensures the well-behaved nature of the optimization problem associated with logistic regression on the wine dataset.

In the context of logistic regression, the convex logistic loss function is characterized by a single global minimum. The algorithms, including Random-Feature Coordinate Descent and Momentum Coordinate Descent, rely on the convexity of the loss function to efficiently find the global minimum. Convexity facilitating convergence without getting stuck in local optima.

It's important to note that these assumptions may be influenced by the choice of dataset and specific characteristics of the logistic loss function. While these conditions hold for logistic regression on the wine dataset, the convergence behavior could vary in the presence of non-convex loss functions or datasets with different properties.

3 Experimental Setup

For this study, the Wine dataset was employed, initially consisting of 178 data points across 13 dimensions and belonging to 3 classes. To create a binary classification problem, the dataset was narrowed down to include only the first two classes, resulting in 130 data points—59 from the first class and 71 from the second. An 8:2 split was applied for training and testing.

Both Momentum and Random-Feature Coordinate Descent algorithms were executed with 1,000 runs each, where each run encompassed 50 iterations. The choice of a relatively low number of iterations was motivated by the observed rapid convergence of the algorithms. The resulting plots showcase the average function evaluations over iterations, providing a robust overview of algorithm behavior.

Hyper-parameters were selected based on empiricism to ensure meaningful comparisons. For coordinate descent, a learning rate of 0.01 was employed. In the context of backtracking line search, the constant in the Armijo condition was set to 10^{-4}, and the beta parameter, responsible for scaling the predicted step size in updating the momentum, was chosen as 0.95 for Momentum Coordinate Descent.

4 Experimental Results

We present experimental results for the Momentum and Random-Feature Coordinate Descent algorithms. The analysis focuses on the first two classes of the Wine dataset, where the mean function evaluation and associated variability for each iteration are depicted in the figure below. The shaded regions around each line represent one standard deviation from the mean function evaluation. For comparison, the loss from scikit-learn's standard logistic regression solver is plotted as a dotted line on the same graph, providing a benchmark for evaluating the performance of the coordinate descent algorithms.

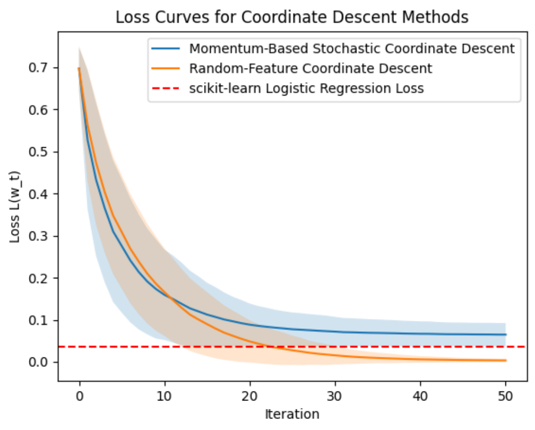

Figure 1: Comparison Between Greedy Coordinate Descent and Random-Feature Coordinate Descent

5 Discussion

In this section, we analyze and interpret the experimental results obtained from the Momentum and Random-Feature Coordinate Descent algorithms applied to logistic regression on the first two classes of the Wine dataset.

Contrary to our initial expectations, the Momentum Coordinate Descent strategy did not outperform scikit-learn's standard logistic regression solver nor Random-Feature Coordinate Descent in terms of the final loss value. Instead, Random-Feature Coordinate Descent surpassed the momentum-based method in terms of the loss it converged to. A small thing to note is that the Momentum Coordinate Descent method exhibited slightly faster convergence in the initial iterations compared to the Random-Feature Coordinate Descent. However, the latter eventually caught up, suggesting that the advantage in early convergence is not sustained.

Interestingly, the variance of the Momentum Coordinate Descent method was found to be larger than that of the Random-Feature Coordinate Descent. Additionally, around the points of convergence, the variance almost disappears for the Random-Feature method, indicating a more consistent convergence to a specific location. In contrast, the Momentum-Based Coordinate Descent method retained some level of variance even after convergence, suggesting that it does not always converge to the same location.

These unexpected findings have implications for the practical use of the Momentum Coordinate Descent strategy in logistic regression. While it may exhibit faster convergence in the early stages, its tendency to have larger variance and not consistently converge to the same location is unusual. The presence of variance in the Momentum-Based Coordinate Descent, even after convergence, seems counterintuitive to the assumption of convexity. Convex optimization problems are generally expected to converge to a unique minimum due to the well-behaved nature of convex functions. Further investigation into the behavior of the Momentum Coordinate Descent, considering the convexity assumption, may be necessary to reconcile this discrepancy. Fine-tuning of hyperparameters and a more in-depth exploration of the optimization landscape could shed light on the observed behavior.

6 K-sparse Coordinate Descent

In this section, we extend our analysis to explore the performance of the Momentum-based Coordinate Descent algorithm under sparsity constraints. The algorithm is modified to enforce a k-sparse solution, where the weight vector has at most k nonzero entries. We investigate the impact of varying sparsity levels on convergence behavior and final loss values.

6.1 Experimental Setup

For the k-sparse experiments, we utilized the same Wine dataset with binary classification between the first two classes as in the previous section. Hyperparameters such as the learning rate (0.01) and backtracking line search with an Armijo condition remained consistent with our earlier experiments.

The key modifications include a decrease in the number of iterations to 25 per run, adapting to the faster convergence behavior observed in the k-sparse setting. Unlike the previous experiments, where a single coordinate was updated in each iteration, the k-sparse experiments involved updating a randomly selected coordinate with the Momentum-based Coordinate Descent algorithm.

Additionally, the method for evaluating and plotting values entailed the following steps: after each iteration of updating the weight vector, we extracted its k-sparse version. The loss was subsequently computed based on this k-sparse variant of the current weight vector. This approach allowed us to gain insights into the model's behavior throughout the optimization process.

We conducted experiments for various sparsity levels, specifically testing for k values of 1, 3, 7, 10, and 13, to explore the effects of different degrees of sparsity on the convergence behavior.

6.2 Experimental Results

In this section, we present experimental results for the Momentum and Random-Feature Coordinate Descent algorithms applied to logistic regression under various k-sparsity values (k = 1, 3, 7, 10, and 13). As in the previous experiments, the figure below illustrates the mean function evaluation along with the associated variability for each iteration. The shaded regions surrounding each line indicate one standard deviation from the mean function evaluation.

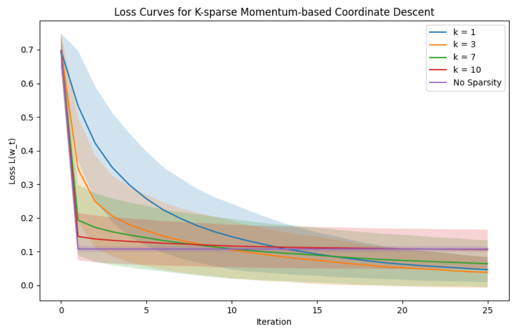

Figure 2: K-sparse Momentum-based Coordinate Descent for Different k Values

6.3 Discussion

k Value	Final Loss
1	0.046272
3	0.038545
7	0.064082
10	0.106584
13	0.107748

Table 1: Final Loss for Different k Values in k-sparse Coordinate Descent

In the k-sparse setting, it is evident that all runs converge significantly faster. This is expected given the modification to the algorithm where more than one coordinate is updated each iteration.

However, there is an unusual trend in the relationship between the sparsity level (k) and the optimization process. Higher values of k now lead to faster convergence, aligning with expectations. Surprisingly, they also result in higher final loss values. This unexpected outcome challenges the assumption that higher sparsity levels would consistently result in lower final loss values. Further investigation into the implementation may be required.

An interesting observation is the presence of groups of k values that exhibit similar performance. For instance, k=1, k=3, and k=7 demonstrate comparable loss values, forming one group. In contrast, k=10 and k=13 constitute another group, with the smaller of the two k values exhibiting a slightly lower final loss. This finding introduces a perspective on the trade-offs associated with sparsity levels, suggesting that excessively high sparsity might not always be beneficial. However, as mentioned previously, it's essential to note that this observation assumes the absence of implementation errors or other confounding factors. Further investigation into the algorithm's behavior is warranted to validate these unexpected outcomes.

7 Limitations

While our study focuses on a binary classification setting, one avenue for future research involves extending our methodology to scenarios with a higher number of classes. The decision to restrict our analysis to classes 1 and 2 was driven by the available dataset, resulting in 59 and 71 points, respectively. Recognizing the potential impact of training on a limited number of points, it is essential to acknowledge that generalization may be constrained. Future work should explore datasets with more extensive class distributions to assess the scalability and robustness of the proposed Momentum-based Coordinate Descent method. While our study provides insights into the Momentum-based Coordinate Descent method, the aforementioned considerations serve as avenues for refinement and expansion in subsequent research efforts.

8 Conclusion

In this study, we delved into the application of coordinate descent methods, with a specific emphasis on the Momentum Coordinate Descent algorithm in logistic regression optimization. Our comparative analysis included a Random-Feature Coordinate Descent approach, providing insights into the convergence properties of these methods.

The experimental results revealed nuanced dynamics in the performance of the Momentum Coordinate Descent algorithm. While it exhibited faster convergence in the initial iterations, its overall performance, in terms of the final loss value, did not surpass that of scikit-learn's standard logistic regression solver or the Random-Feature Coordinate Descent. The presence of variance in the Momentum-based method, even after convergence, raises interesting questions about its behavior in relation to the convexity assumption.

Furthermore, our exploration extended to the k-sparse setting, where the Momentum-based Coordinate Descent algorithm was modified to enforce sparsity constraints. The results introduced a trade-off between sparsity levels and convergence speed, challenging expectations that higher sparsity always leads to lower final loss values.

As we conclude, it's essential to acknowledge the limitations of our study, including the binary classification setting and the relatively limited number of data points. Future research could extend our methodology to scenarios with a higher number of classes and explore datasets with more extensive class distributions to enhance generalizability. Additionally, further investigation into the implementation details is warranted to scrutinize potential factors contributing to the observed dynamics and unexpected behaviors.

Looking ahead, I anticipate further exploration across diverse datasets, conducting more parameter tuning, and personally delving into alternative selection strategies. These efforts are intended to refine and broaden the applicability of coordinate descent methods, addressing the limitations identified in this study.