Coordinate Descent

Vaidehi Gupta
v2gupta@ucsd.edu

1 Coordinate Descent

Coordinate descent algorithms have recently been in use due to its higher efficiency to solve optimization problems by successively minimizing along each coordinate or coordinate hyperplane [3]. It is quite similar to gradient descent except for the fact that, instead of moving all the coordinates along the gradient direction, at each iteration, only selected coordinate(s) are changed by the coordinate descent [2]. The selected coordinate could be a single coordinate or even a batch of coordinates (BCD - Batch Coordinate Descent).

For the minimization problem stated here,

$$\min_w L(w) = L(w_1, w_2, ..., w_d)$$

where $w \in \mathbb{R}^d$ and the function $L\colon \mathbb{R}^d \to \mathbb{R}$ is a continuous, differential convex function.

The basic coordinate descent (CD) framework works as follows: We select a index i and thereby coordinate w_i and update only this coordinate, using some scheme, while keeping the rest exactly the same. This calls for two questions:

1. **Which coordinate to choose?**
   There are multiple approaches to choose an index and updating the selected coordinate accordingly. Some of these are to select components cyclically, select a component at random at each iteration (not necessarily with equal probability). Lastly, we can choose components greedily, choosing the component corresponding to the greatest descent, strongest descent potential, or other scores, at the current iteration.
   In this project, we will be adopting the last mentioned approach which is the greedy approach. These methods choose the index i such that the loss function is minimized the most in that direction. Specifically, we are using the $Gaussian-Southwell$ Selection rule (GS) [2] that chooses i_k such that gradient of the chosen block of components is the greatest. This is given by:

   $$argmax_{1\le j\le s}||\nabla_j L(w^{k-1})||$$

   Also,

   $$w_i{}^k = argmin_{w_i} L(w_1{}^{k-1}, w_2{}^{k-1}, ..., w_d{}^{k-1})$$

   If this block is an individual component, then the above formula reduces to absolute value function. Even though in most cases, the GS rule converges faster than the random selection, but the per-iteration complexity is higher.

2. **How to set the new value of w_i?**
   We'll need to suppose that L is a differentiable function for this step. The updates can be made by applying the coordinate-gradient descent along each chosen component, i.e.

   $$w_i{}^k = w_i{}^{k-1} - \eta_i \nabla_i L(w^{k-1})$$

   where η_i represents the step-size (or learning rate) at each iteration. The step-size can be set constant or can also be adapted in accordance to the gradient value using the backtracking line search approach [1].

   The backtracking line search adaptively chooses the step-size as follows:
   We start with $t = 1$ and fix $0 < \rho < 1$ and while:

   $$L(w + \alpha\Delta w) > L(w) + \alpha c \nabla L(w)^T \Delta w$$

   In our case, $\Delta w = -\nabla L(w)$

   $$\implies L(w - \nabla L(w)) > L(w) - \alpha c ||\nabla L(w)||^2$$

update $\alpha = \alpha * \rho$
After termination: $\eta_i = \alpha$. Here, $0 < c < 1$ represents the strictness of convergence. We finally get the step-size that adapts with each iteration.

For the coordinate descent method adopted in this report, we assume that the cost function $L(w)$ has to be a continuous and also differentiable function. It is not necessary for the function to be convex but it will be more smoother if the function we are trying to minimize is a convex function as well.

2 Convergence

The coordinate descent method used in this project uses the greedy approach to select the coordinate that returns the largest gradient value. That is, it picks the coordinate that minimizes the loss function the most. This means that the loss value will surely decrease with every iteration. Also, the step-size will take care that this loss value eventually converges to the optimal loss value $L*$.
Additionally, with the assumption that the loss function, $L(w) \colon \mathbb{R}^d \to \mathbb{R}$ is convex and differentiable, and its gradient is *Lipschitz* continuous with constant $L > 0$. Then running the gradient descent with a constant step-size t for k iterations is guaranteed to converge with a rate of $O(1/k)$ [5]. This can also be seen in the plot that the convergence has been achieved.

Also, with an increased number of iterations and a good step size, the loss is expected to converge. Initially, with a step-size $\eta = 0.0001$ and iterations = $1 million$, it was not converging because of the too small step-size. Similarly, with a higher step-size $\eta = 0.01$ and smaller number of iterations = $10,000$, the convergence was still not happening due to less number of iterations. Therefore, with the correct balance of not too small or too high step-size and iterations, we were able to converge to the optimal loss value $L*$.

3 Experimental Results

In this section, a comparison between the two approaches for selection of coordinate will be discussed and their performance will be compared against the optimal logistic regression solver.

The loss function that is being considered in this project for logistic regression is:

$$\min_w L(w) = \sum_{j=1}^{m} log(1 + exp[-y_j w^T x_j])$$

Its gradient is calculated as follows:

$$\Delta L(w) = \sum_{j=1}^{m} \frac{-y_j exp[-y_j w^T x_j]}{1 + exp[-y_j w^T x_j]} x_j$$

3.1 Standard logistic regression solver

The standard logistic regression solver from the *sklearn* library has been used as the benchmark logistic regression model. When the classifier was run on the wine dataset, the optimal loss value calculated by the *log_loss* function came out to be:

$$L* = 0.000389$$

It was important that this loss value is not regularized for a fair comparison with the other approaches. However, the regressor from the *sklearn* library has regularization term involved by default. To make sure, the loss value we got is not regularized, the hyper-parameter C has been set to a very high value (magnitude of 10^6) such that regularization does not have any effect.

We will be using this $L*$ value as the optimal loss and benchmark against the two coordinate descent methods we are going to discuss.

3.2 Fixed Step-Size

We'll discuss the performance of the two approaches with a fixed step-size $\eta = 0.01$

3.2.1 Largest-Gradient coordinate descent

The Largest-Gradient CD converges to the Loss $L*$ with a final loss value:

$$Loss(Largest - Gradient) = 0.00047$$

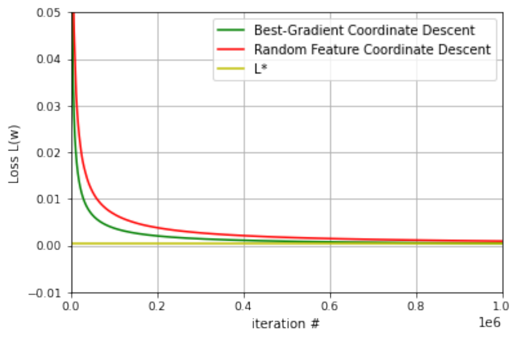

Figure 1: Iteration versus Loss for fixed step-size

3.2.2 Random-feature coordinate descent

The Random-feature CD converges to the Loss $L*$ with a final loss value:

$$Loss(Random) = 0.00094$$

Figure 1 clearly shows that both the largest-gradient and the random-feature coordinate descent methods converge to the optimal Loss $L*$. However, the rate at which both of these methods converge vary greatly. The largest-gradient CD method converges faster than the random-feature CD method and outperforms it to converge in lesser number of iterations.

Approach	**Loss**
Optimal L*	0.00039
Largest-Gradient CD	0.00047
Random-feature CD	0.00094

Table 1: Final Loss values for fixed step-size

Table 2 shows the optimal Loss $L*$ value and also the loss values for the largest-gradient and the random-feature coordinate descent methods for fixed step-size. It can be seen that the final loss value for Largest-Gradient CD is much less than that of Random-feature CD.

3.3 Adaptive Step-Size (Additional Discussion)

Additionally, we also used Backtracking line-search for adaptive step-size that changes with each iteration. However, the results were not as favorable. With this approach, the random-feature CD worked well and converged to the desired value but the same did not happen in the case of largest-gradient CD. In the latter case, the loss value was much higher than expected and the convergence took place after many iterations.

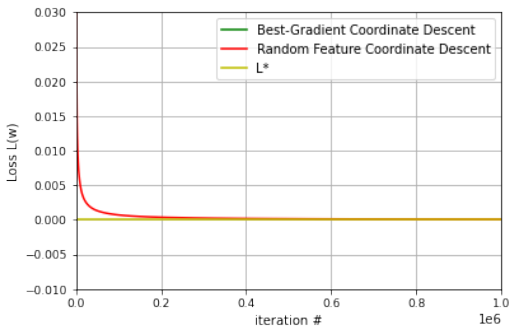

Figure 2: Iteration versus Loss for adaptive step-size

Figure 2 shows clear convergence of Random CD but due to scaling issue, the convergence of largest-gradient CD can not be seen as it took much higher number of iterations.

Approach	**Loss**
Optimal L*	0.000035
Largest-Gradient CD	1.51
Random-feature CD	0.000082

Table 2: Final Loss values for adaptive step-size

Table 2 shows the optimal Loss $L*$ value and also the loss values for the largest-gradient and the random-feature coordinate descent methods for adaptive step-size. It can be seen that the final loss value for Largest-Gradient CD is much higher than that of Random-feature CD.
Therefore, there is a scope of improvement in this method with adaptive step-size since this particularly did not perform as per the expectations.

4 Critical Evaluation

There is a lot of scope for improvement in the coordinate descent method discussed in the first section. Further, changes in the following areas can be made to improve the method and give better results:

1. *Loss function restrictions:* Currently, the loss function is required to be continuous and differentiable. The current method will not work for the cases when the loss function is not differentiable.

2. *Lack of information:* Only first-order derivative information is available right now. The information about the second-order derivative is not available which may help in further development of the coordinate descent method.

3. *Number of iterations:* The current coordinate descent method converges to the optimal $L*$ value after about a million iterations. The target is to have a faster convergence without compromising the efficiency.

4. *Time duration:* Due to higher number of iterations, it takes a longer duration to train the model. Additionally, the step-size has to be set at a higher value, which is not desirable, to increase the convergence rate.

5. *Adaptive step-size for greedy approach:* The backtracking line approach to adapt to suitable step-size is not working that well for the greedy approach of selecting the index with the largest gradient magnitude for the update scheme. The

loss is in fact higher than with a constant step-size. This approach is working well for random index selection though.

Some of the issues mentioned can be resolved by using stochastic gradient instead of just gradient. This will help with the problem of the loss function's need to be differentiable. Also, slow convergence can be resolved by using the block coordinate descent (BCD) which will help with faster convergence thereby decreasing the duration and the number of iterations.

5 Sparse Matrix

The implementation for the sparse matrix case has not been done due to time restrictions. However, generally the greedy approach used in this project to select the index is well-suited for sparse matrix problems. For sparse logistic regression problems, one-dimensional Newton direction coordinate update can be used [4].

6 References

1. Gordon, Geoff, and Ryan Tibshirani. "Gradient descent revisited." Optimization 10.725/36 (2012): 1-31.

2. Shi, Hao-Jun Michael, et al. "A primer on coordinate descent algorithms." arXiv preprint arXiv:1610.00040 (2016).

3. S. J. Wright. Coordinate descent algorithms. Mathematical Programming, 151(1):3–34, 2015.

4. T. T. Wu and K. Lange. Coordinate descent algorithms for lasso penalized regression. The Annals of Applied Statistics, 2(1):224–244, 2008.

5. Tibshirani, Ryan, and Micol Marchetti-Bowick. "Gradient descent: Convergence analysis, 2013." URL http://www. stat. cmu. edu/ryantibs/convexopt-F13/scribes/lec6. pdf.