CSE250B Programming Project 2: Coordinate Descent

Kevin Tan

February 22, 2019

1. **High-level description for coordinate descent method**

   Coordinate descent (CD) methods solve optimization problems by iteratively performing approximate minimization along the selected coordinate axis or hyperplane. Rather than moving all coordinates along a descent direction, CD changes a chosen coordinate at each iterate, moving as if it were on a grid, with each axis corresponding to each component.

   In this assignment, we consider optimizing the loss function $L(\cdot)$ which depends on $w \in R^d$, in the context of logistic regression. We can approach this by initializing w somehow, then repeatedly picking a coordinate $i \in 1, 2, ..., d$ and updating the value of w_i so as to reduce the loss. Within this framework, there are many different approaches for choosing an index and updating the selected coordinate.

   For choosing the index, we will adopt the greedy method, which chooses an index such that the objection function is minimized most, i.e. the index at which the gradient is the maximum magnitude. We use the *Gauss-Southwell selection rule (GS)*:

   $$i_k = \underset{1 \leq j \leq d}{\operatorname{argmax}} |\nabla_j L(w^{k-1})| \tag{1}$$

   For the update scheme, we use the update equation similar to in gradient descent, knowing that $L(w)$ is convex and differentiable:

   $$w_i^k = w_i^{k-1} - \alpha_i \nabla_i L(\mathbf{w}^{k-1}) \tag{2}$$

   To compute the step-size α_i, we use backtracking line search [1]. Note that this approach requires the function $L(\cdot)$ to be differentiable. Starting at $t-1$ and with $0 < \beta < 1$, we update $t = \beta t$ so long as the following condition holds:

   $$L(w - \nabla L(w)) > L(w) - \frac{t}{2}||\nabla L(w)||^2 \tag{3}$$

   and set the step-size $\alpha_i = t$.

2. **Convergence**

   Intuitively, since the greedy choice of index will always choose the coordinate that minimizes the loss function the most, we know the loss will always decrease at each iteration. We can then rely on the step-size to converge to the optimal loss.

   From [1], we know that, assuming $f : R^n \to R$ is convex and differentiable, and ∇f is Lipschitz continuous with constant $L > 0$, backtracking line search is guaranteed to converge at rate $O(1/k)$.

3. **Experimental Results**

We use the UCI Wine Dataset for our experiments, using only the first two classes ($n = 130$). We normalize the data to have mean 0 and variance 1 using `sklearn.preprocessing.StandardScalar`, and prepend a bias term to the data, resulting in shape $(130, 14)$.

We first use `sklearn.linear_model.LogisticRegression` [3] as a standard logistic regression solver (without regularization, $C = 10^{10}$) for a baseline, obtaining a loss $L^* = 3.3329 * 10^{-5}$.

Then, we run our algorithm as well as a random-feature selection algorithm (choosing the index to descend upon at random) and show the results on the graph below. We use the stopping criteria `np.linalg.norm(w_new - w, ord=1) < 0.001` for our algorithm, and find that our algorithm terminates after 4000 iterations. To make a fair comparison against random-feature coordinate descent, we run both algorithms for the same number of iterations (`max_iter=4000`). We use the same backtracking line search strategy for choosing the learning rate at each iteration for both approaches. We initialize w as the zero vector of shape (14,1) with some noise sampled from a normal Gaussian with mean 0 and variance 0.01.

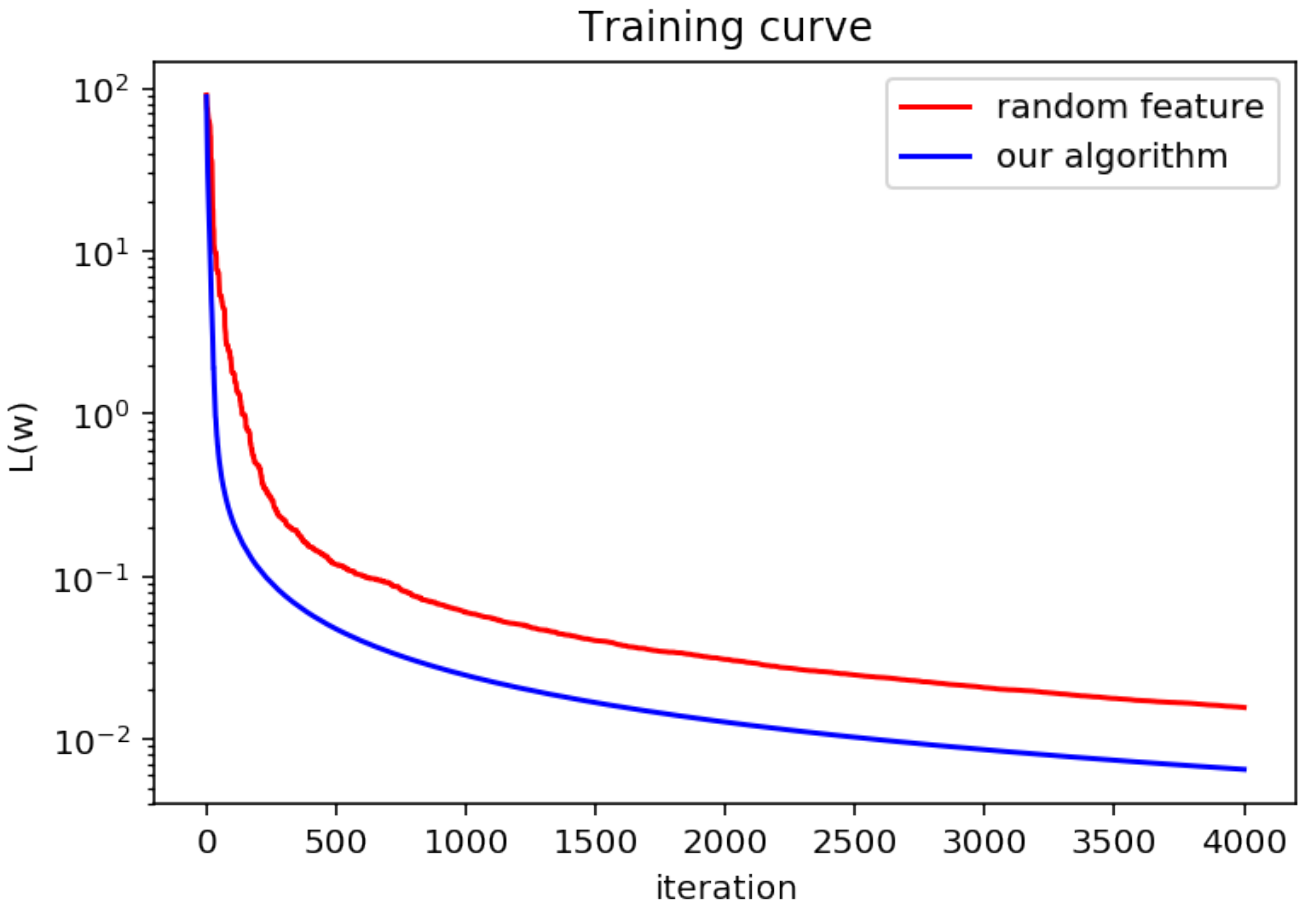

Figure 1: Training loss of our algorithm (blue) versus random-feature coordinate descent (red)

Experimental results show that, after 4000 iterations, the random-feature approach obtains a loss of $L_{random} = 0.01566$, and our approach obtains a loss of $L_{ours} = 0.00652$, both of which asymptote to $L^* = 3.3329 * 10^{-5}$.

4. **Critical Evaluation**

   Future works may empirically compare various approaches to choosing the index upon which to descend: cyclic, random, and greedy methods. We may try to run extensive experiments with the same UCI Wine dataset and choose the method that gives the best training loss. Other methods could be proposed to generalize to other non-smooth problems.

   Furthermore, there exists a wide literature [2] of update schemes besides backtracking line search, which take advantage of choosing a block of coordinates rather than a single coordinate at each iterate. These methods are more suitable for distributed or parallel computing for larger datasets.

5. **Sparse Coordinate Descent (Optional)**

   My proposal is to add a L1 regularizer to the logistic loss function.

References

[1] `https://www.cs.cmu.edu/~ggordon/10725-F12/scribes/10725_Lecture5.pdf`

[2] H.-J. M. Shi, S. Tu, Y. Xu, and W. Yin, A primer on coordinate descent algorithms, arXiv:1610.00040, (2016).

[3] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E., Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, vol. 12, 2011.