CSE 250B: Homework 3

Xiyun Liu A53099348

December 2, 2016

1 Description of your coordinate descent method

To answer the questions asked in the homework description:

1. *Which coordinate to choose:*
   (a) for $k \in 1, 2...14$, for $j \in 1, 2, 3$, update $w_j[k]$. In another word, update w column by column, and in each column, update one coordinate by one. Save the change of loss L into $lossChange$ when each column finishes updating.
   (b) Choose the column k which has the largest value in $lossChange$, then, for $j \in 1, 2, 3$, update $w_j[k]$. Calculate the current loss, and update $lossChange[k]$
   (c) Repeat step(b) until L converges (the change of L during one update is less than 1e-3) or the number of iteration meets max number ($3e5$) of iteration.
2. *How to set the new value of $w_j[k]$:* Initialize all w with 0. Using gradient descent to update the value.
   $$w_j[k]_{t+1} = w_j[k]_t - \eta \frac{\partial L(w)}{\partial w_j[k]}$$
   where η = 5e-4 is the step size.
3. *Do you need the function L(w) to be differentiable:* Yes, because I use gradient descent.

The loss function: In this WINE logistic regression problem, let $Y = \{1, 2, 3\}$ and X is a 128×14 data matrix whose 14^th column is all 1 for offset, we can specify a classifier w which is 3×14 matrix including $w_1, w_2, w_3 \in \mathbb{R}^{14}$:

$$Pr(y = j|x) = \frac{e^{w_{j^{(i)}} \cdot x^{(i)}}}{e^{w_1 \cdot x^{(i)}} + e^{w_2 \cdot x^{(i)}} + e^{w_3 \cdot x^{(i)}}} \tag{1}$$

We need to maximize the likelihood

$$\prod_{i=1}^{128} Pr(y^{(i)}|x^{(i)})$$

Taking negative log to get the loss function for this multiclass logistic regression problem

$$\begin{aligned}
L(w) &= -\sum_{i=1}^{128} \log Pr(y = y^{(i)}|x^{(i)}) \\
&= -\sum_{i=1}^{128} \log \frac{e^{w_{j^{(i)}} \cdot x^{(i)}}}{e^{w_1 \cdot x^{(i)}} + e^{w_2 \cdot x^{(i)}} + e^{w_3 \cdot x^{(i)}}} \\
&= -\sum_{i=1}^{128} w_{j^{(i)}} \cdot x^{(i)} + \sum_{i=1}^{128} \log \{e^{w_1 \cdot x^{(i)}} + e^{w_2 \cdot x^{(i)}} + e^{w_3 \cdot x^{(i)}}\}
\end{aligned}$$

$$\frac{\partial L(w)}{\partial w_{j,k}} = \sum_{i=1}^{128} \begin{cases} -x_k^{(i)}, & \text{if } y^{(i)} = j \\ 0, & \text{otherwise} \end{cases} + \sum_{i=1}^{128} \frac{x_k^{(i)} e^{w_j \cdot x^{(i)}}}{e^{w_1 \cdot x^{(i)}} + e^{w_2 \cdot x^{(i)}} + e^{w_3 \cdot x^{(i)}}}$$

Prediction :

$$Pr(y = j|x) = \underset{j}{argmax}\ w_j \cdot x$$

2 Convergence

My method converges to the optimal loss when the change of loss during one update for all w is less than 1e-3. If the program keeps running, I terminate the program when the number of iteration meets the maximum number of iteration(3e5).

To make sure the algorithm can converge, the loss function should be convex in order to achieve a global optimal. Also, step size should be reasonable, not too large.

3 Experimental results

The training data is normalized by dividing each feature by the maximum value among each feature in case of overflow before running experiments.

Begin by running a standard logistic regression solver. Here I use `linear_model.LogisticRegression` from `scikit − learn`, with parameters $C = 1e10, solver =' lgfbs', multi\_class =' multinomial'$. Since in this problem, we have added the offset into the training data, there should be no regularization. C is the inverse of regularization strength, therefore, C should be as large as could be.

The final loss $L*$ calculated by the loss function I defined in the last section is 0.286

Then compare my coordinate descent method to random-feature coordinate descent. From figure 1, we can tell that my algorithm has a larger decreasing rate than the random-feature coordinate descent and it converges to L* faster() than the random one. However, the final loss(7.81374) is larger than L*, and this is because of the different method to update coordinate. From figure 2, we can see that the error rate of my algorithm and the random-feature coordinate descent are end up with the similar value(0.08). Since the test data is really small, only 50 data points, I can't tell which method is better. I will discuss more in the next section.

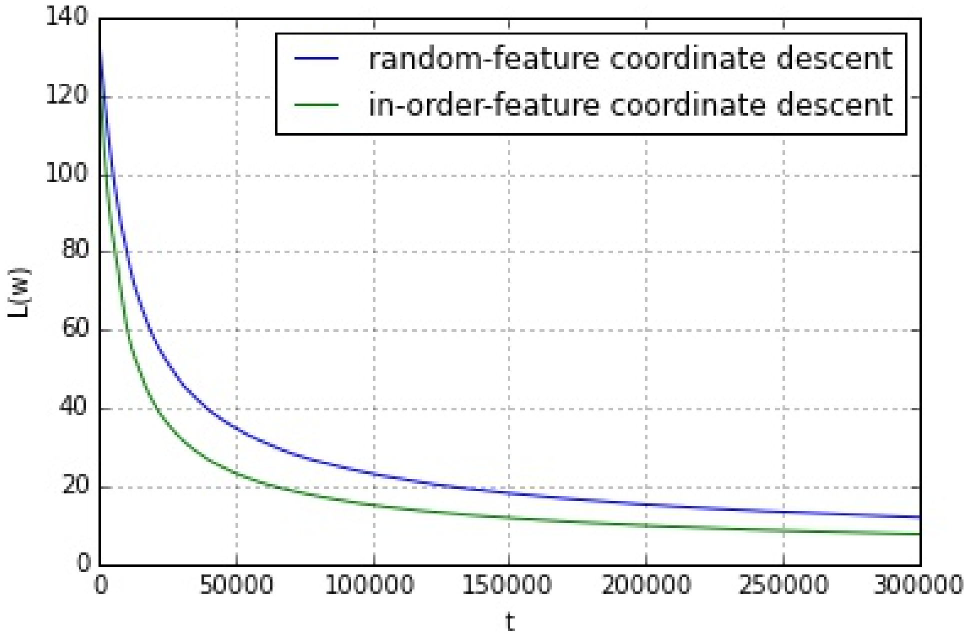

Figure 1: The comparison of the loss of my algorithms $L(w_t)$ decreases with t with the random-feature coordinate descent

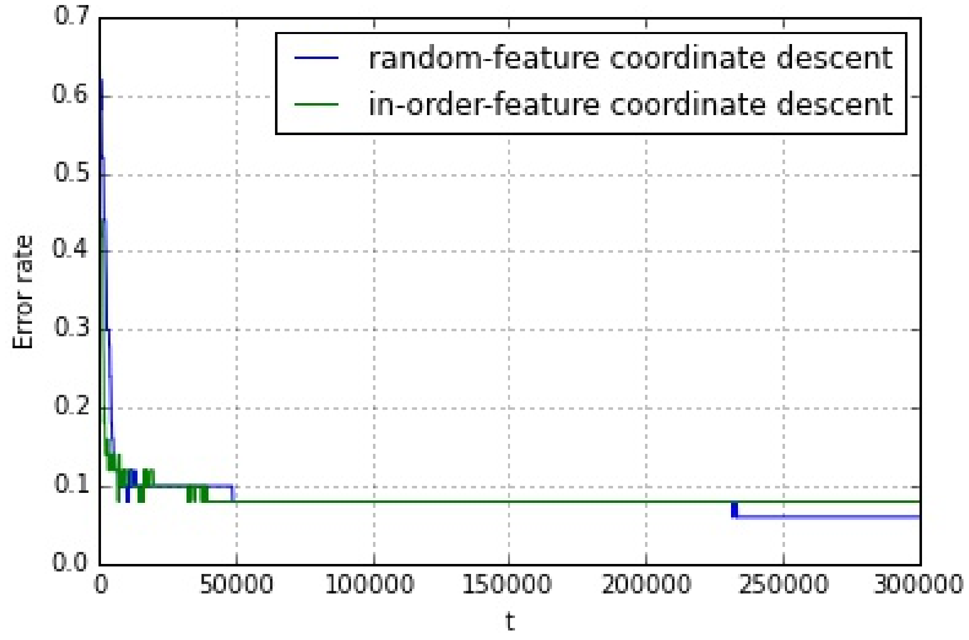

Figure 2: The comparison of error rate decreases with t

4 Critical evaluation

I think there is scope for further improvement in my coordinate descent scheme. First, improve the way to choose coordinate to update. Actually besides the method I write in this report, I also tried to choose coordinate column by column, and in each column, one by one. In other words, instead of choosing the coordinate which reduces the loss most, I just choose the coordinate one by one. It turns out that the performance of this naive coordinate descent is pretty much the same as random-feature coordinate descent. The graphs are attached in the next page. That's why I implemented the algorithm I showed in this report. However, I found its performance really depends on the training data and test data.
I learn that there are other efficient coordinate descent sweep patterns except for sequential and random. If I have time, I would like to try implement them and compare the performance.

Second, improve the way to update coordinate. One of the advantages of coordinate descent is that it is gradient free and easy implementation. However, in this homework, I still use gradient descent to update each coordinate, which may lead to a longer running time. Therefore, we can try other way to update coordinate values, such as line search.

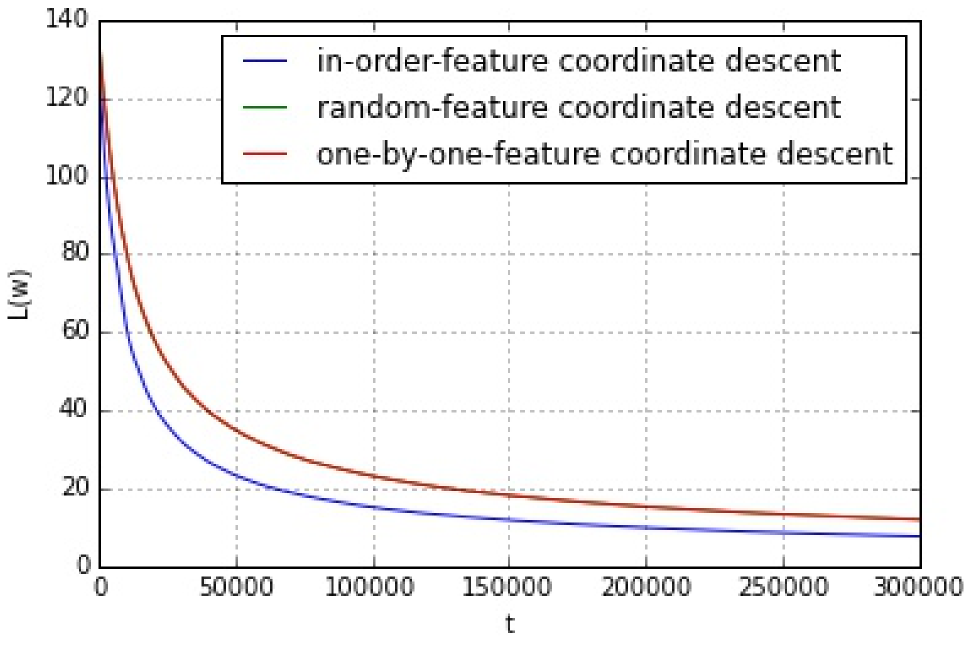

Figure 3: $L(w_t)$ decreases with t

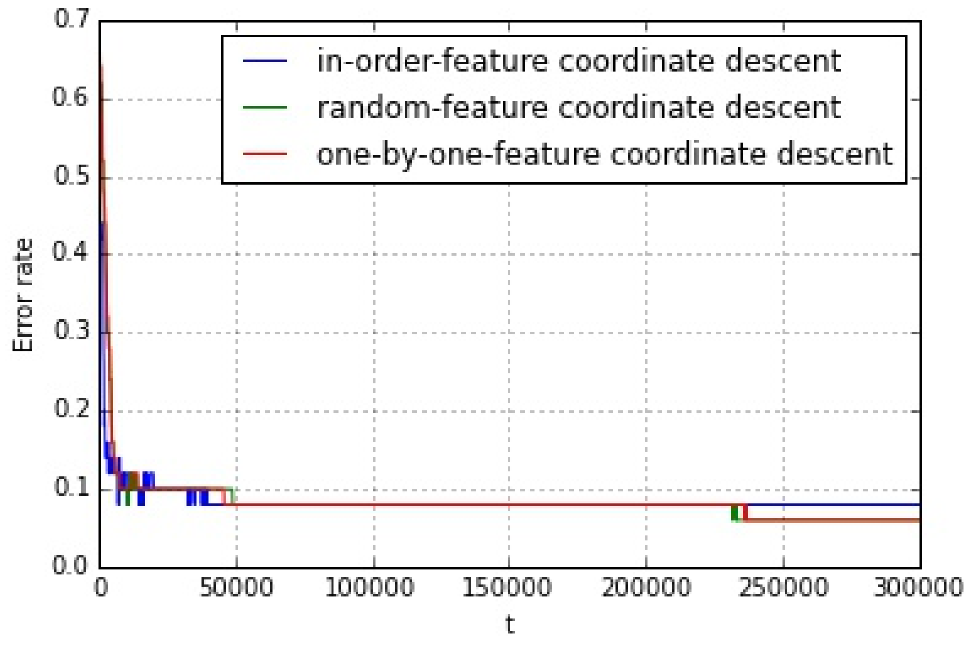

Figure 4: error rate decreases with t