CSE 547: Homework 3

Due on May 19, 2017

Brian de Silva

Collaborators

I collaborated with Emily Dinan on Problem 1.5.

1 (Dual) Coordinate Ascent

In this problem, we will derive the dual coordinate ascent algorithm for the special case of ridge regression. In particular, we seek to solve the problem:

$$\min_w L(w) \text{ where } L(w) = \sum_{i=1}^{n} (w \cdot x_i - y_i)^2 + \lambda \|w\|^2$$

Recall that the solution is:

$$w^* = (X^\top X + \lambda I)^{-1} X^\top Y$$

where X be the $n \times d$ matrix whose rows are x_i, and Y is an n-dim vector.
Recall the argument:

$$\begin{aligned} w^* &= (X^\top X + \lambda I)^{-1} X^\top Y \\ &= X^\top (X X^\top + \lambda I)^{-1} Y \\ &:= \frac{1}{\lambda} X^\top \alpha^* \end{aligned}$$

where $\alpha^* = (I + XX^\top/\lambda)^{-1} Y$.

1. (Linear algebra brush up) Prove that the above manipulations are true (without any assumptions on X or n or d). This involves proving the second equality in the expression.

As in class, define:

$$G(\alpha_1, \alpha_2, \ldots \alpha_n) = \frac{1}{2} \alpha^\top (I + XX^\top/\lambda)\alpha - Y^\top \alpha$$

Now let us consider the consider the coordinate ascent algorithm:

- start with $\alpha = 0$.
- choose coordinate i randomly, and update:

$$\alpha_i = \arg\min_z G(\alpha_1, \ldots \alpha_{i-1}, z, \ldots, \alpha_n)$$

- repeat the previous step until some termination criterion is reached.
- return $w = \frac{1}{\lambda} X^\top \alpha$.

2. Show that the solution to the inner optimization problem for α_i is:

$$\alpha_i = \frac{(y_i - \frac{1}{\lambda}(\sum_{j \neq i} \alpha_j x_j) \cdot x_i)}{1 + \|x_i\|^2/\lambda}$$

3. What is the computational complexity of this update, as it is stated? (Assume that scalar multiplication and scalar addition is order one.)

4. What is the computational complexity of one stochastic gradient descent update?

Now let us consider the update rule from class:

- start with $\alpha = 0$, $w = \frac{1}{\lambda} X^\top \alpha = 0$.
- Choose coordinate i randomly and perform the following update:
  - Compute the differences:
  $$\Delta \alpha_i = \frac{(y_i - w \cdot x_i) - \alpha_i}{1 + \|x_i\|^2/\lambda} \tag{1}$$
  - Update the parameters as follows:
  $$\alpha_i \leftarrow \alpha_i + \Delta\alpha_i, \quad w \leftarrow w + \frac{1}{\lambda} x_i \cdot \Delta\alpha_i \tag{2}$$
- Repeat the previous step until some termination criterion is reached.
- return $w = \frac{1}{\lambda} X^\top \alpha$.

Now let us examine why this algorithm is more efficient.

5. Prove that the above update is valid.
6. What is the computational complexity of the update, Equation *2), in the above implementation?

Solution:

1. Note that both XX^T and X^TX are symmetric positive-semidefinite, so all of their eigenvalues λ_i satisfy $\lambda_i \geq 0$ (they share the same eigenvalues, with the larger of the two matrices having 0 as a repeated eigenvalue with higher algebraic multiplicity). The eigenvalues of $X^TX + \lambda$ and $XX^T + \lambda I$ are therefore $\lambda_i + \lambda$. If $\lambda > 0$ then these eigenvalues are strictly positive, in which case it follows that both matrices are invertible. Hence
$$\begin{array}{rrcl} & X^TY & = & X^TY \\ \iff & X^TY & = & X^TIY \\ \iff & X^TY & = & X^T(XX^T + \lambda I)(XX^T + \lambda I)^{-1}Y \\ \iff & X^TY & = & (X^TX + \lambda I)X^T(XX^T + \lambda I)^{-1}Y \\ \iff & (X^TX + \lambda I)^{-1}X^TY & = & X^T(XX^T + \lambda I)^{-1}Y. \end{array}$$
This shows that the second equality in the expression is true.

2.

2 Let's try it out!

Dimension Reduction with PCA

Project each image onto the top 50 PCA directions. This reduces the dimension of each image from 784 to 50. The reason for doing this is to speed up the computation.

Feature generation: Random Fourier Features to approximate the RBF Kernel

Now we will map each x to a k-dimensional feature vector as follows:
$$x \to (h_1(x), h_2(x), \dots h_k(x))$$
We will use $k = N = 30,000$. Each h_j will be of the form:
$$h_j(x) = sin\left(\frac{v_j \cdot x}{\sigma}\right)$$
To construct all these vectors $v_1, v_2, \dots v_k$, you will independently sample every coordinate for every vector from a standard normal distribution (with unit variance). Here, you can think of σ as a bandwidth parameter.

2.1 SGD and Averaging

Let us optimize the square loss. We do this as a surrogate to obtain good classification accuracy (on the test set). In practice, instead of complete randomization, we often run in *epochs*, where we randomly permute our dataset and then pass through out dataset in this permuted order. Each time we start another epoch, we permute our dataset again.
Let us plot the losses of two quantities. Let us keep around your parameter vector w_t (your weight vector at iteration t). Let us also track the average weight vector $\overline{w}_\tau$ over the last epoch, i.e. $\overline{w}_\tau$ is the average parameter vector over the τ-th epoch. The average parameter is what is suggested to be used by the Polyak-Juditsky averaging method.
Define the total square loss as the *sum* square loss over the 10 classes (so you do not divide by 10 here).
Throughout this question you will use a mini-batch size of 1.

1. Specify your parameter choices: your learning rate (or learning rate scheme) and the kernel bandwidth.
2. You should have one plot showing the both the squared loss after every epoch (starting with your initial squared error). Please label your axes in a readable way. Plot the loss of both w_t and the average weight vector $\overline{w}_\tau$. You should have both the training and test losses on the same plot (so there should be four curves).
3. For the classification loss, do the same, except start your plots at the end of the first epoch. Again, there should be four curves (for the training and test loss of both parameters).
4. What is your final training squared loss and classification loss for both parameters? What is the total number of mistakes that are made on the training set of your final point? Comment on the performance of the average iterate vs. the last point.
5. What is your final test squared loss and classification loss for both parameters? What is the total number of mistakes that are made on the test set of your final point? Comment on the performance of the average iterate vs. the last point.

Solution:

1.

2.2 SDCA

Now let us try out our (dual) coordinate ascent method. Again, instead of complete randomization, we often run in *epochs*, where we randomly permute our dataset and then pass through out dataset in this permuted order. Each time we start another epoch, we permute our dataset again.
Choose a regularization parameter. Do this appropriately so you obtain a good test accuracy.

1. Specify your regularization parameter. Are there any other parameter choices?
2. Now plot the dual loss G. Your plot should show the G-loss after every epoch (starting with your initial squared error). Please label your axes in a readable way.
3. Does it make sense to plot the G-loss on the test set? Why not?
4. Now plot the squared loss after every epoch (starting with your initial squared error). Please label your axes in a readable way. You should have both the training and test losses on the same plot (so there should be two curves).
5. For the classification loss, do the same, except start your plots at the end of the first epoch. Again, there should be two curves (for the training and test loss).

6. What is your best training squared loss and classification loss? What is the total number of mistakes that are made on the training set of your final point?

7. What is your best test squared loss and classification loss? What is the total number of mistakes that are made on the test set of your final point?

8. Compare the speed and accuracy to what you obtained with SGD?

Solution:

1.

2.3 Mini-batching (in the dual) and parallelization!

Now, let us understand parallelization issues with mini-batching.
Here instead of updating one coordinate at a time, we will update multiple coordinates. In particular, we will choose a batch of b coordinates and compute the Δ_i's of these coordinates as specified in Equation 1.
Let us choose a batch size $b = 100$.

1. One possibility is to just update the α_i's as specified in Equation 2. Is this ok?

2. Argue that the following update never decrease the value of G:

$$\alpha_i \leftarrow \alpha_i + \frac{1}{b}\Delta\alpha_i, \quad w \leftarrow w + \frac{1}{b}\frac{1}{\lambda}x_i \cdot \Delta\alpha_i$$

   where the update is performed on all of the chosen coordinates in the mini-batch.

3. Now use this algorithm with $b = 100$. And provide the same training and test loss plots, for the square loss and classification loss as before. (Again, use a permuted order rather than complete randomization). Plot this with your earlier curves for $b = 1$ and have the x axis be the total number of points used. Note that b steps on your x-axis will correspond to one update.

4. In terms of the *total number of data points touched*, does this perform notably better or worse than what you had before, with the $b = 1$?

Now let us improve upon this algorithm. Consider the update rule with parameter γ:

$$\alpha_i \leftarrow \alpha_i + \gamma\Delta\alpha_i, \quad w \leftarrow w + \frac{\gamma}{\lambda}x_i \cdot \Delta\alpha_i$$

where the update is performed on all of the chosen coordinates in the mini-batch.

1. Give an argument as to why choosing $\gamma = 1/b$ is pessimistic. (Either give a concise argument or give an example.)

2. Now empirically (on our $b = 100$ experiments) search a little to find a good choice for gamma. What value of γ did you choose? What value of γ do things diverge?

3. Now use this choice to redo your experiments. Provide the same training and test loss plots, for the square loss and classification loss as before. (Again, use a permuted order rather than complete randomization).

4. In terms of the *total number of data points touched*, does this perform notably better than what you had before, for both SGD and for the $\gamma = 1/b$ case? What have noticed about your total runtime?