CAP6610 Machine Learning, Spring 2022

Midterm 2 Solution

1. (10%) *Train vs test datasets.* Suppose you are building a classifier that identifies cats and dogs. You have a dataset of 3,000 images containing cats, dogs, or other objects (neither cat nor dog). You randomly split the data into a 2,500 image training set and a 500 image test set.

   (a) Why is it important to "reserve" some images for the test dataset? (Why shouldn't we use all 3,000 images to train the classifier?)

   (b) After training your classifier for a while, you observe it performs well on the training images, but poorly on the test images. What is one possible explanation?

   Solution.

   (a) Because no matter how well the model fits the training data, there is no guarantee that the prediction will remain accurate on a new data sample, which was the whole purpose of designing this classifier. Splitting the data set into training/testing is a way to estimate how well it would work on unseen data.

   (b) Overfitting.

2. (20%) *Elementary properties of the quadratic regularized logistic classification.* Consider

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i\boldsymbol{\phi}_i^\top\boldsymbol{\theta})) + \lambda\|\boldsymbol{\theta}\|^2, \tag{1}$$

   for $y_i = \pm 1$. Answer the following true/false questions:

   (a) Problem (1) has multiple locally optimal solutions.

   (b) Let $\boldsymbol{\theta}^\star$ be an optimal solution for (1), $\boldsymbol{\theta}^\star$ is sparse (has many zero entries).

   (c) If the training data is linearly separable, then some coefficients θ_j might become infinite if $\lambda = 0$.

   (d) At optimum, the empirical risk always increases as we increase λ.

   (e) On a test set, the prediction accuracy always increases as we increase λ.

   Solution.

   (a) False. The problem is strongly convex, so there is a unique global optimum.

   (b) False. The quadratic regularization $\|\cdot\|^2$ does not encourage the solution to be sparse.

   (c) True. If the training data is linearly separable, there exists $\boldsymbol{\theta}$ such that $y_i\boldsymbol{\phi}_i^\top\boldsymbol{\theta} > 0$ for all $i = 1,\ldots,n$. For such a $\boldsymbol{\theta}$, multiplying it with a positive scalar can only reduce the empirical logistic loss further. Therefore, if $\lambda = 0$, some coefficients θ_j might become infinite.

   (d) True. We proved this in Homework 3.

   (e) False. In general you cannot say anything about the test set performance.

3. (30%) *Learning algorithms for L_1 regularized Huber regression.* Consider the L_1-norm regularized Huber regression problem

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i) + \lambda \|\boldsymbol{\theta}\|_1, \tag{2}$$

where the loss function is the Huber loss

$$\ell(\varepsilon) = \begin{cases} \varepsilon^2/2 & |\varepsilon| \le a \\ a(|\varepsilon| - a) & |\varepsilon| > a \end{cases}$$

with some constant a. We assume the empirical risk part $(1/n)\sum_{i=1}^n \ell(\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i)$ is *strongly convex*.

(a) Is the Huber loss differentiable? If yes, write down the gradient of the loss of one sample $\ell(\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i)$; if not, find a subgradient of it at any $\boldsymbol{\theta}$;

(b) Give the pseudo-code of the proximal (sub)gradient method with constant step size γ for solving (2). Your answer should not contain any abstract operation such as "the proximal operator of the L_1 norm". What is the convergence rate of this algorithm?

(c) Give the pseudo-code of the stochastic proximal (sub)gradient method with constant step size γ for solving (2). Again, your answer should not contain any abstract operation such as "the proximal operator of the L_1 norm". What is the expected convergence rate of this algorithm?

Hint. The convergence rate of either algorithm is one of the following: superlinear, linear, $(1/t)$-sublinear, or $(1/\sqrt{t})$-sublinear.

Solution. *We discussed the gradient of the Huber loss on March 1 (although not entirely correct during class).*

(a) Yes, the Huber loss is differentiable: at $\varepsilon = \pm a$, the (directional) derivatives from the left and right are both $\pm a$, so this point is smooth. Applying the chain rule, we have

$$\begin{aligned} \nabla \ell(\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i) &= \begin{cases} (\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i)\boldsymbol{\phi}_i & |\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i| < a \\ a\boldsymbol{\phi}_i & \boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i \ge a \\ -a\boldsymbol{\phi}_i & \boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i \le -a \end{cases} \\ &= \min(1, a/|\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i|)(\boldsymbol{\phi}_i^\top \boldsymbol{\theta} - y_i)\boldsymbol{\phi}_i. \end{aligned}$$

(b) The proximal operator for the L_1 norm regularization is soft-thresholding. Together with the gradient expression from part (a), the pseudo-code of proximal gradient descent is

```
repeat
    for i = 1, ..., n do
        θ ← θ − (γ/n) min(1, a/|φ_i^T θ − y_i|)(φ_i^T θ − y_i) φ_i
    end for
    for j = 1, ..., m do
        θ_j = (1 − γλ/|θ_j|)_+ θ_j
    end for
until convergence
```

This is proximal gradient descent applied to a strongly convex problem, so the convergence rate is linear [`lec6.pdf`, page 38].

(c) The main difference is that the search direction is determined by only one sample, not the entire data set:

```
repeat
    sample i from {1, ..., n} uniformly
    θ ← θ − γ min(1, a/|φ_i^⊤ θ − y_i|)(φ_i^⊤ θ − y_i)φ_i
    for j = 1, ..., m do
        θ_j = (1 − γλ/|θ_j|)_+ θ_j
    end for
until convergence
```

This is stochastic proximal gradient descent applied to a strongly convex problem, so the convergence rate is $1/t$-sublinear [`lec6.pdf`, page 26].

4. (20%) *An important inequality to prove convergence of the proximal gradient descent algorithm.* Consider a regularized empirical risk minimization problem

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad L(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta}).$$

We try to solve this optimization problem using proximal gradient descent (PGD): assuming $L(\boldsymbol{\theta})$ is differentiable, PGD iteratively updates the variable as

$$\boldsymbol{\theta}^{(t+1)} = \text{Prox}_{\gamma^{(t)}\lambda r}(\boldsymbol{\theta}^{(t)} - \gamma^{(t)}\nabla L(\boldsymbol{\theta}^{(t)})).$$

To show that it converges to global minimum, a key inequality given in `lec6.pdf` page 38 is

$$L(\boldsymbol{\theta}^{(t+1)}) + \lambda r(\boldsymbol{\theta}^{(t+1)}) \le L(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta}) + \frac{1}{\gamma^{(t)}}(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t+1)})^\top(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}) + \frac{M}{2}\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t+1)}\|^2$$

for any $\boldsymbol{\theta}$. In this question you should prove this inequality using the following assumptions:

- Both L and r are convex. Since L is assumed to be differentiable, we have

$$L(\boldsymbol{\theta}) \ge L(\boldsymbol{\theta}^{(t)}) + \nabla L(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) \tag{3}$$

  for all $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{(t)}$; r is not necessarily differentiable, but there exists a similar inequality using a subgradient.

- The gradients of L are Lipschitz continuous with parameter M, which means

$$L(\boldsymbol{\theta}) \le L(\boldsymbol{\theta}^{(t)}) + \nabla L(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{M}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}\|^2 \tag{4}$$

  for all $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{(t)}$.

Solution. *We went over this proof at the beginning of the lecture on March 22.*

By definition of the proximal operator, $\boldsymbol{\theta}^{(t+1)}$ is the solution of the problem

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \gamma^{(t)}\lambda r(\boldsymbol{\theta}) + \frac{1}{2}\|\boldsymbol{\theta} - (\boldsymbol{\theta}^{(t)} - \gamma^{(t)}\nabla L(\boldsymbol{\theta}^{(t)}))\|^2.$$

Therefore 0 is a subgradient of this objective function at $\boldsymbol{\theta}^{(t+1)}$, which means

$$\frac{1}{\gamma^{(t)}}(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t+1)}) - \nabla L(\boldsymbol{\theta}^{(t)}) \in \partial\lambda r(\boldsymbol{\theta}^{(t+1)}).$$

Since λr is a convex function, and we have a subgradient of it at $\boldsymbol{\theta}^{(t+1)}$, this means

$$\lambda r(\boldsymbol{\theta}) \ge \lambda r(\boldsymbol{\theta}^{(t+1)}) + \left(\frac{1}{\gamma^{(t)}}(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t+1)}) - \nabla L(\boldsymbol{\theta}^{(t)})\right)^\top(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t+1)}). \tag{5}$$

Adding (5) and (3) and rearrange, we get

$$L(\boldsymbol{\theta}^{(t)}) + \lambda r(\boldsymbol{\theta}^{(t+1)}) \leq L(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta}) + \frac{1}{\gamma^{(t)}}(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t+1)})^\top(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}) - \nabla L(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}). \tag{6}$$

The next step is to bound $L(\boldsymbol{\theta}^{(t)})$ with $L(\boldsymbol{\theta}^{(t+1)})$ so that both terms on the left-hand-side are evaluated at the same point. Since we assume L is M-smooth and thus (4) holds for any $\boldsymbol{\theta}$, we let $\boldsymbol{\theta} = \boldsymbol{\theta}^{(t+1)}$ and rearrange the inequality into

$$L(\boldsymbol{\theta}^{(t+1)}) - \nabla L(\boldsymbol{\theta}^{(t)})^\top(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}) - \frac{M}{2}\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\|^2 \leq L(\boldsymbol{\theta}^{(t)}). \tag{7}$$

Combining (6) and (7) shows the key inequality.

5. (10%) *Clustering with pre-assigned vectors.* Suppose that some of the vectors in $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_n$ are assigned to specific groups. For example, we might insist that $\boldsymbol{\phi}_{27}$ be assigned to cluster 5. Suggest a simple modification of the k-means algorithm that respects this requirement. Describe a practical example where this might arise, when each vector represents m features of a medical patient.

   Solution. In each iteration of the k-means algorithm, each vector is assigned to the centroid that is closest to it and then the centroid is updated as the mean of each cluster. If some of the vectors are preassigned to specific clusters, then we can modify the k-means algorithm to fix the assignments of those pre-assigned vectors (no matter how far-away they are from the centroids).

   This might arise when the vectors represent patient feature vectors, and the groups (at least the ones we pre-assign) represent diagnoses of specific diseases. We have a collection of n patient vectors; for some of them, we have a known definitive diagnosis of the disease they have.

6. (10%) *Implementation of PCA.* In MATLAB you are given two functions to compute the singular value decomposition of a matrix, `svd` and `svds`. Explain which one is the preferred function to use in the following scenarios:

   (a) The data matrix $\boldsymbol{\Phi}$ is 100×300 and dense, and we want to embed each column as a 10-dimensional vector;

   (b) The data matrix $\boldsymbol{\Phi}$ is $10^5 \times 10^6$ and sparse (with approximately 10^7 nonzeros), and we want to embed each column as a 100-dimension vector.

   In each case, you are given the data matrix `Phi`. You should describe in detail how to use either functions, possibly with a few additional steps, to obtain the embedding matrix $\boldsymbol{Y}$ and the projection matrix $\boldsymbol{\Theta}$ that satisfies $\boldsymbol{\Theta}^\top\boldsymbol{\Theta} = \boldsymbol{I}$.

   Solution. *We had a relevant discussion during the class on March 29.*

   (a) For a matrix of this size, the `svd` function is the better choice. However, `svd` produces the full SVD, and we have to manually select the 10 principal components:

   ```
   k = 10;
   [U,S,V] = svd(Phi);
   Theta = U(:,1:k);
   Y = S(1:k,1:k)*V(:,1:k)';
   ```

   (b) For a large and sparse matrix, the `svds` function is the better choice, which directly computes the k principal components:

   ```
   k = 100;
   [U,S,V] = svds(Phi,k);
   Theta = U;
   Y = S*V';
   ```