Assignment 1 Solutions

Rishabh Iyer

March 12, 2020

1 Assignment Policies for CS 6301

The following are the policies regarding this assignment.

1. This assignment needs be done individually by everyone.
2. You are expected to work on the assignments on your own. If I find the assignments of a group of (two or more) students very similar, the group will get zero points towards this assignment.
3. Please use Python for writing code. You can submit the code as a Jupyter notebook (for example, Question 3 and 4 in this assignment)
4. For the theory questions, please use Latex (For example questions 1 and 2).
5. This assignment is for 18 points + 2 Bonus Points.
6. This Assignment is due on February 5th (Wednesday)

2 Questions

1. (6 points total) In this assignment, we will compute the Gradient and Hessian for a few supervised learning loss functions. Given training examples $\{(x_1, y_1), \cdots, (x_n, y_n)\}$ where $x_i \in \mathbf{R}^m$ is the feature vectors and y_i is the label

   - (1 Point) Compute the Gradient of the Hinge/SVM Loss: $L(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\}$. Here $y_i \in \{-1, +1\}$.

     Solution: This Function is not differentiable. The subgradient $h_L(w) = \sum_{i=1}^n -I(y_i w^T x_i < 1) y_i x_i$.
   - (2 Points) Compute the Gradient and Hessian of the Smooth SVM Loss: $L(w) = \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\}^2$. Here $y_i \in \{-1, +1\}$.

     Solution: Lets start with the gradient. This function is differentiable everywhere. The gradient $\nabla_w L = \sum_{i=1}^n -2I(y_i w^T x_i < 1) y_i x_i (1 - y_i w^T x_i)$.

     Lets move on to the Hessian. The Hessian is a Matrix so most students will compute the (j, k)th entry of the Matrix:

$$\frac{\partial^2 L}{\partial w_j w_k} = \sum_{i=1}^n I(y_i w^T x_i < 1) 2 x_i[j] x_i[k]$$

where $x_i[k]$ stands for the kth entry of the vector x_i. In Matrix notation, the Hessian is:

$$\nabla_w^2 L = \sum_{i=1}^n 2I(y_i w^T x_i < 1) x_i x_i^T$$

.

- (2 Points) Compute the Gradient and Hessian of Least Squares Loss: Least Squares Loss: $L(w) = \sum_{i=1}^n (y_i - w^T x_i)^2$. Here $y_i \in R$.

  Solution: Lets start with the gradient. This function is differentiable everywhere. The gradient $\nabla_w L = \sum_{i=1}^n 2(w^T x_i - y_i)x_i$.

  Lets move on to the Hessian. The Hessian is a Matrix so most students will compute the (j, k)th entry of the Matrix:

  $$\frac{\partial^2 L}{\partial w_j w_k} = \sum_{i=1}^n 2x_i[j]x_i[k]$$

  where $x_i[k]$ stands for the kth entry of the vector x_i. In Matrix notation, the Hessian is:

  $$\nabla_w^2 L = \sum_{i=1}^n 2x_i x_i^T$$

  .

- (1 Point) Compute the Gradient of a simple 2 Layer Function: $L(w) = \sum_{i=1}^n (y_i - \max(0, w^T x_i + b))^2$. Here $y_i \in R$.

  Solution: Since this is a function of w and b, the correct answer is to compute the gradients with respect to both parameters. However, in the assignment question, I put this as $L(w)$ so it is not clear that this is a function of b as well. As a result, we can still give them the point if they do not compute the gradient w.r.t b.

  $$\nabla_w L = \sum_{i=1}^n -2I(w^T x_i + b > 0) * (y_i - \max(0, w^T x_i + b))^2 x_i$$

  $$\nabla_b L = \sum_{i=1}^n -2I(w^T x_i + b > 0) * (y_i - \max(0, w^T x_i + b))^2$$

2. (2 Points + 2 Bonus Points) Next consider the following contextual bandit Problem. The task is to show k ads to users with each ad comprising of features (title, ad text, query etc.), and given an online policy which (with certain randomization) picks ads to show to users and the system logs feedback (whether the user clicks on the ad or not). We are given bandit logged data in the form of $\{(x_1, a_1, r_1, p_1), \cdots, (x_n, a_n, r_n, p_n)\}$. Assume we can take fixed number of actions $1:k$. Denote $x_i = \{x_i^1, \cdots, x_i^k\}$ as the feature vectors of the k actions for the ith instance. Assume each $x_i^j \in R^m$ is a m-dimensional feature vector. Denote a_i as the chosen action by the current online policy and denote p_i as the probability of the logged action (by the current online policy). Denote r_i as the Reward obtained by choosing action a_i, and finally, we define the policy as $\pi_\theta(x) = \text{argmax}_{i=1:k} \theta^T x^i$. Note that both θ and x^i as m-dimensional vectors. The Soft-Max Relaxation of the Inverse Propensity Objective is:

   $$\text{SM}(\theta) = \sum_{i=1}^n \frac{r_i}{p_i} \frac{\exp(\theta^T x_i^{a_i})}{\sum_{j=1}^k \exp(\theta^T x_i^j)} \tag{1}$$

   Compute the Gradient of this Loss function. In addition, **2 bonus** points if you can compute the Hessian of this Loss.

Solution: The gradient is:

$$\nabla_\theta SM = \sum_{i=1}^{n} \frac{r_i}{p_i}[x_i^{a_i} \frac{\exp(\theta^T x_i^{a_i})}{\sum_{j=1}^{k} \exp(\theta^T x_i^j)} - \frac{\exp(\theta^T x_i^{a_i})}{[\sum_{j=1}^{k} \exp(\theta^T x_i^j)]^2} \sum_{j=1}^{k} x_i^j \exp(\theta^T x_i^j)] \tag{2}$$

Lets now compute the Hessian. Denote $D(\theta) = \sum_{j=1}^{k} \exp(\theta^T x_i^j)$ as the denominator term. The Hessian is:

$$\begin{aligned}\nabla_\theta^2 SM = \sum_{i=1}^{n} \frac{r_i}{p_i}[x_i^{a_i} x_i^{a_i T} \frac{\exp(\theta^T x_i^{a_i})}{D(\theta)} - \frac{\exp(\theta^T x_i^{a_i})}{D(\theta)^2} \sum_{j=1}^{k} x_i^j x_i^{a_i T} \exp(\theta^T x_i^j)\\ - \frac{\exp(\theta^T x_i^{a_i})}{D(\theta)^2} \sum_{j=1}^{k} x_i^j x_i^{a_i T} \exp(\theta^T x_i^j) - \frac{\exp(\theta^T x_i^{a_i})}{D(\theta)^2} \sum_{j=1}^{k} x_i^j x_i^{jT} \exp(\theta^T x_i^j)\\ + \frac{2\exp(\theta^T x_i^{a_i})}{D(\theta)^3} (\sum_{j=1}^{k} x_i^j \exp(\theta^T x_i^j))(\sum_{j=1}^{k} x_i^j \exp(\theta^T x_i^{jT}))]\end{aligned}$$

While computing the above is not technically difficult, it is tedious because it involves multiple applications of the product and division rule of calculus. Many students may not have been able to get the exact expression, so we have given points based on the attempt.

3. (2 Points) Implement Numerically correct versions of the following functions:

- $L(x) = \log(1 + \exp(-x))$
- $L(x) = \log(\exp(x1) + \exp(x2))$
- $L(x) = \frac{\exp(x1)}{\exp(x1)+\exp(x2)}$

Solution: I will provide the solution here as an equation with cases. Many of the students would have provided python code for the same thing.

- **Part 1:**

$$L(x) = \begin{cases} \log(1 + \exp(-x)) & \text{if } x \geq 0 \\ \log(1 + \exp(x)) - x & \text{else} \end{cases} \tag{3}$$

- **Part 2:** There are two ways to do this. The first way is similar to above:

$$L(x) = \begin{cases} x_1 + \log(1 + \exp(x_2 - x_1)) & \text{if } x_1 \geq x_2 \\ x_2 + \log(1 + \exp(x_1 - x_2)) & \text{else} \end{cases} \tag{4}$$

An even better solution is to define $x_{max} = \max(x_1, x_2)$. And then define

$$L(x) = \log(\exp(x_1 - x_{\max}) + \exp(x_2 - x_{\max}))$$

Note that some of the students might use the function of Part 1 to do this. Though not really efficient, it might still be correct. Please verify the cases and make sure they are correct and as intended.

- **Part 3:** Again, there are two ways of solving this. The first is:

$$L(x) = \begin{cases} \frac{1}{1+\exp(x_2-x_1)} & \text{if } x_1 > x_2 \\ \frac{\exp(x_1-x_2)}{1+\exp(x_1-x_2)} & \text{else} \end{cases} \tag{5}$$

The second approach is like above, define $x_{max} = \max(x_1, x_2)$. And then,

$$L(x) = \frac{\exp(x_1 - x_{\max})}{\exp(x_1 - x_{\max}) + \exp(x_2 - x_{\max})}$$

4. (8 Points) Implement the Following Loss Functions in Python. Implement these functions as efficiently as you can (i.e. using vectorized code). Verify the implementations with a simple For Loop based implement. Finally, verify the computation of the Gradients with a numerical implement of the Gradients. Each of the functions below are 2 Points each.

- Logistic Loss: $L(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)) + \lambda ||w||_2^2$
- Hinge Loss/SVMs: $L(w) = \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \lambda ||w||_2^2$. Here $y_i \in \{-1, +1\}$
- Simple 2 Layer Function: $L(w) = \sum_{i=1}^{n} (y_i - \max(0, w^T x_i))^2 + \lambda ||w||_2^2$. Here $y_i \in R$.
- Least Squares Loss: $L(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda ||w||_2^2$. Here $y_i \in R$.

For each of the Loss Functions above, do the following steps:

- Instantiate a random dataset X and y with n (the number of instance) being 100 and m (the number of features) as 10. Initialize y to be a random n-dimensional vector. In the first three loss functions (classification), $y_i \in \{-1, +1\}$. For the last one (regression), y_i is a continuous number between 0 and 1. In both cases, make sure the labels are random.
- Create a random m-dimensional vector w
- For each Loss function implement the function which returns $[f, g]$ where f is the function value and g is the gradient. For example:

```
def LogisticLossFun(w, X, y, lam):
    ...
    return [f, g]
```

- First implement a simple for loop based code. Make sure the code runs.
- Next make sure the code is numerically stable by running it for large m (for example, $m = 100000$).
- Next, study how scalable the code is by running it with $n = 100000$.
- Implement a vectorized version of the code and verify both match for smaller dataset ($n = 100, m = 10$).
- Make sure the gradient is computed correctly. For this, numerically compute the gradient given the function value. The following code will numerically compute the gradient. First define:

```
funObj = lambda w: LogisticLossFun(w,X,y,1)[0]
```

Then define the gradient function as:

```
def numericalGrad(funObj, w,epsilon):
     m = len(w)
     grad = np.zeros(m)
     for i in range(m):
          wp = np.copy(w)
          wn = np.copy(w)
          wp[i] = w[i] + epsilon
          wn[i] = w[i] - epsilon
          grad[i] = (funObj(wp) - funObj(wn))/(2*epsilon)
return grad
```

With a sufficiently small value of *epsilon*, this function should be a good approximation of the gradient. Verify that both gradients (the numerical computation and the one you implement in the functions above) as the same.

Solution: For the solution of this, I am attaching a Jupyter notebook which I put together and went over in class. This contains the code for Parts 1, 2 and 4. Below are the steps we used for evaluating the assignment.

- We made sure the simple for loop based implementation works and is numerically stable for large m. Many students did not implement this, though it is clear in the question that you need to implement the for loop version first.
- Then we checked the vectorized implementation and made sure both match.
- Next, we made sure the vectorized code is faster and more efficient (for example, try with $n = 100000$).
- Finally, we made sure the gradients are correct and the numerical definition matched the implemented one.