# Assignment 1

Rishabh Iyer

January 22, 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 January 31st (Friday)

## 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 $y_i \in \{-1, +1\}$.
   - (2 Pointa) 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 $y_i \in \{-1, +1\}$.
   - (2 Pointa) 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 $y_i \in R$.
   - (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 $y_i \in R$.
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 $i$th 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 $\theta$ and $x^i$ as $m$-dimensional vectors. The Soft-Max Relaxation of the Inverse Propensity Objective is:

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

Compute the Gradient of this Loss function. In addition, **a bonus** 1 point if you can compute the Hessian of this Loss.

3. (2 Points) Implement Numerically correct versions of the following functions:
   - $L(x) = \log(1 + \exp(-x))$
   - $L(x) = \log(\exp(x1) + \exp(x2))$
   - $L(x) = \frac{\exp(x1)}{\exp(x1)+\exp(x2)}$

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

  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.