Homework 3 - Damped Newton's method for multi-class logistic regression

Acknowledgment: This assignment is inspired by the similar assignment by Ping Li when I took Statistical Computing Class at Cornell as a graduate student.

Attention: Because math rendering of .Rmd is not ideal, please see the enclosed pdf for correct rendering of all equations (an exact copy of this file)

Multi-class logistic regression

Logistic regression is one of the most popular classifiers. Consider the training data consisting of n samples (x_i, y_i), $x_i \in \mathbb{R}^p$, $y_i \in \{0, \dots, K-1\}$ (K classes, the coding starts from 0). For each class $k \in \{0, \dots, K-1\}$, we consider class probabilities for sample i conditioning on the corresponding vector of covariates x_i:

$$P(y_i = k|x_i) = p_k(x_i), \quad \sum_{k=0}^{K-1} p_k(x_i) = 1.$$

We assume the following model holds for class probabilities

$$p_k(x_i) = \frac{e^{x_i^\top \beta_k}}{\sum_{l=0}^{K-1} e^{x_i^\top \beta_l}}.$$

Unlike binary case where there is only one β, we now have K vectors $\beta_k \in \mathbb{R}^p$, $k \in \{0, \dots, K-1\}$, one for each class. Because of the constraint that class probabilities sum to one, the above model is over-parametrized (in binary logistic, we had only one β rather than two, as we can always express one probability as 1 minus all the others). Thus, typical implementations of multi-class logistic regression pick one of the classes as a reference. However, the over-parametrization problem is solved by adding ridge regularization, that is by considering the following objective function

$$f(\beta) = \left[-\sum_{i=1}^{n} \left\{ \sum_{k=0}^{K-1} 1(y_i = k) \log p_k(x_i) \right\} + \frac{\lambda}{2} \sum_{k=0}^{K-1} \sum_{j=1}^{p} \beta_{k,j}^2 \right], \quad \text{where} \quad p_k(x_i) = \frac{e^{x_i^\top \beta_k}}{\sum_{l=0}^{K-1} e^{x_i^\top \beta_l}}.$$

with some $\lambda > 0$ over $\beta = (\beta_0, \dots, \beta_{K-1}) \in \mathbb{R}^{p \times K}$. Here $1(y_i = k)$ is the indicator function, that is it is equal to one when $y_i = k$ and is equal to zero when $y_i \neq k$. The ridge regularization makes the solution unique as it essentially looks for K vectors β_k that satisfy the above model **and have the minimal euclidean norm**.

Thus, the objective function $f(\beta)$ depends on the supplied training data $X \in \mathbb{R}^{n \times p}$ (with 1st column being all ones to account for intercept) and $y \in \{0, \dots, K-1\}^n$; and its argument is matrix $\beta = (\beta_0, \dots, \beta_{K-1}) \in \mathbb{R}^{p \times K}$, where each column corresponds to specific β_k.

We find the matrix β by minimizing the above $f(\beta)$. Once the minimizer $\beta^* \in \mathbb{R}^{p \times K}$ is found, the classification for a new $x \in \mathbb{R}^p$ is performed by assigning x to the class with the largest probability $p_k(x)$, where

$$p_k(x) = \frac{e^{x^\top \beta_k^*}}{\sum_{l=0}^{K-1} e^{x^\top \beta_l^*}}.$$

Damped Newton's method implementation

In this assignment, we will implement Damped Newton's method to minimize $f(\beta)$.

Let $P_k = P_k(X; \beta) \in \mathbb{R}^n$ be a vector containing class specific probabilities $p_k(x_i)$ as defined above (depend on β). Let $W_k = W_k(X; \beta)$ be a $n \times n$ diagonal matrix with diagonal elements $w_{kii} = p_k(x_i)(1 - p_k(x_i))$

(depend on β as $p_k(x_i)$ depend on β). There are K total vectors P_k, and K total matrices W_k. It can be shown that the Damped Newton's update with learning rate $\eta > 0$ for $f(\beta)$ has the form

$$\beta_k^{(t+1)} = \beta_k^{(t)} - \eta(X^\top W_k X + \lambda I)^{-1}\left[X^\top\{P_k - 1(Y = k)\} + \lambda\beta_k^{(t)}\right], \quad k = 0, \ldots, K-1;$$

where $1(Y = k) \in \mathbb{R}^n$ is a vector with elements $1(y_i = k)$. Here both W_k and P_k depend on $\beta^{(t)}$.

Task 1: Prove that the formula above for β_k update indeed corresponds to damped Newton's method for minimizing $f(\beta)$ (e.g. derive gradient and Hessian, and plug in). Upload your derivations to this repository as a pdf file called **Derivations.pdf**

- You can hand-write, and then transfer to pdf using one of the phone scanner apps

OR

- If you are familiar with R markdown and prefer to use it, you can create **Derivations.Rmd**, and transfer that into pdf; upload both to Github

OR

- You can use LaTex, and then upload the pdf

Starter code

Task 2: FunctionsLR.R contains the wrapper for the following function

```
LRMultiClass(X, y, Xt, yt, numIter, eta, lambda, beta_init)
```

Here `X`, `y` are the training data, and `Xt`, `yt` are the testing data. This function will first perform some checks on the input arguments (as specified), then evaluate the objective function/class assignments at starting point `beta_init`, and then proceed to implement damped Newton's update for exactly `numIter` iterations. For simplicity, we will only run the algorithm for this fixed number of iterations. Note that the 1st iteration is the 1st time you update the beta and errors (with iteration 0 being the values at starting point).

Within each iteration, you will update β according to the update above, calculate the objective value at new beta, and calculate classification error using that beta (in %) on both training and test data. You are welcome to create as many additional functions as you want (and put them all within **FunctionsLR.R**), but I will explicitly test only `LRMultiClass` function for input/output concordance. More instructions are provided within the comments of the wrapper code. As with the k-means algorithm, you are not allowed to use any external libraries.

Things to keep in mind when implementing:

- the number of classes K should be determined based on supplied input
- the class coding goes from 0 to $K-1$, whereas R matrix/vector indexing starts from 1. There are several ways to account for this and there is not a single solution here. Also note that you can always recode the classes any way you want within the function as long as you get correct output (from 0 to $K-1$) on the specified input (from 0 to $K-1$).
- the errors are returned as % rather than proportions, which means that for 20% error rate the returned number should be 20 (rather than 0.2)
- calculating $(X^\top W_k X + \lambda I)^{-1}$ is tricky as the matrix W_k may be too large to explicitly store in memory. You should avoid storing W_k as a matrix.
- you would not be able to completely eliminate for loops (both because of iterations and because of K classes), however you want to avoid for loops over samples for class assignments

- you should check your code on simple examples before proceeding to the data (i.e. do I calculate objective function correctly? what happens if I use two normal populations? does the objective value improve across iterations? how does the classification error change across iterations?). You will be expected to save your checks in **Tests.R** file. I will use automatic tests to check that your code is correct on more than just the data example with different combinations of parameters.

Application to Letter data

Task 3: You are given two datasets to see the application of your code:

- **letter-train.txt**, **letter-test.txt** - you will use these data of 16 features to assign one of the 26 labels (corresponding to different letters in the alphabet, comes from images)
- the file **Test_letter.R** contains some code to load the data and extract covariates/assignments. You need to complete it with addition of intercept columns, run your algorithm and check the performance based on provided graphical plots. You will also be asked to time the function on your computer (We will redo the timing again but it should give you some idea). More comments are provided within the code.

Grading for this assignment

Your assignment will be graded based on

- derivations of damped Newton's update *(5% of the grade)*
- correctness *(45% of the grade)*

Take advantage of objective function values over iterations and error rates as a way to indirectly check the correctness of your function. For the letter data and default values, I get around 22% error rate on training data, and around 26% error rate on testing data after 50 iterations.

- speed of implementation (you need to be smart in calculating Hessian to pass the requirement) *(30% of the grade)*

My code takes around 4.3 seconds on my laptop for the letter data with 50 iterations, $\lambda = 1$ and $\eta = 0.1$. You will get full credit if your code runs **at most 3 times slower** than mine. You will get +5 **bonus** points if your **completely correct** code on 1st submission is faster than mine (median your time/median mine time < 0.9).

- code style/documentation *(10% of the grade)*

You need to comment different parts of the code so it's clear what they do, have good indentation, readable code with names that make sense. See guidelines on R style, and posted grading rubric.

- version control/commit practices *(10% of the grade)*

I expect you to start early on this assignment, and work gradually. You want to commit often, have logically organized commits with short description that makes sense. See guidelines on good commit practices, and posted grading rubric.