Homework X - SGD for multi-class classification with neural nets

Acknowledgment: This assignment is inspired by softmax classifier assignment in CS231n.

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 revisited (review from before)

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, typically one of the classes is picked as a reference. However, the over-parametrization problem is solved by adding ridge regularization, that is by considering the following loss function

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

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**.

In machine learning literature, it is common to call $f(\beta)$ the **softmax** loss. It 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.

NN implementation

In this assignment, we will implement a two layer fully connected neural network classifier using the above loss function. The network will have p inputs and K outputs, where each output

$$o_k(x_i) = \max(0, x_i^\top W_1 + b_1)^\top W_{2k} + b_{2k} \in \mathbb{R},$$

or all k outputs in vector form

$$o(x_i) = \max(0, x_i^\top W_1 + b_1)^\top W_2 + b_2 \in \mathbb{R}^K.$$

The corresponding class probabilities are

$$p_k(x_i) = \frac{e^{o_k(x_i)}}{\sum_{l=0}^{K-1} e^{o_l(x_i)}}.$$

The loss function puts ridge penalty on both weights terms $W_1 \in \mathbb{R}^{p \times h}$ and $W_2 \in \mathbb{R}^{h \times K}$

$$f(W_1, b_1, W_2, b_2) = \left[-\frac{1}{n} \sum_{i=1}^{n} \left\{ \sum_{k=0}^{K-1} 1(y_i = k) \log p_k(x_i) \right\} + \frac{\lambda}{2} (\|W_1\|_F^2 + \|W_2\|_F^2) \right].$$

You will need to complete **FunctionsNN.R** starter code to implement the following functions:

- initialization of W_1, W_2, b_1, b_2
- calculation of the loss function when $\lambda = 0$, its gradient with respect to outputs $o_k(x)$ when $\lambda = 0$, and misclassification error (in %, that is 20 corresponds to 20% error rate)
- one pass (forward and backward), use $\lambda = 0$ in forward pass, and directly incorporate λ into gradient for backward pass
- error calculation for validation data using supplied b_1, W_1, b_2, W_2 (forward pass only)
- full NN train function keeping track of train and validation error at each epoch

Things to keep in mind when implementing:

- unless the number of classes K is a specified input, it should be determined based on supplied y
- 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)
- 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

Warning: A working implementation of HW3 will make it much easier to complete the codes

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_NN.R** contains some code to load the data and extract covariates/assignments. First, it repeats your HW3 assignment (as a reminder, if you keep training, you should get around 19.5% error on train, and 25% error on test with linear classifier)
- The default given implementation of neural net with given seeds should give you much better errors on train (~6%) and test (~16%). Experiment with hidden unit size, number of epochs, ridge parameter lambda, and learning rate to see if you can obtain an even better result.

Grading for this assignment

Your assignment will be graded based on

- correctness of functions *(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.

- playing with parameters on letter example to get better error *(5% of the grade)*
- speed of implementation (you need to be smart in calculating Hessian to pass the requirement) *(30% of the grade)*

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.