ECE 4803, Fall 2020

Homework #6

Due Tuesday, October 20, at 11:59pm

1. Prepare a one paragraph summary of what we talked about in class since the last assignment. I do not want just a bulleted list of topics, I want you to use complete sentences and establish context (Why is what we have learned relevant? How does it connect with other classes?). The more insight you give, the better.

2. Consider the function
$$f(x) = \log\left(1 + e^{ax}\right),$$
where a is a constant.

   (a) Show that f is *not* strongly convex. [Hint: Recall the second derivative computed in homework 1.]

   (b) Determine a value of M for which f is M-smooth.

   (c) Now consider
$$\tilde{f}(x) = \log\left(1 + e^{ax}\right) + \delta x^2,$$
where $\delta > 0$ is a constant. Now show that $\tilde{f}$ is both strongly convex and M-smooth (and give the associated constants m and M).

3. **Logistic regression** is a simple, but surprisingly powerful, tool for solving a fundamental problem in machine learning: learning a *classifier* that can distinguish between two classes from a set of training data. Specifically, suppose that we have a set of *training data* $(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)$, where each $\boldsymbol{x}_n \in \mathbb{R}^D$ is a "feature vector" and each $y_n \in \{0, 1\}$ is a "label" that indicates which of two classes $\boldsymbol{x}_n$ corresponds to. The goal in learning a classifier is to find a function h such that $h(\boldsymbol{x})$ predicts the correct label y for $\boldsymbol{x}$ that we have never seen before. We can do this by trying to learn a function h that gives us $h(\boldsymbol{x}_n) = y_n$ on (most of) the training set. In this problem you will explore this idea and implement it on a simple example.

   Consider the **logistic** function
$$g(t) = \frac{1}{1 + e^{-t}}.$$

   Note that $g(t)$ is always a number between 0 and 1. In logistic regression, we set $t = \boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b$, where $\boldsymbol{a} \in \mathbb{R}^D$ and $b \in R$, so that for any $\boldsymbol{x}$ we can compute $g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b)$, returning a number between 0 and 1. Our goal is to learn the parameters $\boldsymbol{a}$ and b so that we can interpret $g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b)$ as an estimate of the probability that $\boldsymbol{x}$ belongs to class 1 (and hence, $1 - g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b)$ is the probability that $\boldsymbol{x}$ belongs to class 0). To form a classifier, we then simply compare $g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b)$ to a threshold in order to estimate the label based on which one has a higher predicted probability, i.e.,
$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b) \geq \frac{1}{2}, \\ 0 & \text{if } g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x} + b) < \frac{1}{2}. \end{cases}$$

(a) Show that $h(\boldsymbol{x})$ can equivalently be written as

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{a}^T\boldsymbol{x} + b \geq 0 \\ 0 & \text{if } \boldsymbol{a}^T\boldsymbol{x} + b < 0. \end{cases}$$

(b) Draw a picture illustrating the set $\{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{a}^T\boldsymbol{x} + b \geq 0\}$ for

$$\boldsymbol{a} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad b = \frac{1}{2}.$$

(c) The key question in fitting a logistic regression model is deciding on how to set $\boldsymbol{a}$ and b based on the training data. To see how this is done, note that if $y_n = 1$, we would like to have $g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b) \approx 1$, in which case

$$\log(g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b)) \approx 0.$$

Similarly, if $y_n = 0$, then we would like $g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b) \approx 0$, or said differently, $1 - g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b) \approx 1$, in which case

$$\log(1 - g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b)) \approx 0.$$

Note that in both cases, the terms on the left-hand side are always negative, so that to make them ≈ 0 corresponds to making these terms as *large* as possible. Combining these desired criteria together, we can express the problem of fitting $\boldsymbol{a}$ and b as

$$\underset{\boldsymbol{a},b}{\text{maximize}} \sum_{n:y_n=1} \log(g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b)) + \sum_{n:y_n=0} \log(1 - g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b)).$$

It is somewhat more convenient to re-express this as the equivalent problem:

$$\underset{\boldsymbol{a},b}{\text{maximize}} \sum_{n=1}^{N} y_n \log(g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b)) + (1 - y_n) \log(1 - g(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b)).$$

Show that, by plugging in the formula for g, this problem reduces to

$$\underset{\boldsymbol{a},b}{\text{maximize}} \sum_{n=1}^{N} y_n(\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b) - \log(1 + e^{\boldsymbol{a}^{\mathrm{T}}\boldsymbol{x}_n + b}).$$

(d) We would now like to use an iterative algorithm like gradient descent to solve this optimization problem. To do this, we will need to calculate the gradient. Note that if we use the notation

$$\widetilde{\boldsymbol{x}}_n = \begin{bmatrix} \boldsymbol{x}_n \\ 1 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{a} \\ b \end{bmatrix},$$

then we can re-write our objective function as

$$f(\boldsymbol{\theta}) = \sum_{n=1}^{N} y_n \boldsymbol{\theta}^{\mathrm{T}} \widetilde{\boldsymbol{x}}_n - \log(1 + e^{\boldsymbol{\theta}^{\mathrm{T}} \widetilde{\boldsymbol{x}}_n}).$$

With this notation, compute a formula for the gradient $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ and the Hessian matrix $\boldsymbol{D}_f^2(\boldsymbol{\theta})$. [Hint: You may use the answers from homework 1 to avoid any actual computation here, but be careful as the notation in this problem is a little different...]

(e) Do you think $f(\boldsymbol{\theta})$ is M-smooth for some finite value of M? Do you think $f(\boldsymbol{\theta})$ is strongly convex? (You do not have to calculate M or m here.)

(f) Implement a solver for estimating $\boldsymbol{\theta}$ using gradient descent. Test your implementation on the dataset produced by the following code:

```
import numpy as np
from sklearn import datasets

np.random.seed(2020) # Set random seed so results are repeatable
x,y = datasets.make_blobs(n_samples=100,n_features=2,centers=2,cluster_std=6.0)
```

Use an initial guess of $\boldsymbol{\theta}^{(0)} = \mathbf{0}$. To begin, consider a fixed step size of $\alpha \approx 0.001$ (although you should feel free to play around with α a bit). Report how many iterations your algorithm takes to converge. Remember that you are trying to *minimize* $-f(\boldsymbol{\theta})$. If your algorithm is not converging, a sign error is a likely culprit.

(g) Now implement the backtracking line search strategy described in the notes, and create a modified version of gradient descent in which you update α at each iteration using a line search. Experiment with the c and ρ. Report the c and ρ that work best, and how many iterations gradient descent requires when using backtracking.

(h) Next implement a solver using the "heavy ball method". First try setting $\alpha \approx 0.001$ and $\beta \approx 0.95$ and run the code with these fixed step sizes. Play around with α and β to see if you can get any improvement. Report the values of α and β that work best, and how many iterations the method requires. Then try using backtracking (using the same parameters as before) to adjust α at each iteration (you can leave β fixed here). Now how many iterations are required?

(i) Next implement a solver using "Nesterov's optimal method". Try both using a fixed stepsize of $\alpha \approx 0.001$ as well as backtracking. Use the rule of thumb $\beta = (k-1)/(k+2)$. Report how many iterations are required for both versions.

(j) Finally, implement a solver using Newton's method. Report the number of iterations required.