LIGN 167: Problem Set 2

October 19, 2023

Collaboration policy: You may collaborate with up to three other students on this problem set (that is, total group size of 4). You should submit **one response** per group, using group submissions on Gradescope. When you submit your work, you must indicate who was in the group, and what each of your individual contributions were.

Starter code: We are providing starter code for the problem set. This starter code contains function signatures for all of the functions that you are defining. You **must** use this starter code when writing answers to the problems. If you do not follow the format of the starter code, **you will not receive credit.**

IMPORTANT: For each problem, you are expected to do 4 things:

1. Provide the correct answer to the problem.
2. Provide the solution that GPT-3.5 gives.
3. If there are errors in the responses from GPT-3.5, explain where they occur.
4. For each problem, include links to your conversations with GPT-3.5.

Please submit this work in two files. One file should be named **ps2.py**. This should contain the correct answer to the problem. The second file should be named **gpt.py**. This should contain the code generated by GPT-3.5, as well as your explanations of the mistakes that occurred.

In this problem set you will be implementing gradient descent for optimizing real-valued functions. For the rest of the class, we will be most interested in applying gradient descent for optimizing statistical models, including neural networks.

In class we defined logistic regression in the following manner. We have a dataset that consists of two parts: $X = \{\vec{x}_1, ..., \vec{x}_n\}$ and $Y = \{y_1, ..., y_n\}$. Each element $\vec{x}_i$ in X is a k-dimensional vector: $\vec{x}_i = (x_{i,1}, ..., x_{i,k})$. Each element y_i in Y is a Boolean variable: $y_i \in \{0, 1\}$.

Our goal is to predict the value of each y_i from the corresponding input value $\vec{x}_i$. We want to find parameter values $\vec{a}$ and b that maximize the following probability:

$$P(Y|X, \vec{a}, b) = \prod_{i=1}^{n} P(y_i|\vec{x}_i, \vec{a}, b) \tag{1}$$

Here $P(y_i|\vec{x}_i, \vec{a}, b)$ is defined by the logistic distribution:

$$P(y_i|\vec{x}_i, \vec{a}, b) = \begin{cases} \frac{1}{1+e^{-(\vec{a}\cdot\vec{x}_i+b)}}, & \text{if } y_i = 1 \\ 1 - \frac{1}{1+e^{-(\vec{a}\cdot\vec{x}_i+b)}}, & \text{if } y_i = 0 \end{cases} \tag{2}$$

We can equivalently write this distribution using the sigmoid function σ:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3}$$

$$P(y_i|\vec{x}_i, \vec{a}, b) = \begin{cases} \sigma(\vec{a}\cdot\vec{x}_i + b), & \text{if } y_i = 1 \\ 1-\sigma(\vec{a}\cdot\vec{x}_i + b), & \text{if } y_i = 0 \end{cases} \tag{4}$$

The parameter b is a scalar, while the parameter $\vec{a}$ is a k-dimensional vector: $\vec{a} = (a_1, ..., a_k)$

As discussed in class, the problem of maximizing $P(Y|X, \vec{a}, b)$ can be transformed into the problem of minimizing a particular loss function. We define the loss function L as follows:

$$\begin{aligned} L(\vec{a}, b|X, Y) &= -\frac{1}{n}\log P(Y|X, \vec{a}, b) && (5)\\ &= -\frac{1}{n}\log\prod_{i=1}^{n} P(y_i|\vec{x}_i, \vec{a}, b) && (6)\\ &= -\frac{1}{n}\sum_{i=1}^{n}\log P(y_i|\vec{x}_i, \vec{a}, b) && (7)\end{aligned}$$

We will use the following notation to denote the loss associated with datapoint $\vec{x}_i, y_i$:

$$\ell_i = -\log P(y_i|\vec{x}_i, \vec{a}, b) \tag{8}$$

In this notation, the total loss L can be written:

$$L(\vec{a}, b|X, Y) = \frac{1}{n}\sum_{i=1}^{n}\ell_i \tag{9}$$

The values of $\vec{a}, b$ that maximize $P(Y|X, \vec{a}, b)$ are the same values of $\vec{a}, b$ that minimize $L(\vec{a}, b|X, Y)$. Thus we will apply gradient descent to L in order to find the (approximately) optimal values of $\vec{a}, b$.

We will assume throughout that b=0. Thus we will only be trying to find the optimal value of $\vec{a}$.

We previously derived a formula for the partial derivatives $\frac{\partial \ell_i}{\partial a_j}$:

$$\begin{aligned} \frac{\partial \ell_i}{\partial a_j} &= -(y_i - P(y_i = 1|\vec{x}_i, \vec{a}))x_{i,j} && (10)\\ &= -(y_i - \sigma(\vec{x}_i\cdot\vec{a}))x_{i,j} && (11)\end{aligned}$$

The partial derivative of L can therefore be written as:

$$\frac{\partial L}{\partial a_j} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial \ell_i}{\partial a_j} \tag{12}$$

Problem 1. Write a function *logistic_positive_prob* that takes as input a vector x_i of dimension k and a vector a of dimension k. These vectors are assumed to be NumPy arrays of length k.

The function should return the probability $P(y_i = 1|\vec{x}_i, \vec{a})$, as defined by Equation 4. (Remember that we have set $b = 0$ throughout the problem set.) As starter code, we have provided an implementation of the sigmoid function σ in pset2.py (posted on Piazza).

Problem 2. Write a function *logistic_derivative_per_datapoint*, which takes four arguments: *y_i* (either 0 or 1), *x_i* (a k-dimensional NumPy array), a (a k-dimensional NumPy array), and j (an integer between 0 and $k-1$). It should return $\frac{\partial \ell_i}{\partial a_j}$, using formula 10 and the answer to the previous problem.

Problem 3. Write a function *logistic_partial_derivative*. It should take four arguments: y (a list of n 0's or 1's), x (a list of n NumPy arrays; each NumPy array should have length k), a (a k-dimensional NumPy array), and j (an integer between 0 and $k-1$). The variables y and x are each Python lists of length n; the i'th element of the lists, $x[i]$ and $y[i]$, represents the i'th observation in our dataset, x_i and y_i.

The function should return the partial derivative of the loss function, $\frac{\partial L}{\partial a_j}$, as given in Equation 12. It should do this using the function *logistic_derivative_per_datapoint*.

Problem 4. Write a function *compute_logistic_gradient* that takes three arguments: a, y, and x. The arguments a, y, and x should have the same types as in Problem 3.

The function should return a NumPy Array of length k (the same length as input a). This NumPy array should contain the gradient of the loss function L, i.e. ∇L. It should use the function *logistic_partial_derivative* from Problem 3 to do this.

Problem 5. Write a function *gradient_update*, which takes three arguments: a, lr, and *gradient*. The variable a is a NumPy array of length k, and it represents our current guess for the parameter vector $\vec{a}$. The variable lr is the learning rate, which is a real number greater than 0. The variable *gradient* is a NumPy array of length k, which represents the gradient at the current time step.

The function should return the updated value for the parameter vector a, after applying the gradient update rule with learning rate equal to lr.

Problem 6. Write a function *gradient_descent_logistic*, which takes five arguments: *initial_a*, lr, *num_iterations*, y, and x. The variables y and x should have the same types as in previous problems. The variable *initial_a* is a NumPy array of length k, which is our initial guess for the value of the parameters a. The variable lr is again the learning rate. The variable *num_iterations* is an integer greater than 0, which is the number of iterations that we will run gradient descent for.

The function should run the gradient descent algorithm for *num_iterations* iterations. These iterations of gradient descent should optimize the logistic regression parameter vector $\vec{a}$, given the input dataset y and x. It should return a NumPy array which represents the final estimate of the parameter vector $\vec{a}$.

(Use the *gradient_update* function that you wrote in the previous problem.)

Problem 7. In your starter code, you will find an implementation of stochastic gradient descent for logistic regression. The next several problems will be about this PyTorch implementation. You should install PyTorch for these problems.

First, read through this PyTorch tutorial: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

The class *TorchLogisticClassifier* is defined at the top of your PyTorch code. What does the function *__init__* do? Which line in the starter code calls the function *__init__*?

Problem 8. What mathematical function is computed by the *forward* method of the class *TorchLogisticClassifier*? What line of the starter code calls the function *forward*?

Problem 9. Explain, step-by-step, what the inner loop (for d in dataset) in function *nonbatched_gradient_descent* is doing. How do the lines correspond to steps of stochastic gradient descent? You can ignore *optimizer.zero_grad*() for this problem.

Problem 10. What does *optimizer.zero_grad*() do, and why is it necessary for the algorithm to work correctly? What happens if you remove *optimizer.zero_grad*() from the code?

Problem 11. Write a function *batched_gradient_descent* (the template is given to you). The function should implement a batched version of stochastic gradient descent, with batch size given by the argument *batch_size*. Each batch should consist of consecutive examples in the list *dataset*. For example, if the batch size is 2, then the first batch consists of the first two elements of *dataset*, the second batch consists of the third and fourth elements of *dataset*, and so on.

After your gradient descent loop is complete, your function should return the model.

Note that the function *nonbatched_gradient_descent* (which was already included for you) implements stochastic gradient descent with batch size 1. Your function should generalize this to any batch size. You can assume that the length of the dataset is divisible by the batch size.

Formally, let batch $B = \{d_i, ..., d_{i+k}\}$, where k is the batch size. Your function should loop over batches. For each batch B, your function should compute the following gradient:

$$\nabla_{\vec{w}} L(\vec{w}|B) = \frac{1}{k} \sum_{d \in B} \nabla_{\vec{w}} L(\vec{w}|d) \tag{13}$$

$\vec{w}$ is the weight/parameter vector for the model, and L is the loss function. The batch gradient is a sum of the gradient at each individual data point (divided by the batch size k).

Hint: Think about your answer to Problem 10. In order to obtain a mathematically correct solution, you only need to change a couple of lines of code.