# LIGN 167: Problem Set 2

October 18, 2018

**Collaboration policy**: You may collaborate with up to two other students on this problem set. You must write up your own answers to the problems; do not just copy and paste from your collaborators. You must also submit your work individually. If you do not submit a copy of the problem set under your own name, you will not get credit. When you submit your work, you must indicate who you worked with, and what each of your individual contributions were.

**Getting started**: We will be uploading a file called pset2.py to Piazza. This file will contain some starter code for the problem set (some functions that you should call in your answers), as well as function signatures for your answers. Please use these function signatures for creating your functions.

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 $\sigma$:

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

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

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:

$$L(\vec{a}, b|X, Y) = -\log P(Y|X, \vec{a}, b) \tag{5}$$

$$= -\log \prod_{i=1}^{n} P(y_i|\vec{x}_i, \vec{a}, b) \tag{6}$$

$$= -\sum_{i=1}^{n} \log P(y_i|\vec{x}_i, \vec{a}, b) \tag{7}$$

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:

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

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 $\frac{\partial \ell_i}{\partial a_j}$:

$$\frac{\partial \ell_i}{\partial a_j} = -(y_i - P(y_i = 1|\vec{x}_i, \vec{a}))x_{i,j} \tag{10}$$

$$= -(y_i - \sigma(\vec{x}_i \cdot \vec{a}))x_{i,j} \tag{11}$$

The partial derivative of $L$ can therefore be written as:

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

**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 $\sigma$ 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 $k$ (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 $k$ (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. $\nabla L$. 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 the starter code that you've been given (in pset2.py), there is a function *create_dataset*. This function returns two lists, $x$ and $y$. The variable $x$ is a list of 100 2-dimensional NumPy arrays, and $y$ is a list of Boolean values sampled from the logistic distribution, given the elements of $x$ as input. The output $y$ is generated assuming that the parameter vector $\vec{a}$ equals (10,10).

Call *create_dataset* once to create values of $x$ and $y$. Using matplotlib.pyplot.scatter, plot the elements of $x$, along with the labels $y$. The plot should be two-dimensional, with color used to indicate the value of each element $y_i$ in $y$. See here for an example: https://stackoverflow.com/questions/17682216/scatter-plot-and-color-mapping-in-python

**Problem 8.** Call *create_dataset* a number of times to create some datasets. On each of these datasets, call *gradient_descent_logistic* to get an estimate for the parameter vector $\vec{a}$. Use a learning rate $lr = 0.01$, *initial_a* equal to (-1,-1), and *num_iterations* $= 1000$.

What values of $\vec{a}$ are returned by your gradient descent algorithm? Does this suggest that gradient descent is accurately solving the classification problem?

**Important: For this question and the other free-response questions below, you must write out your answers in your problem set submission.**

**Problem 9.** Repeat Problem 8, but change *initial_a* to different values (for example (-1,0) or (0,-1)). What do you notice about the estimated value of $\vec{a}$ that is returned each time? What does this tell you about the behavior of gradient descent on logistic regression?

(It is recommended that you call *create_dataset* a number of times for each value of *initial_a*, in order to get a clear picture of what is going on.)

**Problem 10.** Repeat Problem 8, but change the learning rate $lr$ to different values. For example, try $lr = 0.001$ and $lr = 0.1$. What do you notice about the estimated value of $\vec{a}$ that is returned each time. What does this tell you about the behavior of gradient descent on logistic regression?

**Problem 11.** A common technique in statistical modeling, including neural networks, is called *regularization*. Regularization involves adding an additional penalty term to the loss function. L2-regularization (also known as ridge regression) involves the following modification to Equation 5 (note that we are dropping the parameter $b$, which we have assumed is set to 0):

$$L_{ridge}(\vec{a}|X,Y) = -\sum_{i=1}^{n} \log P(y_i|\vec{x}_i,\vec{a}) + \lambda\|a\|^2 \tag{13}$$

$$= -\sum_{i=1}^{n} \log P(y_i|\vec{x}_i,\vec{a}) + \lambda\sum_{i=1}^{k} a_i^2 \tag{14}$$

Here we have used the definition of the norm $\|a\|^2 = \sum_{i=1}^{k} a_i^2$. The term $\lambda > 0$ is a real number.

Write a function *logistic_l2_partial_derivative*, which takes four arguments: $y$, $x$, $a$, and $j$. These four arguments should be of the same types as in Problem 3, where you wrote *logistic_partial_derivative*. The function should return the partial derivative of this new loss function, $\frac{\partial L_{ridge}}{\partial a_j}$.

Your function should use *logistic_derivative_per_datapoint*, which you defined in Problem 2.

You need to choose a value of $\lambda$ in order to compute the partial derivative. Set $\lambda$ equal to 0.1 for right now. You will be changing this value in later problems.

**Problem 12.** Write a function *compute_logistic_l2_gradient*, which takes three arguments: $a$, $y$, and $x$. The arguments $a$, $y$, and $x$ should have the same types as in Problem 4.

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 new regularized loss function $L_{ridge}$, i.e. $\nabla L_{ridge}$. It should use the function *logistic_l2_partial_derivative* from Problem 11 to do this.

**Problem 13.** In this problem we will be writing a function which generalizes the gradient descent function that you wrote in Problem 6. In Problem 6, you wrote out the gradient descent algorithm, for the specific case of (ordinary) logistic regression. In this problem, you will write a gradient descent algorithm which can work for optimizing arbitrary functions.

Define a function $gradient\_descent$, which takes six arguments: $initial\_a$, $lr$, $num\_iterations$, $y$, $x$, and $gradient\_fn$. The first five variables should have the same types as in Problem 6.

The last argument, $gradient\_fn$, should be a function which takes some inputs, and computes a gradient. You have already written two such functions: $compute\_logistic\_gradient$ in Problem 4 and $compute\_logistic\_l2\_gradient$ in Problem 12.

You should assume that $gradient\_fn$ takes three arguments, $a$, $y$, and $x$, as in Problems 4 and 12. You can also assume that it returns a NumPy array with the same length as $a$.

The function $gradient\_descent$ should run the gradient descent algorithm for $num\_iterations$ iterations. These iterations of gradient descent should optimize the parameter vector $\vec{a}$, given the input dataset $y$ and $x$. Each gradient update should use the function $gradient\_fn$, which has been given as an argument, in order to compute the gradient.

The function $gradient\_descent$ should return a NumPy array which represents the final estimate of the parameter vector $\vec{a}$.

(Note: this problem is giving you a small introduction to *functional programming* in Python. If you are confused about how one function can take another function as an argument, see the first part of these lecture notes: https://courses.csail.mit.edu/6.01/spring08/handouts/week2/week2-notes.pdf)

**Problem 14.** Call $create\_dataset$ a number of times to create some datasets. On each of these datasets, use gradient descent to optimize the parameter vector $\vec{a}$, given the $L_{ridge}$ loss function defined in Equation 13.

More specifically, on each of your datasets, call $gradient\_descent$ (from Problem 13) using the $compute\_logistic\_l2\_gradient$ function (from Problem 12). This will return an estimate for the parameter vector $\vec{a}$, given the $L_{ridge}$ loss function.

Use a learning rate $lr = 0.01$, $initial\_a$ equal to (-1,-1), and $num\_iterations = 1000$.

Compare the estimated values of $\vec{a}$ that are returned here to the values that were returned in Problem 8 (which used the same parameter settings). What is different about the values of $\vec{a}$ that are returned here? Why does this difference exist?

**Problem 15.** Within Problem 11, when you were defining $logistic\_l2\_partial\_derivative$, you set the value $\lambda$ equal to 0.1. In this problem, we want to explore the effect of changing $\lambda$.

Decrease $\lambda$ in $logistic\_l2\_partial\_derivative$ to 0.01. Repeat what you did in Problem 14, with this new value of $\lambda$. What do you notice about the estimated values of $\vec{a}$, as compared to Problem 14?

Now increase $\lambda$, first to 1 and then to 10. What changes about the value of the estimated $\vec{a}$? What does $\lambda$ seem to be controlling?

At the end of this process, **please remember to change the value of $\lambda$ back to** 0.1 **in your definition of** $logistic\_l2\_partial\_derivative$. We want to be able to grade you fairly!