week4

January 25, 2021

1 Week 4: Numerical Methods and Feature Processing

Training or fitting a machine learning algorithm often involves solving a minimization problem of some form. For example, given a penalty parameter λ, LASSO requires solving the following minimization problem for the parameters $\beta_0, \beta_1, \ldots, \beta_p$:

$$\beta_0, \beta_1, \ldots, \beta_p = \arg \min_{b_0, b_1, \ldots, b_p} \sum_{i=1}^{n} (Y_i - b_0 - b_1 X_{1,i} - b_2 X_{2,i} - \cdots - \beta_p X_{p,i})^2 + \lambda \sum_{k=1}^{p} |b_k|$$

In the case that $\lambda = 0$ (even when $p >> n$!), this minimization has a closed form solution that is effeciently computed. However, in general with machine learning models a closed form formula for the solution is not available. Moreover, we are often fitting these machine learning models on large datasets or fitting them many times to choose our hyperparameters. So, it is important that we learn effecient numerical methods to solve minmization problems and know when these methods work well.

1.1 Gradient Descent: Concepts

1.1.1 An overview of minimization

In general, we will choose the parameters of our model to minize some (generally convex) cost function that depends on the data.

$$\theta_0 = \arg \min_{\theta} C(\theta; Y, X)$$

I review some basics of minimization below.

Suppose I am given a twice continuously-differentiable function $f(x)$. I want to solve for the value, x_0, that minimizes f. In order to check that x_0 is a local minimum for f, we need to check the following first and second order conditions:

$$f'(x) = 0 \tag{1}$$

$$f''(x) > 0 \tag{2}$$

If there are multiple points that satisfy the above conditions, we will have to compare the value of f at each of these points to find the true global minimum.

However, if f is a convex function, then we know that $f''(x) > 0$ for all values of x. Moreover, we know that there can only be one point x_0 such that $f'(x_0) = 0$. In this case, solving for a global minimum of f simply requires finding the one point x_0 that satisfies the first order condition:

$$f'(x_0) = 0$$

If f is well behaved then we can explicitly solve for an inverse for f', that is we can find f'^{-1} and solve for:

$$f'^{-1}(0) = x_0$$

For example, suppose $f(x) = x^2 + x$. We can easily verify that $f''(x) = 2 > 0$ so that f is a convex function. Then $f'(x) = 2x + 1$ has a well defined inverse, $f'^{-1}(y) = \frac{y-1}{2}$. So, in order to solve for the minimum of $f(x)$, we can solve for x_0:

$$f'^{-1}(0) = \frac{0-1}{2} = -\frac{1}{2}$$

In the case of simple linear regression, the cost function:

$$C(\theta; Y, X) = \sum_{i=1}^{n} (Y_i - \theta_0 - \theta_1 X_{1,i} - \cdots - \theta_p X_{p,i})^2$$

Is such that the derivative of the cost function is linear in θ and has a well defined inverse. We can solve for θ_0 via:

$$\theta_0 = \left(\mathbf{X}'\mathbf{X}\right)^{-1} \left(\mathbf{X}'\mathbf{Y}\right)$$

where $\mathbf{X}$ is a matrix of of our covariates and $\mathbf{Y}$ is a vector of our outcomes.

However, for regularized or penalized cost functions, like in LASSO, Ridge Regression, or Elastic Net, there is no well defined formula for the inverse of the derivative of the cost function. So, in order to solve these minimization problems we will need to numerically find the point at which the derivative of the cost function is equal to 0. The algorithm we will review to help us do this is called gradient descent.

1.1.2 Gradient Descent: Method and Implementation

The gradient of a multidimensional function $F : \mathbb{R}^p \to \mathbb{R}$ is just a vector of it's derivatives w.r.t all components of x, that is:

$$\nabla F(\bar{x}) = \begin{pmatrix} \frac{\partial F}{\partial x_1}(\bar{x}) \\ \vdots \\ \frac{\partial F}{\partial x_p}(\bar{x}) \end{pmatrix}$$

To minimize a multidimensional convex function it is necessary and suffecient to find the point at which the gradient is 0.

Gradient descent is founded off the observation from multivariable calculus that a multivariable function $F(x) : \mathbb{R}^p \to \mathbb{R}$, that we may be trying to minimize, decreases the fastest at a point $\bar{x}$ if one goes in the direction of the negative gradient: $-\nabla F(\bar{x})$.

With this in mind, the gradient descent algorithm for minimizing a cost function $C(\theta; Y, X)$ is as follows:

1. Guess an initial value for θ, call this θ_0.
2. Calculate the gradient of $C(\cdot; Y, X)$ at θ_0, $\nabla C(\theta_0; Y, X)$.

3. Update your guess in the direction of the negative gradient:

$$\theta_{\text{new}} = \theta_0 - \gamma \nabla C(\theta_0; Y, X)$$

4. Repeat starting at step one, setting $\theta_0 = \theta_{\text{new}}$. Do this for a maximum number of times (or untill the gradient is small enough that you think you are close enough to the mimumum).

There are a few things to note here. The first is that once we are close to the minimum, $\nabla C(\theta; Y, X)$ will be close to the 0 vector. So, our θ_{new} will be close to θ_0. This is called our algorithm "converging". Many gradient descent algorithms will stop after the change between the new theta and old theta is small enough (within some preset tolerance level). This generally means that your problem is close to the true solution.

The second is that the parameter γ has to be chosen. We can think of this as the size of the "step" that we take in each iteration. Large values of γ may bring you closer to the solution initially, as you are moving quickly down the gradient. However, as you approach the solution, a large value of γ may cause you to "over-shoot" and so may ultimately lead to a slower rate of convergence. In general is is best to try a few different values of γ to figure out which works best. Alternatively, if you have a lot of time and convex cost function, just set a small value of γ and let the algorithm run for a longer time.

Finally, note that, sometimes there is not a clean solution for the gradient of our cost function. This will be the case later on as we get to more advanced ML techniques like random forests and neural nets. In this case, we will have to numerically compute the derivatives as well. Don't worry about this now, but just now that there are methods and libraries in python that can help you do this.

1.1.3 Gradients of LASSO, Ridge Regression

For LASSO, the cost function $C(\cdot; Y, X)$ is given:

$$C(\mathbf{b}; Y, X) = \sum_{i=1}^{n} (Y_i - b_0 - b_1 X_{1,i} - \cdots - b_p X_{p,i})^2 + \lambda \sum_{j=1}^{p} |b_j|$$

In this case, the derivative of C with respect to a single parameter $b_j, j \in \{1, \ldots, p\}$ is given

$$\frac{\partial C}{\partial b_j}(\mathbf{b}; Y, X) = \sum_{i=1}^{n} (Y_i - b_0 - b_1 X_{1,i} - \cdots - b_p X_{p,i}) X_{j,i} + \lambda \cdot \text{sign}(b_j)$$

and the derivative with respect to β_0 is given by

$$\frac{\partial C}{\partial b_0}(\mathbf{b}; Y, X) = \sum_{i=1}^{n} (Y_i - b_0 - b_1 X_{1,i} - \cdots - b_p X_{p,i}) + \lambda \cdot \text{sign}(b_0)$$

Stacking these together gives us the gradient.

In ridge regression, the cost function is given:

$$C(\mathbf{b}; Y, X) = \sum_{i=1}^{n} (Y_i - b_0 - b_1 X_{1,i} - \cdots - b_p X_{p,i})^2 + \lambda \left(\sum_{j=1} b_j^2 \right)^{1/2}$$

In this case, the derivative of C with respect to a single parameter $\beta_j, j \in \{1, \ldots, p\}$ is given

$$\frac{\partial C}{\partial b_j}(\mathbf{b}; Y, X) = \sum_{i=1}^{n} (Y_i - b_0 - b_1 X_{1,i} - \cdots - b_p X_{p,i}) X_{j,i} + \lambda \left(\sum_{j=1}^{p} b_j^2 \right)^{-1/2} b_j$$

and the derivative with respect to β_0 is given by

$$\frac{\partial C}{\partial b_0}(\mathbf{b}; Y, X) = \sum_{i=1}^{n} (Y_i - b_0 - b_1 X_{1,i} - \cdots - b_p X_{p,i}) + \lambda \left(\sum_{j=1}^{p} b_j^2 \right)^{-1/2} b_0$$

Stacking these all together will give us a gradient.

1.2 Feature Processing

The last thing we will review here is how to generate features or variables and why we might want to do so.

1.2.1 Dealing with Categorical Data

Linear Regression, along with many other machine learning techniques, requires our input variables to be real valued. For this reason, in order to use categorical data, we may have to convert it into real values. We can do this in a couple ways.

The first way would be to convert the categorical random variable into numbers in a way that preserves some sense of order between the categorical random variables.

[ ]: