Will Monroe
CS 109

Lecture Notes #22
August 14, 2017

Logistic Regression

Based on a chapter by Chris Piech

Logistic regression is a classification algorithm[1] that works by trying to learn a function that approximates $P(Y|X)$. It makes the central assumption that $P(Y|X)$ can be approximated as a sigmoid function applied to a linear combination of input features. It is particularly important to learn because logistic regression is the basic building block of artificial neural networks.

Mathematically, for a single training data point $(\mathbf{x}, y)$, logistic regression assumes:

$$P(Y = 1 \mid \mathbf{X} = \mathbf{x}) = \sigma(z) \text{ where } z = \theta_0 + \sum_{i=1}^{m} \theta_i x_i$$

This assumption is often written in the equivalent forms:

$$P(Y = 1 \mid \mathbf{X} = \mathbf{x}) = \sigma(\theta^T \mathbf{x}) \qquad \text{where we always set } x_0 \text{ to be 1}$$
$$P(Y = 0 \mid \mathbf{X} = \mathbf{x}) = 1 - \sigma(\theta^T \mathbf{x}) \qquad \text{by total law of probability}$$

Using these equations for probability of $Y \mid X$ we can create an algorithm that selects values of theta that maximize that probability for all data. I am first going to state the log probability function and partial derivatives with respect to theta. Then later we will (a) show an algorithm that can chose optimal values of theta and (b) show how the equations were derived.

An important thing to realize is that: given the best values for the parameters (θ), logistic regression often can do a great job of estimating the probability of different class labels. However, given bad, or even random, values of θ it does a poor job. The amount of "intelligence" that your logistic regression machine learning algorithm has depends on how good its values of θ are.

Notation

Before we get started I want to make sure that we are all on the same page with respect to notation. In logistic regression, θ is a vector of parameters of length m and we are going to learn the values of those parameters based off of n training examples. The number of parameters should be equal to the number of features of each data point (see section 1).

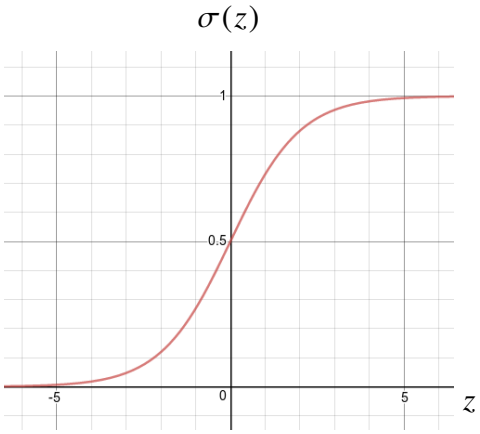
Two pieces of notation that we use often in logistic regression that you may not be familiar with:

$$\theta^T \mathbf{x} = \sum_{i=1}^{m} \theta_i x_i = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The superscript T in $\theta^T \mathbf{x}$ represents a matrix transpose; the operation $\theta^T \mathbf{x}$ is equivalent to taking the *dot product* of the vectors θ and $\mathbf{x}$, or simply a weighted sum of the components of $\mathbf{x}$ (with θ containing the weights).

[1] Yes, this is a terribly confusing name, given that **regression** refers to tasks that require predicting continuous values. Perhaps *logistic classification* would have been better.

The function $\sigma(z) = \frac{1}{1+e^{-z}}$ is called the **logistic function** (or *sigmoid function*); it looks like this:



An important quality of this function is that it maps all real numbers to the range (0, 1). In logistic regression, $\sigma(z)$ turns an arbitrary "score" z into a number between 0 and 1 that is interpreted as a probability. Positive numbers become high probabilities; negative numbers become low ones.

Log Likelihood

In order to chose values for the parameters of logistic regression, we use maximum likelihood estimation (MLE). As such we are going to have two steps: (1) write the log-likelihood function and (2) find the values of θ that maximize the log-likelihood function.

The labels that we are predicting are binary, and the output of our logistic regression function is supposed to be the probability that the label is one. This means that we can (and should) interpret each label as a Bernoulli random variable: $Y \sim \text{Ber}(p)$ where $p = \sigma(\theta^T \mathbf{x})$.

To start, here is a super slick way of writing the probability of one data point (recall this is the equation form of the probability mass function of a Bernoulli):

$$P(Y = y|X = \mathbf{x}) = \sigma(\theta^T \mathbf{x})^y \cdot \left[1 - \sigma(\theta^T \mathbf{x})\right]^{(1-y)}$$

Now that we know the probability mass function, we can write the likelihood of all the data:

$$L(\theta) = \prod_{i=1}^{n} P(Y = y^{(i)} \mid X = \mathbf{x}^{(i)}) \qquad \text{the likelihood of independent training labels}$$

$$= \prod_{i=1}^{n} \sigma(\theta^T \mathbf{x}^{(i)})^{y^{(i)}} \cdot \left[1 - \sigma(\theta^T \mathbf{x}^{(i)})\right]^{(1-y^{(i)})} \qquad \text{substituting the likelihood of a Bernoulli}$$

And if you take the log of this function, you get the log likelihood for logistic regression. The log likelihood equation is:

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Recall that in MLE the only remaining step is to chose parameters (θ) that maximize log likelihood.

Gradient of Log Likelihood

Now that we have a function for log-likelihood, we simply need to chose the values of theta that maximize it. Unfortunately, if we try just setting the derivative equal to zero, we'll quickly get frustrated: there's no closed form for the maximum. However, we can find the best values of theta by using an optimization algorithm. The optimization algorithm we will use requires the partial derivative of log likelihood with respect to each parameter. First I am going to give you the partial derivative (so you can see how it is used); we'll derive it a bit later:

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Gradient Ascent Optimization

Our goal is to choosing parameters (θ) that maximize likelihood, and we know the partial derivative of log likelihood with respect to each parameter. We are ready for our optimization algorithm.

In the case of logistic regression we can't solve for θ mathematically. Instead we use a computer to chose θ. To do so we employ an algorithm called gradient ascent (a classic in optimization theory). The idea behind gradient ascent is that gradients point "uphill". If you continuously take small steps in the direction of your gradient, you will eventually make it to a local maximum. In the case of logistic regression you can prove that the result will always be a global maximum.

The update to our parameters that results in each small step can be calculated as:

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{old}})}{\partial \theta_j^{\text{old}}}$$

$$= \theta_j^{\text{old}} + \eta \cdot \sum_{i=1}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Where η is the magnitude of the step size that we take. If you keep updating θ using the equation above, you will converge on the best values of θ. You now have an intelligent model. Here is the gradient ascent algorithm for logistic regression in pseudo-code:

```
Initialize: θ_j = 0 for all 0 ≤ j ≤ m
Repeat many times:
    gradient[j] = 0 for all 0 ≤ j ≤ m
    For each training example (x, y):
        For each parameter j:
            gradient[j] += x_j (y − 1 / (1 + e^{−θ^T x}))
    θ_j += η * gradient[j] for all 0 ≤ j ≤ m
```

It is also common to have a parameter θ_0 that is added as a constant to the $\theta^T \mathbf{x}$ inside the sigmoid. Rather than computing special derivatives for θ_0, we can simply define an additional feature x_0 that always takes the value 1. Taking a weighted average then results in adding θ_0, the weight for x_0.

Derivations

In this section we provide the mathematical derivations for the gradient of log-likelihood. The derivations are worth knowing because these ideas are heavily used in Artificial Neural Networks.

Our goal is to calculate the derivative of the log likelihood with respect to each theta. To start, here is the definition for the derivative of a sigmoid function with respect to its inputs:

$$\frac{\partial}{\partial z}\sigma(z) = \sigma(z)[1-\sigma(z)] \qquad \text{to get the derivative with respect to } \theta\text{, use the chain rule}$$

Take a moment and appreciate the beauty of the derivative of the sigmoid function. The reason that sigmoid has such a simple derivative stems from the natural exponent in the sigmoid denominator.

Since the likelihood function is a sum over all of the data, and in calculus the derivative of a sum is the sum of derivatives, we can focus on computing the derivative of one example. The gradient of theta is simply the sum of this term for each training data point.

First I am going to show you how to compute the derivative the hard way. Then we are going to look at an easier method. The derivative of gradient for one data point $(\mathbf{x}, y)$:

$$\begin{aligned}
\frac{\partial LL(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} y \log \sigma(\theta^T \mathbf{x}) + \frac{\partial}{\partial \theta_j}(1-y)\log[1-\sigma(\theta^T \mathbf{x}] && \text{derivative of sum of terms} \\
&= \left[\frac{y}{\sigma(\theta^T \mathbf{x})} - \frac{1-y}{1-\sigma(\theta^T \mathbf{x})}\right]\frac{\partial}{\partial \theta_j}\sigma(\theta^T \mathbf{x}) && \text{derivative of } \log f(x) \\
&= \left[\frac{y}{\sigma(\theta^T \mathbf{x})} - \frac{1-y}{1-\sigma(\theta^T \mathbf{x})}\right]\sigma(\theta^T \mathbf{x})[1-\sigma(\theta^T \mathbf{x})]x_j && \text{chain rule + derivative of } \sigma \\
&= \left[\frac{y-\sigma(\theta^T \mathbf{x})}{\sigma(\theta^T \mathbf{x})[1-\sigma(\theta^T \mathbf{x})]}\right]\sigma(\theta^T \mathbf{x})[1-\sigma(\theta^T \mathbf{x})]x_j && \text{algebraic manipulation} \\
&= \left[y-\sigma(\theta^T \mathbf{x})\right]x_j && \text{cancelling terms}
\end{aligned}$$

Derivatives Without Tears

That was the hard way. Logistic regression is the building block of artificial neural networks. If we want to scale up, we are going to have to get used to an easier way of calculating derivatives. For that we are going to have to welcome back our old friend the chain rule (of derivatives). By the chain rule:

$$\begin{aligned}
\frac{\partial LL(\theta)}{\partial \theta_j} &= \frac{\partial LL(\theta)}{\partial p}\cdot\frac{\partial p}{\partial \theta_j} && \text{where } p = \sigma(\theta^T \mathbf{x}) \\
&= \frac{\partial LL(\theta)}{\partial p}\cdot\frac{\partial p}{\partial z}\cdot\frac{\partial z}{\partial \theta_j} && \text{where } z = \theta^T \mathbf{x}
\end{aligned}$$

Chain rule is the decomposition mechanism of calculus. It allows us to calculate a complicated partial derivative ($\frac{\partial LL(\theta)}{\partial \theta_j}$) by breaking it down into smaller pieces.

$$
\begin{aligned}
LL(\theta) &= y \log p + (1-y)\log(1-p) && \text{where } p = \sigma(\theta^T \mathbf{x}) \\
\frac{\partial LL(\theta)}{\partial p} &= \frac{y}{p} - \frac{1-y}{1-p} && \text{taking the derivative}
\end{aligned}
$$

$$
\begin{aligned}
p &= \sigma(z) && \text{where } z = \theta^T \mathbf{x} \\
\frac{\partial p}{\partial z} &= \sigma(z)[1-\sigma(z)] && \text{taking the derivative of the sigmoid}
\end{aligned}
$$

$$
\begin{aligned}
z &= \theta^T \mathbf{x} && \text{as previously defined} \\
\frac{\partial z}{\partial \theta_j} &= x_j && \text{only } x_j \text{ interacts with } \theta_j
\end{aligned}
$$

Each of those derivatives was much easier to calculate. Now we simply multiply them together.

$$
\begin{aligned}
\frac{\partial LL(\theta)}{\partial \theta_j} &= \frac{\partial LL(\theta)}{\partial p} \cdot \frac{\partial p}{\partial z} \cdot \frac{\partial z}{\partial \theta_j} \\
&= \Big[\frac{y}{p} - \frac{1-y}{1-p}\Big] \cdot \sigma(z)[1-\sigma(z)] \cdot x_j && \text{substituting in for each term} \\
&= \Big[\frac{y}{p} - \frac{1-y}{1-p}\Big] \cdot p[1-p] \cdot x_j && \text{since } p = \sigma(z) \\
&= [y(1-p) - p(1-y)] \cdot x_j && \text{multiplying in} \\
&= [y-p]x_j && \text{expanding} \\
&= [y - \sigma(\theta^T \mathbf{x})]x_j && \text{since } p = \sigma(\theta^T \mathbf{x})
\end{aligned}
$$