Logistic Regression

Classification Problem

Let's talk about the classification problem. This is like a regression problem except that, the target variable y we now want to predict take on only a small number of discrete values. For now, we focus on binary classification problem in which y can take on only two values, 0 and 1. Most of the cases that we look can be generalized to multi-class case. For instance, if we are trying to build a spam classifier for email, then x^i may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols "-" and "+". Given x^i corresponding to the y^i is also called the label of the training example.

Logistic Regression

We could approach the classification problem ignoring the fact that y is discrete valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$

To fix it let's change the form of a hypotheses function $h_\theta(x)$. we will chose :

$$\begin{aligned} h_\theta(x) &= g(\theta^T x) \\ &= \frac{1}{1 + e^{\theta^T x}} \end{aligned}$$

Here, function g is called logistic function/ sigmoid function given by :

$$g(z) = \frac{1}{1 + e^{-z}}$$

Here, is a plot that showing $g(z)$.

Notice that $g(z)$ value goes to 1 as $z \to \infty$ and $g(z)$ value goes to 0 as $z \to -\infty$. Moreover, $g(z)$ and hence also $h(x)$ is always bounded between 0 and 1. As before, we are keeping the convention of letting $x_0 = 1$ so that $\theta^T x = \theta_0 + \sum_{j=1}^{d} \theta_j x_j$.

For now, let's take the choice of g as given. Other functions that smoothly increase from 0 to 1 can also be used, but for a couple of reasons that we'll see later (when we talk about GLMs, and when we talk about generative learning algorithms), the choice of the logistic function is a fairly natural one. Before moving on, here's a useful property of the derivative of the sigmoid function, which we write as g' :

$$\begin{aligned} g'(z) &= \frac{d}{dz}\,\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1+e^{-z})}\left(1-\frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)) \end{aligned}$$

So, given the logistic regression model, how do we fit θ for it? Following how we saw least squares regression could be derived as the maximum likelihood estimator under a set of assumptions, let's endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood.

Let us assume that :

$$\begin{gathered} p(y=1|x;\theta) = h_\theta(x), \\ \text{And, } p(y=0|x;\theta) = 1-h_\theta(x) \end{gathered}$$

Note that this can be written more compactly as :

$$p(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y}$$

Assuming that the n training examples were generated independently, we can then write down the likelihood of the parameters as :

$$\begin{aligned} \mathcal{L}(\theta) &= p(\vec{y}|x;\theta) \\ &= \prod_{i=1}^{n} p(y^i|x^i;\theta) \\ &= \prod_{i=1}^{n}(h_\theta(x^i))^{y^i}(1-h_\theta(x^i))^{1-y^i} \end{aligned}$$

As before, it will be easier to maximize the log likelihood :

$$\begin{aligned} l(\theta) &= log\ \mathcal{L}(\theta) \\ &= \sum_{i=1}^{n} y^i\, log(h_\theta(x^i)) + (1-y^i)\, log\,(1-h_\theta(x^i)) \end{aligned}$$

How do we maximize the likelihood? Similar to our derivation in the case of linear regression, we can use gradient ascent. Written in vectorial notation, our updates will therefore be given by $\theta := \theta + \alpha \nabla_\theta l(\theta)$. (Note the positive rather than negative sign in the update formula, since we're maximizing, rather than minimizing, a function now.) Let's start by working with just one training example (x, y), and take derivatives to derive the stochastic gradient ascent rule :

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} l(\theta_j) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{(1-g(\theta^T x))} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{(1-g(\theta^T x))} \right) g(\theta^T x)(1-g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= (y(1-g(\theta^T x)) - (1-y) g(\theta^T x)) x_j \\
&= (y - h_\theta(x)) x_j
\end{aligned}$$

Above, we used the fact that : $g'(z) = g(z)(1-g(z))$. This therefore gives us the stochastic gradient ascent rule :

$$\theta_j = \theta_j + \alpha(y^i - h_\theta(x^i)) x_j$$

If we compare this to the LMS update rule, we see that it looks identical; but this is not the same algorithm, because $h_\theta(x)$) is now defined as a non-linear function $\theta^T x^i$. Nonetheless, it's a little surprising that we end up with the same update rule for a rather different algorithm and learning problem. Is this coincidence, or is there a deeper reason behind this? We'll answer this when we get to GLM models.
