Part II
Classification and logistic regression

Let's now talk about the classification problem. This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. For now, we will focus on the **binary classification** problem in which y can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-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 $x^{(i)}$, the corresponding $y^{(i)}$ is also called the **label** for the training example.

5 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 $y \in \{0, 1\}$.

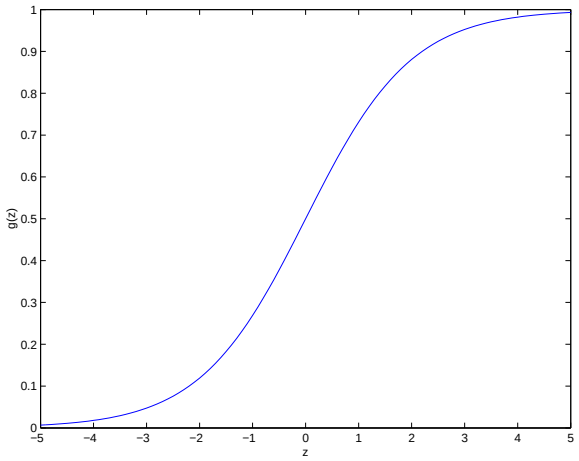
To fix this, let's change the form for our hypotheses $h_\theta(x)$. We will choose

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

where

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

is called the **logistic function** or the **sigmoid function**. Here is a plot showing $g(z)$:



Notice that $g(z)$ tends towards 1 as $z \to \infty$, and $g(z)$ tends towards 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 $\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$.

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':

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

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

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

Note that this can be written more compactly as

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

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

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

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

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

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 $\theta := \theta + \alpha \nabla_\theta \ell(\theta)$. (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:

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

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

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

If we compare this to the LMS update rule, we see that it looks identical; but this is *not* the same algorithm, because $h_\theta(x^{(i)})$ is now defined as a non-linear function of $\theta^T x^{(i)}$. 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. (See also the extra credit problem on Q3 of problem set 1.)

6 Digression: The perceptron learning algorithm

We now digress to talk briefly about an algorithm that's of some historical interest, and that we will also return to later when we talk about learning theory. Consider modifying the logistic regression method to "force" it to output values that are either 0 or 1 or exactly. To do so, it seems natural to change the definition of g to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

If we then let $h_\theta(x) = g(\theta^T x)$ as before but using this modified definition of g, and if we use the update rule

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}.$$

then we have the **perceptron learning algorithm**.

In the 1960s, this "perceptron" was argued to be a rough model for how individual neurons in the brain work. Given how simple the algorithm is, it will also provide a starting point for our analysis when we talk about learning theory later in this class. Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression and least squares linear regression; in particular, it is difficult to endow the perceptron's predictions with meaningful probabilistic interpretations, or derive the perceptron as a maximum likelihood estimation algorithm.