A short intro to

Logistic Regression

Daniel Heesch

1 Motivation

Logistic regression is a simple yet powerful classification method for binary data, and it can readily be extended to multiple classes. Let's motivate it by first considering how one could use the simplest regression model, linear regression, to perform classification. We'll soon find it wholly inadequate for what we want to achieve and it will be instructive to see exactly why. The linear regression model is simply

$$f(x) = \theta^T x$$

where we assume $x_0 = 1$, so that, for example, in two dimensions

$$f(x) = x_1\theta_1 + \theta_0.$$

As in many parameter estimation problems, θ is found by minimising some loss function that captures how close our current fit is. When we make distributional assumptions, the loss function is often in terms of the likelihood of the data: the optimal θ is the one under which the observed data has the highest probability. For example, linear regression typically assumes that the dependent variable is normally distributed around the mean $f(x)$. The maximum likelihood solution can then be shown to be the θ that minimises the sum of squared errors (the differences between predicted and correct values):

$$\sum_i (\theta^T x^{(i)} - y^{(i)})^2.$$

Given our optimal fit, one *could* use it as a very crude classifier by thresholding the output value:

$$y = \begin{cases} 0 & \theta^T x < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

but the 0.5 is entirely arbitrary and does not perform all too well on our toy dataset:

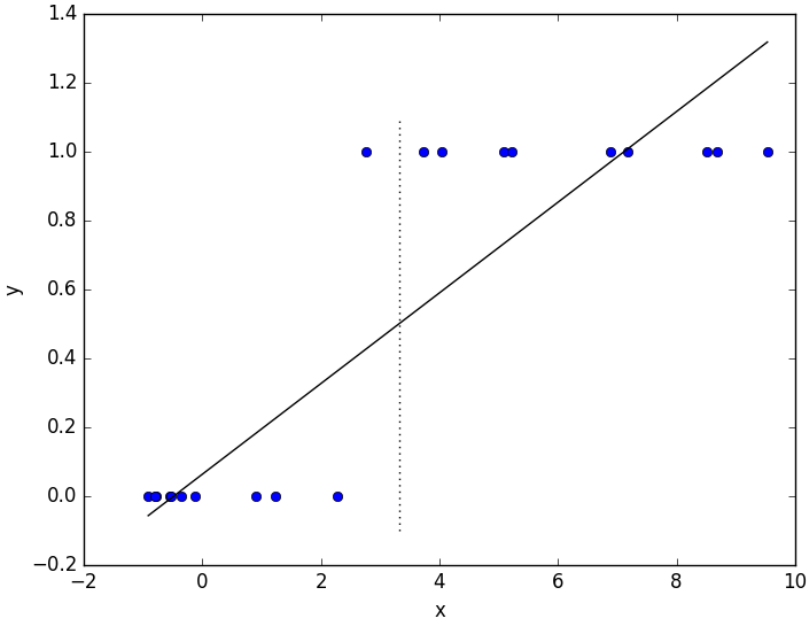

Figure 1: Linear regression in one dimension. The dotted line marks the decision boundary on x when the threshold is chosen to be 0.5.

This should not be a surprise as the loss function makes no reference to a classification model. It compares the observed data $\in \{0, 1\}$ with $\theta^T x$, not to the actual class, so the optimal θ makes data points hug the line as closely as possible, no matter what effect that has on classification performance.

What's to be done? We can keep the linear model and choose a more sensible loss function that explicitly measures classification accuracy. This would lead to linear classification. Or we change the model altogether to something that is better at representing the binary nature of the prediction problem. Logistic regression is one such model.

2 Logistic Regression

2.1 The model

Whereas linear regression maps the input data $\in \mathbb{R}^n$ to $\mathbb{R}$, logistic regression predicts values in the open interval between 0 and 1. Those bounds are by design because they can be viewed as probabilities. To get from $\mathbb{R}$ to $(0, 1)$, the dot product $\in \mathbb{R}$ from linear regression is first mapped onto $\mathbb{R}^+$ through

exponentiation, and then mapped onto (0,1) by dividing the result by 1+ itself. Combining these two mappings in one function, we get

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

where, in this context, $z = w^T x$. σ is known as the sigmoid function. The class probabilities of the two classes are then modelled as

$$Pr\{y = 1|z\} = \frac{1}{1+e^{-z}}$$

and

$$Pr\{y = 0|z\} = 1 - \sigma(z) = \frac{e^{-z}}{1+e^{-z}}.$$

Just as in linear regression, this function is smooth and monotonic, so we can apply gradient descent and be guaranteed to find the optimal solution.

Once we have the optimal parameters, the fitted curve may look something like this.

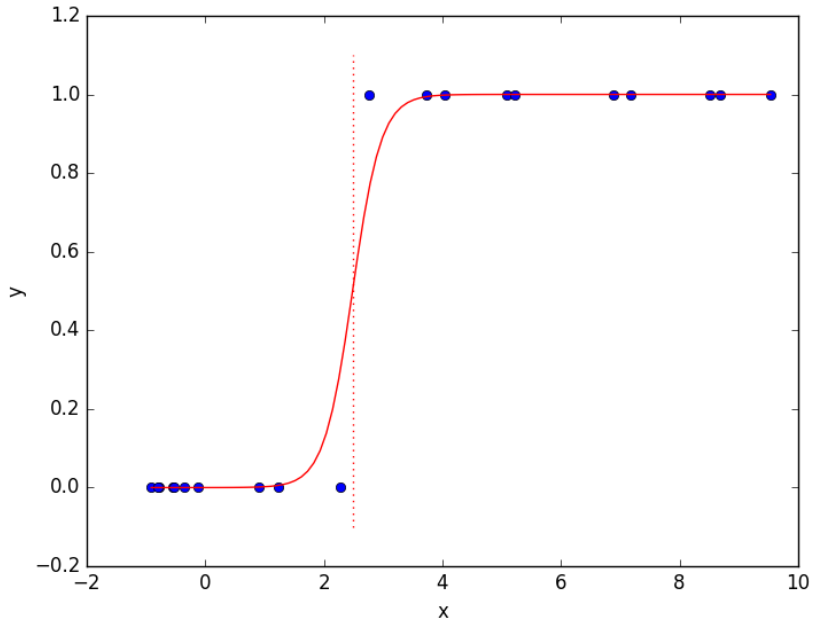

Figure 2: Logistic regression in one dimension.

2.2 The loss function

Again we seek to find the parameters that maximise the likelihood function, that is those under which the data is most probable. This is particularly easy to do because logistic regression explicitly models the class probabilities as $\sigma(z)$ and $1 - \sigma(z)$.

The overall likelihood is given by the product of probabilities over all data points, which for two classes we can write concisely as

$$\mathcal{L}(\theta) = \prod_i \sigma(z_i)^{y^{(i)}} (1 - \sigma(z_i))^{1-y^{(i)}}$$

where we write $z_i = \theta^T x_i$. The exponents are either 1 or 0 and thus have the effect of selecting exactly one of the two probabilities. Taking logs and changing sign to make it a loss (the smaller the better), we get:

$$-\log \mathcal{L}(\theta) = -\sum_i y^{(i)} \log \sigma(z_i) + (1 - y^{(i)}) \log(1 - \sigma(z_i)).$$

2.3 From Regression to Classification

Because the function gives us the class probabilities, it is easy to turn the regression output into a classifier. We simply choose the class with the highest probability. In the two-class setting, this is the one with probability > 0.5.

$$y = \begin{cases} 0 & \frac{e^{\theta^T x}}{1+e^{\theta^T x}} < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

Note how the optimality of the threshold of 0.5, unlike in the linear regression case, follows from our interpretation of $f(x)$ as a probability.

2.4 Softmax Regression

Before we look at the gradients, let's generalise the logistic model to multiple classes. The corresponding model is referred to as softmax regression or multinomial logistic regression. Each class is given its own sigmoid function with its own parameter vector, subject to the condition that, for any given input x, they all sum to one. Specifically, we define

$$P_j(x, \theta) = Pr\{y = j | x, \theta\} = \frac{e^{\theta_j^T x}}{\sum_i e^{\theta_i^T x}} \qquad j = 1, \ldots, k.$$

You may wonder why there is now one parameter vector for each class, whereas the logistic regression model with two classes has only one. We can rewrite the model with $k-1$ parameter vectors simply by dividing denominator and denumerator by, say, $e^{\theta_k^T x}$. This would give

$$P_j(x,\theta) = \begin{cases} \frac{e^{(\theta_j-\theta_k)^T x}}{1+\sum_{i=1}^{k-1} e^{(\theta_i-\theta_k)^T x}} & j = 1,\ldots,k-1 \\ \frac{1}{1+\sum_{i=1}^{k-1} e^{(\theta_i-\theta_k)^T x}} & j = k \end{cases}$$

and thus $k-1$ parameter vectors. In practice, one uses the original, overparameterised formula with one parameter vector per class. It has the benefit of treating all classes uniformly but it requires a regularisation term to be added to the loss function so that parameters are kept small. The function $e^{x_i}/\sum_j x_j$ is referred to as the softmax function.

2.5 Loss function for the general case

The maximum likelihood function for softmax regression is similar to that of logistic regression, except that we cannot write it quite so concisely because the class variable is not constrained to be 0 or 1. The likelihood of θ is

$$\mathcal{L}(\theta|x,y) = \prod_{i=1}^{m} P_{y^{(i)}}(x^{(i)},\theta).$$

For each input $x^{(i)}$, we choose the P of the corresponding class $y^{(i)}$. The loss function becomes

$$-\log \mathcal{L}(\theta|x,y) = -\sum_{i=1}^{m} \log P_{y^{(i)}}(x^{(i)},\theta).$$

For the purpose of gradient descent, we average over the m datapoints and add a regularisation term to favour small parameters. Note that we do not want to constrain the bias value, so will not include the first component of each parameter vector, θ_{i0}. Our final objective function is

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \log P_{y^{(i)}}(x^{(i)},\theta) + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=1}^{n}\theta_{ij}^2.$$

To apply gradient-based techniques such as stochastic gradient descent, all we need is the derivatives of J with respect to each of the k parameter vectors. These are easy to derive.

2.6 Gradients of loss function

We want to find, for each class c the derivatives of J with respect to the parameter vector θ_c. The derivative is itself a vector, hence we write it using the ∇ operator:

$$\nabla_{\theta_c} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \nabla_{\theta_c} \log P_{y^{(i)}}(x^{(i)}, \theta) + \nabla_{\theta_c}\mathcal{R}(\theta).$$

So the ith element of the gradient with respect to θ_c is

$$\frac{\partial J(\theta)}{\partial \theta_{ci}} = -\frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \theta_{ci}} \log P_{y^{(i)}}(x^{(i)}, \theta) + \frac{\partial}{\partial_{\theta_c i}}\mathcal{R}(\theta).$$

Let's ignore the regularisation term for the moment and focus on the sum, and within it on one specific term corresponding to a specific training example $(x^{(i)}, y^{(i)})$. Note now that the derivative with respect to θ_c depends on whether or not the given example is of class c. If it is, the parameter appears in both the denominator and denumerator of P, otherwise it's only present in the denominator (the normalisation term). Other than this little subtlety, the gradients are easy to obtain. Using the fact that

$$\frac{d}{dx}\log f(x) = \frac{f'(x)}{f(x)},$$

the first term is just

$$\nabla_{\theta_c} \log P_{y^{(i)}}(x^{(i)}, \theta) = \frac{1}{P_{y^{(i)}}(x^{(i)}, \theta)} \nabla_{\theta_c} P_{y^{(i)}}(x^{(i)}, \theta).$$

Writing S for the normalising sum, we find, for $y^{(i)} = l$:

$$\nabla_{\theta_c} \log P_{y^{(i)}}(x^{(i)}, \theta) \quad = \quad \frac{x^{(i)} e^{\theta_c^T x^{(i)}} S - x^{(i)} e^{2\theta_c^T x^{(i)}}}{S^2} \frac{S}{e^{\theta_c^T x^{(i)}}} = x^{(i)}[1 - P_c(x^{(i)}, \theta)]$$

Meanwhile, for $y^{(i)} \neq l$,

$$\nabla_{\theta_c} \log P_{y^{(i)}}(x^{(i)}, \theta) \quad = \quad \frac{-x^{(i)} e^{\theta_{y^{(i)}}^T x} e^{\theta_m^T x}}{S^2} \frac{1}{e^{\theta_{y^{(i)}}^T x}} = -x^{(i)} P_c(x^{(i)}, \theta).$$

We can write this concisely with an indicator function as follows

$$\nabla_{\theta_c} \log P_{y^{(i)}}(x^{(i)}, \theta) = x^{(i)}(\mathbb{1}\{y^{(i)} = l\} - P_c(x^{(i)}, \theta)).$$

$\mathbb{1}\{y^{(i)} = l\}$ is 1 if $y^{(i)} = l$ and 0 otherwise. The derivative of the regularisation

term is just

$$\nabla_{\theta_c} R(\theta) = \lambda \theta_c.$$

We need to make sure that the gradient for the regularisation term is not added for the bias parameter θ_{i0}, so we multiply it with a vector $z = [0\ 1\ \ldots\ 1]$ of length n to give our total gradient:

$$\begin{aligned}
\nabla_{\theta_c} J(\theta) &= -\frac{1}{m} \left(\sum_{i=1}^{m} x^{(i)} \left[\mathbb{1}\{y^{(i)} = l\} - P_c(x^{(i)}, \theta) \right] \right) + \lambda z \theta_c \\
&= \frac{1}{m} \left(\sum_{i=1}^{m} x^{(i)} \left[P_c(x^{(i)}, \theta) - \mathbb{1}\{y^{(i)} = l\} \right] \right) + \lambda z \theta_c.
\end{aligned}$$

3 Wrap up

Once you have the gradients, all that's left is to choose your favourite gradient descent algorithm and plug them in. Because logistic regression requires comparatively few parameters (of the order $O(nm)$ with n and m being the number of input dimensions and the number of classes), there is less risk of overfitting and a model with good generalisation can be obtained with relatively few data points. Before jumping onto something more complex, you may save yourself some time by first fitting a humble logistic classifier. It could be all you need.