Handout 03: Logistic Regression (Optional)

Classification

Many of the most interesting problems in statistical learning require the prediction of a response variable that takes on categorical values. Examples include spam detection, image recognition, predicting the outcome of a sporting event, and estimating the potential for a cataclysmic weather event. Our methods have so far focused only on the estimation of a continuous response variable. The flexibility of the linear model, fortunately, allows us to extend it in ways capable of estimating categorical responses.

To start, consider a task where the response variable takes on only two categories. As a first step, we can create a numeric response vector Y the same way we constructed indicator variables for the matrix X. For example, in a classifier to predict the outcome of a sporting event we might define Y as

$$y_i = \begin{cases} 1, & \text{category}_i = \text{win} \\ 0, & \text{category}_i \neq \text{win} \end{cases}. \tag{1.1}$$

Assuming we also have a model matrix X there is no algorithmic reason that we cannot directly apply linear regression using this binary response Y. How can we use the output of such a model to predict the classes on a new dataset? The fitted values $\widehat{Y}$ will not be exactly equal to 0 or 1. We therefore discretize the fitted values according to

$$z_i = \begin{cases} 1, & \widehat{y}_i \geq 0.5 \\ 0, & \widehat{y}_i < 0.5 \end{cases} \tag{1.2}$$

and assign the predicted classes according the values z_i. Splitting by the value of 0.5 is reasonable being halfway between 0 and 1, though of course it is possible to pick a different cutoff value depending on the needs of a particular problem.

Many of the nice theoretical properties of linear regression, which unfortunately we have not been able to cover much due to time constrains and the need for more probability theory, are broken when we have discrete input data points. While the strategy above can work well when needed, a modification based on these theoretical properties often slightly outperforms the naïve approach.

Model of the log-odds ratio

I want to start by defining the quantity p_i to be the probability that y_i is equal to 1. Otherwise, with probability $1 - p_i$ the variable y_i will be equal to 0. In the naive approach above we are implicitly modeling:

$$p_i = x_i^t b = \sum_j x_{i,j} \cdot b_j. \tag{1.3}$$

One problem with this is that $x_i^t b$ can take on any real value but the probability must be contained in the interval $[0, 1]$. Instead, what if we modelled the *odds-ratio*. This is the probability of observing a 1 divided by the probability of observing a 0 (gamblers typically represent bets in terms of the odds-ratio). The odds-ratio can take on any positive value, which is closer but still not the entire real line. To fix this, we will model the *logarithm* of the odds-ratio as follows:

$$\log\left(\frac{p_i}{1-p_i}\right) = x_i^t b. \tag{1.4}$$

Some arithmetic gets us a formula for p_i directly:

$$\begin{align}
\frac{p_i}{1-p_i} &= e^{x_i^t b} \tag{1.5}\\
p_i &= (1-p_i) \cdot e^{x_i^t b} \tag{1.6}\\
p_i + p_i \cdot e^{x_i^t b} &= e^{x_i^t b} \tag{1.7}\\
p_i \cdot \left(1 + e^{x_i^t b}\right) &= e^{x_i^t b} \tag{1.8}\\
p_i &= \frac{e^{x_i^t b}}{1 + e^{x_i^t b}} \tag{1.9}
\end{align}$$

It will also be nice to have a concise formula for $1 - p_i$, the probability of observing $y_i = 0$:

$$\begin{align}
1 - p_i &= 1 - \frac{e^{x_i^t b}}{1 + e^{x_i^t b}} \tag{1.10}\\
&= \frac{1 + e^{x_i^t b}}{1 + e^{x_i^t b}} - \frac{e^{x_i^t b}}{1 + e^{x_i^t b}} \tag{1.11}\\
&= \frac{1}{1 + e^{x_i^t b}} \tag{1.12}
\end{align}$$

Logistic regression

Logistic regression uses the model defined above to take data vectors x_i and produce predicted probability p_i that given y_i is equal to one. How do we go about using training data to find the optimal values for the vector b? The idea is to pick a vector b that maximizes the probability of observing our training data. If you have taken probability or statistics, this is what you have probably heard called the maximum likelihood estimator (MLE). How can we write down the probability of observing a certain set of values? Notice that we can write the probability of observing a particular y_i in a clever way without the need for conditional logic (if this is confusing, plug in $y_i = 0$ and $y_i = 1$ to see how this simplifies):

$$Pr(y_i) = p_i^{y_i} \cdot (1 - p_i)^{1-y_i}. \tag{1.13}$$

The probability of observing two independent events is the product of their probabilities. Assuming the data observations are independent, this gives us:

$$Pr(y) = \prod_i p_i^{y_i} \cdot (1 - p_i)^{1-y_i}. \tag{1.14}$$

We want to find a value for b that maximizes this quantity, but note that maximizing a quantity is the same as maximizing the logarithm of a quantity (the *maximum* itself is different, but the value that maximizes it is the same). Therefore we will use the negative logarithm of the value in Equation 1.14 as our loss function:

$$\begin{aligned} l(b) &= -\left(\sum_i y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)\right) && (1.15)\\ &= -\left(\sum_i \log(1 - p_i) + y_i \cdot \log\left[\frac{p_i}{1 - p_i}\right]\right) && (1.16) \end{aligned}$$

Here we can plug in our formula for $1 - p_i$ and p_i:

$$\begin{aligned} l(b) &= -\left(\sum_i \log\left[\frac{1}{1 + e^{x_i^t b}}\right] + y_i \cdot x_i^t b\right) && (1.17)\\ &= \sum_i \log(1 + e^{x_i^t b}) - y_i x_i^t b. && (1.18) \end{aligned}$$

Now, we want to minimize the quantity. What are the partial derivatives with respect to b_j?

$$\begin{aligned} \frac{\partial l(b)}{\partial b_j} &= \sum_i \frac{1}{1 + e^{x_i^t b}} \cdot e^{x_i^t b} \cdot x_{i,j} - y_i x_{i,j} && (1.19)\\ &= \sum_i (p_i \cdot x_{i,j} - y_i x_{i,j}) && (1.20)\\ &= \sum_i (p_i - y_i) \cdot x_{i,j} && (1.21) \end{aligned}$$

Writing this as a matrix equation, we can write down the entire gradient compactly as:

$$\nabla_b l(b) = X^t(p - y). \tag{1.22}$$

The gradient with respect to β of the log-likelihood is known as the *score function*. To find the maximum likelihood estimators we need to find where the gradient is equal to zero. Notice that at the minimizer the residuals $y - p$ are perpendicular to the column space of X. This is exactly what you showed held for the ordinary least squares solution as well!

Finally, note that if we take following two formulas:

$$\begin{aligned} p_i &= \frac{e^{x_i^t b}}{e^{x_i^t b} + 1} \\ 1 - p_i &= \frac{1}{1 + e^{x_i^t b}} \end{aligned}$$

And multiple both of these by:

$$1 = \frac{e^{-x_i^t b}}{e^{-x_i^t b}}$$

You will get the alternative forms:

$$\begin{aligned} p_i &= \frac{1}{1 + e^{-x_i^t b}} \\ 1 - p_i &= \frac{e^{-x_i^t b}}{e^{-x_i^t b} + 1} \end{aligned}$$

So if you multiply $x_i^t b$ by negative one, you flip the equations for p_i and $(1 - p_i)$. So, logistic regression is in some sense *symmetric* between 0 and 1. If you flip the signs of the regression vector β you will have the logistic model with the two classes flipped.

Solving the logistic score function with gradient descent

Unfortunately we cannot write down an analytic solution that sets the logistic score function to zero. We need an iterative way of finding a solution where the gradient is zero (or, at least, sufficiently small). The way that most statistical programs, such as R and Python, solve the logistic regression equation is using a second-order method known as Newton-Ralphson updating. In this context it is also sometimes called Iteratively Reweighted Least Squares (IRWLS). We are instead going to look at a different approach; it is not as good as IRWLS for this specific problem, but can be adapted far more readily to other estimators we will see in this course.

Gradient descent is an algorithm that tries to minimize a function by taking small steps in the direction of the negative gradient. Lets say that we start with a guess for the regression vector $b^{(0)}$. Gradient descent would then move to a new estimator defined by:

$$b^{(1)} = b^{(0)} - \rho \cdot \nabla_b l(b^{(0)}). \tag{1.23}$$

The same equation is used to move from $b^{(1)}$ to $b^{(2)}$, and so forth. In theory, it moves in the direction that $l(b)$ is decreasing the fastest. We can iterate this until either the values of b stop changing very much or the gradient is sufficiently small. The value ρ is a positive constant called the *learning rate*. Too large and the iteration will jump wildly over the input space and never converge on a good value for b. Too small and it will take forever to actually reach (if ever) a good value. You need to manually adjust ρ in order to find a good balance.