Chapter 12

Logistic Regression

12.1 Modeling Conditional Probabilities

So far, we either looked at estimating the conditional expectations of continuous variables (as in regression), or at estimating distributions. There are many situations where however we are interested in input-output relationships, as in regression, but the output variable is discrete rather than continuous. In particular there are many situations where we have binary outcomes (it snows in Pittsburgh on a given day, or it doesn't; this squirrel carries plague, or it doesn't; this loan will be paid back, or it won't; this person will get heart disease in the next five years, or they won't). In addition to the binary outcome, we have some input variables, which may or may not be continuous. How could we model and analyze such data?

We could try to come up with a rule which guesses the binary output from the input variables. This is called **classification**, and is an important topic in statistics and machine learning. However, simply guessing "yes" or "no" is pretty crude — especially if there is no perfect rule. (Why should there be?) Something which takes noise into account, and doesn't just give a binary answer, will often be useful. In short, we want probabilities — which means we need to fit a stochastic model.

What would be nice, in fact, would be to have conditional distribution of the response Y, given the input variables, $\Pr(Y|X)$. This would tell us about how precise our predictions are. If our model says that there's a 51% chance of snow and it doesn't snow, that's better than if it had said there was a 99% chance of snow (though even a 99% chance is not a sure thing). We have seen how to estimate conditional probabilities non-parametrically, and could do this using the kernels for discrete variables from lecture 6. While there are a lot of merits to this approach, it does involve coming up with a model for the joint distribution of outputs Y and inputs X, which can be quite time-consuming.

Let's pick one of the classes and call it "1" and the other "0". (It doesn't matter which is which. Then Y becomes an **indicator variable**, and you can convince yourself that $\Pr(Y=1) = \mathbf{E}[Y]$. Similarly, $\Pr(Y=1|X=x) = \mathbf{E}[Y|X=x]$. (In a phrase, "conditional probability is the conditional expectation of the indicator".)

This helps us because by this point we know all about estimating conditional expectations. The most straightforward thing for us to do at this point would be to pick out our favorite smoother and estimate the regression function for the indicator variable; this will be an estimate of the conditional probability function.

There are two reasons not to just plunge ahead with that idea. One is that probabilities must be between 0 and 1, but our smoothers will not necessarily respect that, even if all the observed y_i they get are either 0 or 1. The other is that we might be better off making more use of the fact that we are trying to estimate probabilities, by more explicitly modeling the probability.

Assume that $\Pr(Y=1|X=x) = p(x;\theta)$, for some function p parameterized by θ. parameterized function θ, *and* further assume that observations are independent of each other. The the (conditional) likelihood function is

$$\prod_{i=1}^{n} \Pr\left(Y=y_i|X=x_i\right) = \prod_{i=1}^{n} p(x_i;\theta)^{y_i}(1-p(x_i;\theta)^{1-y_i}) \tag{12.1}$$

Recall that in a sequence of Bernoulli trials $y_1,\ldots y_n$, where there is a constant probability of success p, the likelihood is

$$\prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i} \tag{12.2}$$

As you learned in intro. stats, this likelihood is maximized when $p=\hat{p}=n^{-1}\sum_{i=1}^{n} y_i$. If each trial had its own success probability p_i, this likelihood becomes

$$\prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} \tag{12.3}$$

Without some constraints, estimating the "inhomogeneous Bernoulli" model by maximum likelihood doesn't work; we'd get $\hat{p}_i = 1$ when $y_i = 1$, $\hat{p}_i = 0$ when $y_i = 0$, and learn nothing. If on the other hand we assume that the p_i aren't just arbitrary numbers but are linked together, those constraints give non-trivial parameter estimates, and let us generalize. In the kind of model we are talking about, the constraint, $p_i = p(x_i;\theta)$, tells us that p_i must be the same whenever x_i is the same, and if p is a continuous function, then similar values of x_i must lead to similar values of p_i. Assuming p is known (up to parameters), the likelihood is a function of θ, and we can estimate θ by maximizing the likelihood. This lecture will be about this approach.

12.2 Logistic Regression

To sum up: we have a binary output variable Y, and we want to model the conditional probability $\Pr(Y=1|X=x)$ as a function of x; any unknown parameters in the function are to be estimated by maximum likelihood. By now, it will not surprise you to learn that statisticians have approach this problem by asking themselves "how can we use linear regression to solve this?"

1. The most obvious idea is to let $p(x)$ be a linear function of x. Every increment of a component of x would add or subtract so much to the probability. The conceptual problem here is that p must be between 0 and 1, and linear functions are unbounded. Moreover, in many situations we empirically see "diminishing returns" — changing p by the same amount requires a bigger change in x when p is already large (or small) than when p is close to 1/2. Linear models can't do this.

2. The next most obvious idea is to let $\log p(x)$ be a linear function of x, so that changing an input variable *multiplies* the probability by a fixed amount. The problem is that logarithms are unbounded in only one direction, and linear functions are not.

3. Finally, the easiest modification of $\log p$ which has an unbounded range is the **logistic** (or **logit**) **transformation**, $\log \frac{p}{1-p}$. We can make *this* a linear function of x without fear of nonsensical results. (Of course the results could still happen to be *wrong*, but they're not *guaranteed* to be wrong.)

This last alternative is **logistic regression**.

Formally, the model logistic regression model is that

$$\log \frac{p(x)}{1-p(x)} = \beta_0 + x \cdot \beta \tag{12.4}$$

Solving for p, this gives

$$p(x; b, w) = \frac{e^{\beta_0 + x\cdot\beta}}{1 + e^{\beta_0 + x\cdot\beta}} = \frac{1}{1 + e^{-(\beta_0 + x\cdot\beta)}} \tag{12.5}$$

Notice that the over-all specification is a lot easier to grasp in terms of the transformed probability that in terms of the untransformed probability.[1]

To minimize the mis-classification rate, we should predict $Y = 1$ when $p \geq 0.5$ and $Y = 0$ when $p < 0.5$. This means guessing 1 whenever $\beta_0 + x \cdot \beta$ is non-negative, and 0 otherwise. So logistic regression gives us a **linear classifier**. The **decision boundary** separating the two predicted classes is the solution of $\beta_0 + x \cdot \beta = 0$, which is a point if x is one dimensional, a line if it is two dimensional, etc. One can show (exercise!) that the distance from the decision boundary is $\beta_0/\|\beta\| + x \cdot \beta/\|\beta\|$. Logistic regression not only says where the boundary between the classes is, but also says (via Eq. 12.5) that the class probabilities depend on distance from the boundary, in a particular way, and that they go towards the extremes (0 and 1) more rapidly when $\|\beta\|$ is larger. It's these statements about probabilities which make logistic regression more than just a classifier. It makes stronger, more detailed predictions, and can be fit in a different way; but those strong predictions could be wrong.

Using logistic regression to predict class probabilities is a *modeling choice*, just like it's a modeling choice to predict quantitative variables with linear regression.

[1]Unless you've taken statistical mechanics, in which case you recognize that this is the Boltzmann distribution for a system with two states, which differ in energy by $\beta_0 + x \cdot \beta$.

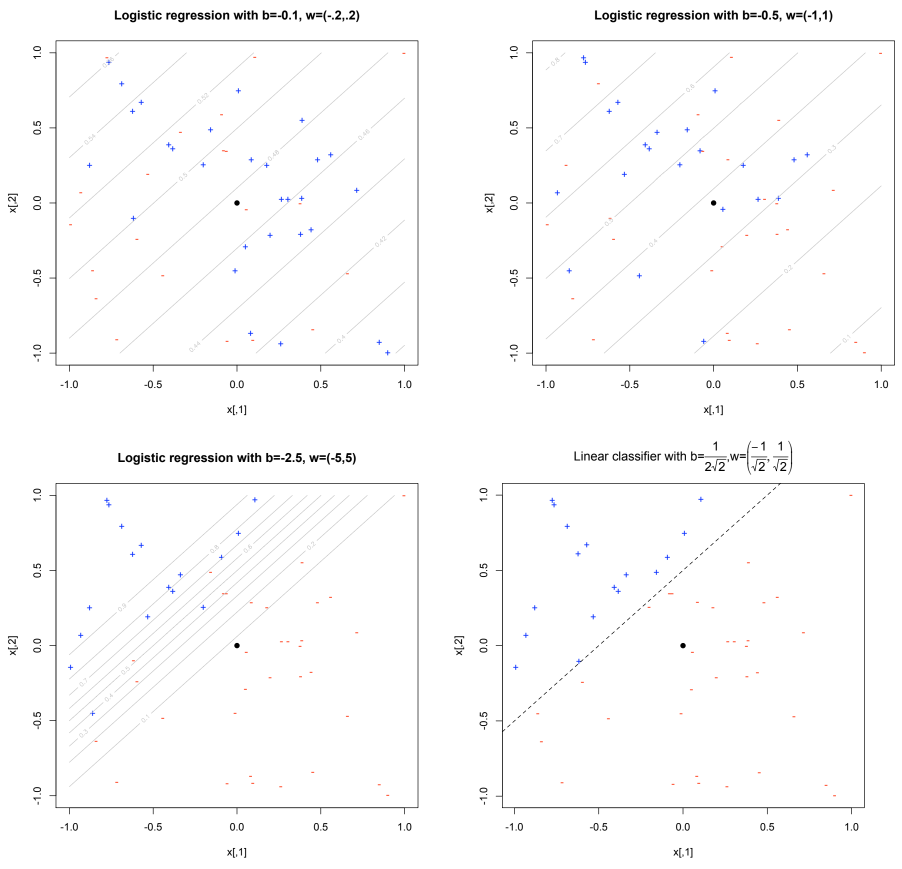

Figure 12.1: Effects of scaling logistic regression parameters. Values of x_1 and x_2 are the same in all plots ($\sim$ Unif$(-1, 1)$ for both coordinates), but labels were generated randomly from logistic regressions with $\beta_0 = -0.1$, $\beta = (-0.2, 0.2)$ (top left); from $\beta_0 = -0.5$, $\beta = (-1, 1)$ (top right); from $\beta_0 = -2.5$, $\beta = (-5, 5)$ (bottom left); and from a perfect linear classifier with the same boundary. The large black dot is the origin.

In neither case is the appropriateness of the model guaranteed by the gods, nature, mathematical necessity, etc. We begin by positing the model, to get something to work with, and we end (if we know what we're doing) by checking whether it really does match the data, or whether it has systematic flaws.

Logistic regression is one of the most commonly used tools for applied statistics and discrete data analysis. There are basically four reasons for this.

1. Tradition.

2. In addition to the heuristic approach above, the quantity $\log p/(1-p)$ plays an important role in the analysis of contingency tables (the "log odds"). Classification is a bit like having a contingency table with two columns (classes) and infinitely many rows (values of x). With a finite contingency table, we can estimate the log-odds for each row empirically, by just taking counts in the table. With infinitely many rows, we need some sort of interpolation scheme; logistic regression is linear interpolation for the log-odds.

3. It's closely related to "exponential family" distributions, where the probability of some vector v is proportional to $\exp \beta_0 + \sum_{j=1}^{m} f_j(v)\beta_j$. If one of the components of v is binary, and the functions f_j are all the identity function, then we get a logistic regression. Exponential families arise in many contexts in statistical theory (and in physics!), so there are lots of problems which can be turned into logistic regression.

4. It often works surprisingly well as a classifier. But, many simple techniques often work surprisingly well as classifiers, and this doesn't really testify to logistic regression getting the probabilities right.

12.2.1 Likelihood Function for Logistic Regression

Because logistic regression predicts probabilities, rather than just classes, we can fit it using likelihood. For each training data-point, we have a vector of features, x_i, and an observed class, y_i. The probability of that class was either p, if $y_i = 1$, or $1-p$, if $y_i = 0$. The likelihood is then

$$L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1-p(x_i)^{1-y_i} \tag{12.6}$$

(I could substitute in the actual equation for p, but things will be clearer in a moment if I don't.) The log-likelihood turns products into sums:

$$\begin{aligned}
\ell(\beta_0, \beta) &= \sum_{i=1}^{n} y_i \log p(x_i) + (1 - y_i) \log 1 - p(x_i) && (12.7)\\
&= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1 - p(x_i)} && (12.8)\\
&= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i(\beta_0 + x_i \cdot \beta) && (12.9)\\
&= \sum_{i=1}^{n} - \log 1 + e^{\beta_0 + x_i \cdot \beta} + \sum_{i=1}^{n} y_i(\beta_0 + x_i \cdot \beta) && (12.10)
\end{aligned}$$

where in the next-to-last step we finally use equation 12.4.

Typically, to find the maximum likelihood estimates we'd differentiate the log likelihood with respect to the parameters, set the derivatives equal to zero, and solve. To start that, take the derivative with respect to one component of β, say β_j.

$$\begin{aligned}
\frac{\partial \ell}{\partial \beta_j} &= -\sum_{i=1}^{n} \frac{1}{1 + e^{\beta_0 + x_i \cdot \beta}} e^{\beta_0 + x_i \cdot \beta} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} && (12.11)\\
&= \sum_{i=1}^{n} \left(y_i - p(x_i; \beta_0, \beta)\right) x_{ij} && (12.12)
\end{aligned}$$

We are not going to be able to set this to zero and solve exactly. (That's a transcendental equation, and there is no closed-form solution.) We can however approximately solve it numerically.

12.2.2 Logistic Regression with More Than Two Classes

If Y can take on more than two values, say k of them, we can still use logistic regression. Instead of having one set of parameters β_0, β, each class c in $0 : (k-1)$ will have its own offset $\beta_0^{(c)}$ and vector $\beta^{(c)}$, and the predicted conditional probabilities will be

$$\Pr\left(Y = c | \vec{X} = x\right) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}} \qquad (12.13)$$

You can check that when there are only two classes (say, 0 and 1), equation 12.13 reduces to equation 12.5, with $\beta_0 = \beta_0^{(1)} - \beta_0^{(0)}$ and $\beta = \beta^{(1)} - \beta^{(0)}$. In fact, no matter how many classes there are, we can always pick one of them, say $c = 0$, and fix its parameters at exactly zero, without any loss of generality[2].

[2]Since we can arbitrarily chose which class's parameters to "zero out" without affecting the predicted probabilities, strictly speaking the model in Eq. 12.13 is **unidentified**. That is, different parameter settings lead to *exactly* the same outcome, so we can't use the data to tell which one is right. The usual response here is to deal with this by a convention: we *decide* to zero out the parameters of the first class, and then estimate the contrasting parameters for the others.

Calculation of the likelihood now proceeds as before (only with more bookkeeping), and so does maximum likelihood estimation.

12.3 Newton's Method for Numerical Optimization

There are a huge number of methods for numerical optimization; we can't cover all bases, and there is no magical method which will always work better than anything else. However, there are some methods which work very well on an awful lot of the problems which keep coming up, and it's worth spending a moment to sketch how they work. One of the most ancient yet important of them is Newton's method (alias "Newton-Raphson").

Let's start with the simplest case of minimizing a function of one scalar variable, say $f(\beta)$. We want to find the location of the global minimum, β^*. We suppose that f is smooth, and that β^* is a regular interior minimum, meaning that the derivative at β^* is zero and the second derivative is positive. Near the minimum we could make a Taylor expansion:

$$f(\beta) \approx f(\beta^*) + \frac{1}{2}(\beta - \beta^*)^2 \left.\frac{d^2 f}{d\beta^2}\right|_{\beta=\beta^*} \tag{12.14}$$

(We can see here that the second derivative has to be positive to ensure that $f(\beta) > f(\beta^*)$.) In words, $f(\beta)$ is close to quadratic near the minimum.

Newton's method uses this fact, and minimizes a quadratic *approximation* to the function we are really interested in. (In other words, Newton's method is to replace the problem we want to solve, with a problem which we *can* solve.) *Guess* an initial point $\beta^{(0)}$. If this is close to the minimum, we can take a second order Taylor expansion around $\beta^{(0)}$ and it will still be accurate:

$$f(\beta) \approx f(\beta^{(0)}) + (\beta - \beta^{(0)}) \left.\frac{df}{dw}\right|_{\beta=\beta^{(0)}} + \frac{1}{2}\left(\beta - \beta^{(0)}\right)^2 \left.\frac{d^2 f}{dw^2}\right|_{\beta=\beta^{(0)}} \tag{12.15}$$

Now it's easy to minimize the right-hand side of equation 12.15. Let's abbreviate the derivatives, because they get tiresome to keep writing out: $\left.\frac{df}{dw}\right|_{\beta=\beta^{(0)}} = f'(\beta^{(0)})$, $\left.\frac{d^2f}{dw^2}\right|_{\beta=\beta^{(0)}} = f''(\beta^{(0)})$. We just take the derivative with respect to β, and set it equal to zero at a point we'll call $\beta^{(1)}$:

$$0 = f'(\beta^{(0)}) + \frac{1}{2} f''(\beta^{(0)}) 2(\beta^{(1)} - \beta^{(0)}) \tag{12.16}$$

$$\beta^{(1)} = \beta^{(0)} - \frac{f'(\beta^{(0)})}{f''(\beta^{(0)})} \tag{12.17}$$

The value $\beta^{(1)}$ should be a better guess at the minimum β^* than the initial one $\beta^{(0)}$ was. So if we use *it* to make a quadratic approximation to f, we'll get a better approximation, and so we can *iterate* this procedure, minimizing one approximation and then using that to get a new approximation:

$$\beta^{(n+1)} = \beta^{(n)} - \frac{f'(\beta^{(n)})}{f''(\beta^{(n)})} \tag{12.18}$$

Notice that the true minimum β^* is a **fixed point** of equation 12.18: if we happen to land on it, we'll stay there (since $f'(\beta^*) = 0$). We won't show it, but it can be proved that *if* $\beta^{(0)}$ is close enough to β^*, then $\beta^{(n)} \rightarrow \beta^*$, and that in general $|\beta^{(n)} - \beta^*| = O(n^{-2})$, a very rapid rate of convergence. (Doubling the number of iterations we use doesn't reduce the error by a factor of two, but by a factor of four.)

Let's put this together in an algorithm.

```
my.newton = function(f,f.prime,f.prime2,beta0,tolerance=1e-3,max.iter=50) {
  beta = beta0
  old.f = f(beta)
  iterations = 0
  made.changes = TRUE
  while(made.changes & (iterations < max.iter)) {
   iterations <- iterations +1
   made.changes <- FALSE
   new.beta = beta - f.prime(beta)/f.prime2(beta)
   new.f = f(new.beta)
   relative.change = abs(new.f - old.f)/old.f -1
   made.changes = (relative.changes > tolerance)
   beta = new.beta
   old.f = new.f
  }
  if (made.changes) {
    warning("Newton's method terminated before convergence")
  }
  return(list(minimum=beta,value=f(beta),deriv=f.prime(beta),
              deriv2=f.prime2(beta),iterations=iterations,
              converged=!made.changes))
}
```

The first three arguments here have to all be *functions*. The fourth argument is our initial guess for the minimum, $\beta^{(0)}$. The last arguments keep Newton's method from cycling forever: `tolerance` tells it to stop when the function stops changing very much (the relative difference between $f(\beta^{(n)})$ and $f(\beta^{(n+1)})$ is small), and `max.iter` tells it to never do more than a certain number of steps no matter what. The return value includes the estmated minimum, the value of the function there, and some diagnostics — the derivative should be very small, the second derivative should be positive, etc.

You may have noticed some potential problems — what if we land on a point where f'' is zero? What if $f(\beta^{(n+1)}) > f(\beta^{(n)})$? Etc. There are ways of handling these issues, and more, which are incorporated into real optimization algorithms from numerical analysis — such as the `optim` function in R; I strongly recommend you use that, or something like that, rather than trying to roll your own optimization code.[3]

12.3.1 Newton's Method in More than One Dimension

Suppose that the objective f is a function of multiple arguments, $f(\beta_1, \beta_2, \ldots \beta_p)$. Let's bundle the parameters into a single vector, w. Then the Newton update is

$$\beta^{(n+1)} = \beta^{(n)} - H^{-1}(\beta^{(n)}) \nabla f(\beta^{(n)}) \tag{12.19}$$

where ∇f is the **gradient** of f, its vector of partial derivatives $[\partial f / \partial \beta_1, \partial f / \partial \beta_2, \ldots \partial f / \partial \beta_p]$, and H is the **Hessian** of f, its matrix of second partial derivatives, $H_{ij} = \partial^2 f / \partial \beta_i \partial \beta_j$.

Calculating H and ∇f isn't usually very time-consuming, but taking the inverse of H is, unless it happens to be a diagonal matrix. This leads to various **quasi-Newton** methods, which either approximate H by a diagonal matrix, or take a proper inverse of H only rarely (maybe just once), and then try to update an estimate of $H^{-1}(\beta^{(n)})$ as $\beta^{(n)}$ changes.

12.3.2 Iteratively Re-Weighted Least Squares

This discussion of Newton's method is quite general, and therefore abstract. In the particular case of logistic regression, we can make everything look much more "statistical".

Logistic regression, after all, is a linear model for a transformation of the probability. Let's call this transformation g:

$$g(p) \equiv \log \frac{p}{1-p} \tag{12.20}$$

So the model is

$$g(p) = \beta_0 + x \cdot \beta \tag{12.21}$$

and $Y|X = x \sim Binom(1, g^{-1}(\beta_0 + x \cdot \beta))$. It seems that what we should want to do is take $g(y)$ and regress it linearly on x. Of course, the variance of Y, according to the model, is going to chance depending on x — it will be $(g^{-1}(\beta_0 + x \cdot \beta))(1 - g^{-1}(\beta_0 + x \cdot \beta))$ — so we really ought to do a weighted linear regression, with weights inversely proportional to that variance. Since writing $\beta_0 + x \cdot \beta$ is getting annoying, let's abbreviate it by μ (for "mean"), and let's abbreviate that variance as $V(\mu)$.

The problem is that y is either 0 or 1, so $g(y)$ is either $-\infty$ or $+\infty$. We will evade this by using Taylor expansion.

$$g(y) \approx g(\mu) + (y - \mu) g'(\mu) \equiv z \tag{12.22}$$

The right hand side, z will be our *effective* response variable. To regress it, we need its variance, which by propagation of error will be $(g'(\mu))^2 V(\mu)$.

[3]`optim` actually is a wrapper for several different optimization methods; `method=BFGS` selects a Newtonian method; BFGS is an acronym for the names of the algorithm's inventors.

Notice that both the weights *and* z depend on the parameters of our logistic regression, through μ. So having done this once, we should really use the new parameters to update z and the weights, and do it again. Eventually, we come to a fixed point, where the parameter estimates no longer change.

The treatment above is rather heuristic[4], but it turns out to be equivalent to using Newton's method, with the expected second derivative of the log likelihood, instead of its actual value.[5] Since, with a large number of observations, the observed second derivative should be close to the expected second derivative, this is only a small approximation.

12.4 Generalized Linear Models and Generalized Additive Models

Logistic regression is part of a broader family of **generalized linear models** (GLMs), where the conditional distribution of the response falls in some parametric family, and the parameters are set by the linear predictor. Ordinary, least-squares regression is the case where response is Gaussian, with mean equal to the linear predictor, and constant variance. Logistic regression is the case where the response is binomial, with n equal to the number of data-points with the given x (often but not always 1), and p is given by Equation 12.5. Changing the relationship between the parameters and the linear predictor is called changing the **link function**. For computational reasons, the link function is actually the function you apply to the mean response to get back the linear predictor, rather than the other way around — (12.4) rather than (12.5). There are thus other forms of binomial regression besides logistic regression.[6] There is also Poisson regression (appropriate when the data are counts without any upper limit), gamma regression, etc.; we will say more about these in Chapter 13.

In R, any standard GLM can be fit using the (base) `glm` function, whose syntax is very similar to that of `lm`. The major wrinkle is that, of course, you need to specify the family of probability distributions to use, by the `family` option — `family=binomial` defaults to logistic regression. (See `help(glm)` for the gory details on how to do, say, probit regression.) All of these are fit by the same sort of numerical likelihood maximization.

One caution about using maximum likelihood to fit logistic regression is that it can seem to work badly when the training data *can* be linearly separated. The reason is that, to make the likelihood large, $p(x_i)$ should be large when $y_i = 1$, and p should be small when $y_i = 0$. If β_0, β_0 is a set of parameters which perfectly classifies the training data, then $c\beta_0, c\beta$ is too, for any $c > 1$, but in a logistic regression the second

[4]That is, mathematically incorrect.

[5]This takes a reasonable amount of algebra to show, so we'll skip it. The key point however is the following. Take a single Bernoulli observation with success probability p. The log-likelihood is $Y \log p + (1-Y)\log 1 - p$. The first derivative with respect to p is $Y/p - (1-Y)/(1-p)$, and the second derivative is $-Y/p^2 - (1-Y)/(1-p)^2$. Taking expectations of the second derivative gives $-1/p - 1/(1-p) = -1/p(1-p)$. In other words, $V(p) = -1/\mathbf{E}\left[\ell''\right]$. Using weights inversely proportional to the variance thus turns out to be equivalent to dividing by the expected second derivative.

[6]My experience is that these tend to give similar error rates as classifiers, but have rather different guesses about the underlying probabilities.

set of parameters will have more extreme probabilities, and so a higher likelihood. For linearly separable data, then, there is no parameter vector which *maximizes* the likelihood, since ℓ can always be increased by making the vector larger but keeping it pointed in the same direction.

You should, of course, be so lucky as to have this problem.

12.4.1 Generalized Additive Models

A natural step beyond generalized linear models is **generalized additive models** (GAMs), where instead of making the transformed mean response a *linear* function of the inputs, we make it an *additive* function of the inputs. This means combining a function for fitting additive models with likelihood maximization. The R function here is `gam`, from the CRAN package of the same name. (Alternately, use the function `gam` in the package `mgcv`, which is part of the default R installation.) We will look at how this works in some detail in Chapter 13.

GAMs can be used to check GLMs in much the same way that smoothers can be used to check parametric regressions: fit a GAM and a GLM to the same data, then simulate from the GLM, and re-fit both models to the simulated data. Repeated many times, this gives a distribution for how much better the GAM will seem to fit than the GLM does, *even when the GLM is true*. You can then read a p-value off of this distribution.

12.4.2 An Example (Including Model Checking)

Here's a worked R example, using the data from the upper right panel of Figure 12.1. The 50×2 matrix `x` holds the input variables (the coordinates are independently and uniformly distributed on $[-1, 1]$), and `y.1` the corresponding class labels, themselves generated from a logistic regression with $\beta_0 = -0.5$, $\beta = (-1, 1)$.

```
> logr = glm(y.1 ~ x[,1] + x[,2], family=binomial)
> logr

Call:  glm(formula = y.1 ~ x[, 1] + x[, 2], family = binomial)

Coefficients:
(Intercept)       x[, 1]       x[, 2]
     -0.410       -1.050        1.366

Degrees of Freedom: 49 Total (i.e. Null);  47 Residual
Null Deviance:     68.59
Residual Deviance: 58.81  AIC: 64.81
> sum(ifelse(logr$fitted.values<0.5,0,1) != y.1)/length(y.1)
[1] 0.32
```

The **deviance** of a model fitted by maximum likelihood is (twice) the difference between its log likelihood and the maximum log likelihood for a **saturated** model, i.e., a model with one parameter per observation. Hopefully, the saturated model can give a perfect fit.[7] Here the saturated model would assign probability 1 to the observed outcomes[8], and the logarithm of 1 is zero, so $D = 2\ell(\widehat{\beta_0}, \widehat{\beta})$. The null deviance is what's achievable by using just a constant bias b and setting $w = 0$. The fitted model definitely improves on that.[9]

The fitted values of the logistic regression are the class probabilities; this shows that the error rate of the logistic regression, if you force it to predict actual classes, is 32%. This sounds bad, but notice from the contour lines in the figure that lots of the probabilities are near 0.5, meaning that the classes are just genuinely hard to predict.

To see how well the logistic regression assumption holds up, let's compare this to a GAM.[10]

```
> library(gam)
> gam.1 = gam(y.1~lo(x[,1])+lo(x[,2]),family="binomial")
> gam.1
Call:
gam(formula = y.1 ~ lo(x[, 1]) + lo(x[, 2]), family = "binomial")

Degrees of Freedom: 49 total; 41.39957 Residual
Residual Deviance: 49.17522
```

This fits a GAM to the same data, using lowess smoothing of both input variables. Notice that the residual deviance is lower. That is, the GAM fits better. We expect this; the question is whether the difference is significant, or within the range of what we should expect when logistic regression is valid. To test this, we need to simulate from the logistic regression model.

```
simulate.from.logr = function(x, coefs) {
  require(faraway) # For accessible logit and inverse-logit functions
  n = nrow(x)
  linear.part = coefs[1] + x %*% coefs[-1]
  probs = ilogit(linear.part) # Inverse logit
  y = rbinom(n,size=1,prob=probs)
  return(y)
}
```

Now we simulate from our fitted model, and re-fit both the logistic regression and the GAM.

[7]The factor of two is so that the deviance will have a χ^2 distribution. Specifically, if the model with p parameters is right, the deviance will have a χ^2 distribution with $n - p$ degrees of freedom.

[8]This is not possible when there are multiple observations with the same input features, but different classes.

[9]AIC is of course the Akaike information criterion, $-2\ell + 2q$, with q being the number of parameters (here, $q = 3$). AIC has some truly devoted adherents, especially among non-statisticians, but I have been deliberately ignoring it and will continue to do so. Basically, to the extent AIC succeeds, it works as fast, large-sample approximation to doing leave-one-out cross-validation. Claeskens and Hjort (2008) is a thorough, modern treatment of AIC and related model-selection criteria from a statistical viewpoint.

[10]Previous examples of using GAMs have mostly used the `mgcv` package and spline smoothing. There is no particular reason to switch to the `gam` library and lowess smoothing here, but there's also no real reason not to.

```
delta.deviance.sim = function (x,logistic.model) {
  y.new = simulate.from.logr(x,logistic.model$coefficients)
  GLM.dev = glm(y.new ~ x[,1] + x[,2], family="binomial")$deviance
  GAM.dev = gam(y.new ~ lo(x[,1]) + lo(x[,2]), family="binomial")$deviance
  return(GLM.dev - GAM.dev)
}
```

Notice that in this simulation we are not generating new $\vec{X}$ values. The logistic regression and the GAM are both models for the response *conditional* on the inputs, and are agnostic about how the inputs are distributed, or even whether it's meaningful to talk about their distribution.

Finally, we repeat the simulation a bunch of times, and see where the observed difference in deviances falls in the sampling distribution.

```
> delta.dev = replicate(1000,delta.deviance.sim(x,logr))
> delta.dev.observed = logr$deviance - gam.1$deviance # 9.64
> sum(delta.dev.observed > delta.dev)/1000
[1] 0.685
```

In other words, the amount by which a GAM fits the data better than logistic regression is pretty near the middle of the null distribution. Since the example data really *did* come from a logistic regression, this is a relief.

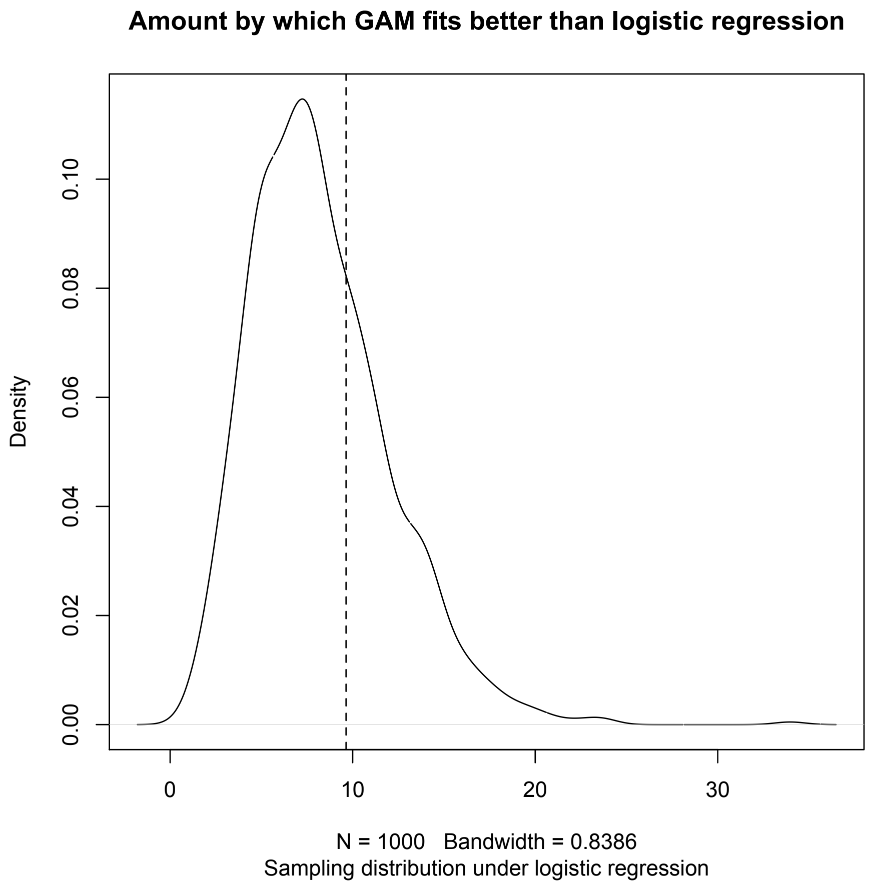

Figure 12.2: Sampling distribution for the difference in deviance between a GAM and a logistic regression, on data generated from a logistic regression. The observed difference in deviances is shown by the dashed horizontal line.

12.5 Exercises

To think through, not to hand in.

1. A multiclass logistic regression, as in Eq. 12.13, has parameters $\beta_0^{(c)}$ and $\beta^{(c)}$ for each class c. Show that we can always get the same predicted probabilities by setting $\beta_0^{(c)} = 0$, $\beta^{(c)} = 0$ for any one class c, and adjusting the parameters for the other classes appropriately.

2. Find the first and second derivatives of the log-likelihood for logistic regression with one predictor variable. Explicitly write out the formula for doing one step of Newton's method. Explain how this relates to re-weighted least squares.