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, guessing "yes" or "no" is pretty crude — especially if there is no perfect rule. (Why should there be a perfect rule?) 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 should be. 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 will see, in Chapter 16, general approaches to estimating conditional probabilities non-parametrically, which can use the kernels for discrete variables from Chapter 4. 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 basic statistics, 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, if we *model* the probabilities, 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 chapter 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 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

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

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 $p < 0.5$ (Exercise 1). 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 thermodynamics or physical chemistry, in which case you recognize that this is the Boltzmann distribution for a system with two states, which differ in energy by $\beta_0 + x \cdot \beta$.

```
x <- matrix(runif(n=50*2,min=-1,max=1),ncol=2)
par(mfrow=c(2,2))
plot.logistic.sim(x,beta.0=-0.1,beta=c(-0.2,0.2))
y.1 <- plot.logistic.sim(x,beta.0=-0.5,beta=c(-1,1))
plot.logistic.sim(x,beta.0=-2.5,beta=c(-5,5))
plot.logistic.sim(x,beta.0=-2.5e2,beta=c(-5e2,5e2))
```

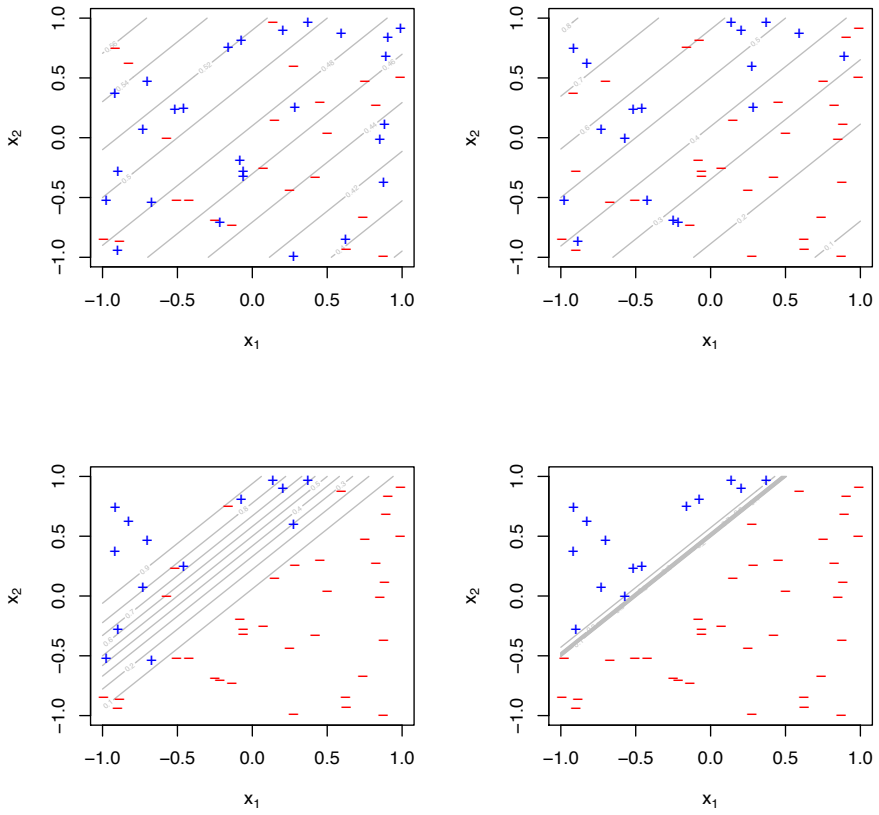

FIGURE 12.1: *Effects of scaling logistic regression parameters. Values of x_1 and x_2 are the same in all plots ($\sim$ Unif$(-1,1)$ for both coordinates), but labels were generated randomly from logistic regressions with $\beta_0 = -0.1$, $\beta = (-0.2, 0.2)$ (top left); from $\beta_0 = -0.5$, $\beta = (-1,1)$ (top right); from $\beta_0 = -2.5$, $\beta = (-5,5)$ (bottom left); and from $\beta_0 = 2.5 \times 10^2$, $\beta = (-5 \times 10^2, 5 \times 10^2)$. Notice how as the parameters get increased in constant ratio to each other, we approach a deterministic relation between Y and x, with a linear boundary between the classes. (We save one set of the random binary responses for use later, as the imaginatively-named* `y.1`.*)*

```
sim.logistic <- function(x, beta.0,beta,bind=FALSE) {
  require(faraway) # For accessible logit and inverse-logit functions
  linear.parts <- beta.0+(x%*%beta)
  y <- rbinom(nrow(x),size=1,prob=ilogit(linear.parts))
  if (bind) { return(cbind(x,y)) } else { return(y) }
}

plot.logistic.sim <- function(x, beta.0, beta, n.grid=50,
                              labcex=0.3, col="grey", ...) {
  grid.seq <- seq(from=-1,to=1,length.out=n.grid)
  plot.grid <- as.matrix(expand.grid(grid.seq,grid.seq))
  require(faraway)
  p <- matrix(ilogit(beta.0 + (plot.grid %*% beta )),nrow=n.grid)
  contour(x=grid.seq,y=grid.seq,z=p, xlab=expression(x[1]),
          ylab=expression(x[2]),main="",labcex=labcex,col=col)
  y <- sim.logistic(x,beta.0,beta,bind=FALSE)
  points(x[,1],x[,2],pch=ifelse(y==1,"+","-"),col=ifelse(y==1,"blue","red"))
  invisible(y)
}
```

CODE EXAMPLE 28: *Code to simulate binary responses from a logistic regression model, and to plot a 2D logistic regression's probability contours and simulated binary values. (How would you modify this to take the responses from a data frame?*

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:

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

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.

12.3 Numerical Optimization of the Likelihood

While our likelihood isn't nice enough that we have an explicit expression for the maximum (the way we do in OLS or WLS), it is a pretty well-behaved function,

and one which is amenable to lots of the usual numerical methods for optimization (see Appendix F). In particular, like most log-likelihood functions, it's suitable for an application of Newton's method. Briefly (see Appendix F.1.2 for details), Newton's method starts with an initial guess about the optimal parameters, and then calculates the gradient of the log-likelihood with respect to those parameters. It then adds an amount proportional to the gradient to the parameters, moving up the surface of the log-likelihood function. The size of the step in the gradient direction is dictated by the second derivatives — it takes bigger steps when the second derivatives are small (so the gradient is a good guide to what the function looks like), and small steps when the curvature is large.

12.3.1 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 like a good old fashioned statistics problem.

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

$$g(p) \equiv \log \frac{p}{1-p} \tag{12.13}$$

So the model is

$$g(p) = \beta_0 + x \cdot \beta \tag{12.14}$$

and $Y|X = x \sim \text{Binom}(1, g^{-1}(\beta_0 + x \cdot \beta))$. 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 change 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 $g^{-1}(\beta_0 + x \cdot \beta)$ is getting annoying, let's abbreviate it by $p(x)$ or just p, and let's abbreviate that variance as $V(p)$.

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.

$$g(y) \approx g(p) + (y - p)g'(p) \equiv z \tag{12.15}$$

The right hand side, z will be our *effective* response variable, which we will regress on x. To see why this should give us the right coefficients, substitute for $g(p)$ in the definition of z,

$$z = \beta_0 + x \cdot \beta + (y - p)g'(p) \tag{12.16}$$

and notice that, if we've got the coefficients right, $\mathbf{E}[Y|X = x] = p$, so $(y - p)$ should be mean-zero noise. In other words, when we have the right coefficients, z *is* a linear function of x plus mean-zero noise. (This is our excuse for throwing away the rest of the Taylor expansion, even though we know the discarded terms are infinitely large!) That noise doesn't have constant variance, but we can work it out,

$$\text{Var}[Z|X = x] = \text{Var}\left[(Y - p)g'(p)|X = x\right] = (g'(p))^2 V(p)\,, \tag{12.17}$$

and so use that variance in weighted least squares to recover β.

Notice that z and the weights both involve the parameters of our logistic regression, through $p(x)$. 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. This loop — start with a guess about the parameters, use it to calculate the z_i and their weights, regress on the x_i to get new parameters, and repeat — is known as **iterative reweighted least squares** (IRLS or IRWLS), **iterative weighted least squares** (IWLS), etc.

The treatment above is rather heuristic[2], but it turns out to be equivalent to using Newton's method, only with the expected second derivative of the log likelihood, instead of its actual value. This takes a reasonable amount of algebra to show, so we'll skip it (but see Exercise 3)[3]. 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 and 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 (usually 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.[4] 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.

[2] That is, mathematically incorrect.

[3] The two key points are as follows. First, the gradient of the log-likelihood turns out to be the sum of the $z_i x_i$. (Cf. Eq. 12.12.) Second, 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. But gradient divided by second derivative is the increment we use in Newton's method, QED.

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

Perfect Classification 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 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. This is actually done in R with the same `gam` function we used for additive models (hence the name). We will look at how this works in some detail in Chapter 13. For now, the basic idea is that the iteratively re-weighted least squares procedure of §12.3.1 doesn't really require the model for the log odds to be linear. We get a GAM when we fit an additive model to the z_i; we could even fit an arbitrary non-parametric model, like a kernel regression, though that's not very common.

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. This is illustrated in §12.6 below.

12.5 Model Checking

The validity of the logistic regression model is no more a fact of mathematics or nature than is the validity of the linear regression model. Both are sometimes convenient assumptions, but neither is guaranteed to be correct, nor even some sort of generally-correct default. In either case, if we want to use the model, the proper scientific (and statistical) procedure is to *check* the validity of the modeling assumptions.

12.5.1 Residuals

In your linear models course, you learned a lot of checks based on the residuals of the model (see Chapter 2). Many of these ideas translates to logistic regression, but we need to re-define residuals. Sometimes people work with the "response" residuals,

$$y_i - p(x_i) \tag{12.18}$$

which should have mean zero (why?), but are heteroskedastic even when the model is true (why?). Others work with standardized or **Pearson** residuals,

$$\frac{y_i - p(x_i)}{V(p(x_i))} \tag{12.19}$$

and there are yet other notions of residuals for logistic models. Still, both the response and the Pearson residuals should be unpredictable from the covariates, and the latter should have constant variance.

12.5.2 Non-parametric Alternatives

Chapter 10 discussed how non-parametric regression models can be used to check whether parametric regressions are well-specified. The same ideas apply to logistic regressions, with the minor modification that in place of the difference in MSEs, one should use the difference in log-likelihoods, or (what comes to the same thing, up to a factor of 2) the difference in deviances. The use of generalized additive models (§12.4.1) as the alternative model class is illustrated in §12.6 below.

12.5.3 Calibration

Because logistic regression predicts actual *probabilities*, we can check its predictions in a more stringent way than an ordinary regression, which just tells us the mean value of Y, but is otherwise silent about its distribution. If we've got a model which tells us that the probability of rain on a certain class of days is 50%, it had better rain on half of those days, or there model is just *wrong* about the probability of rain. More generally, we'll say that the model is **calibrated** (or **well-calibrated**) when

$$\Pr(Y = 1 | \hat{p}(X) = p) = p \tag{12.20}$$

That is, the actual probabilities should match the predicted probabilities. If we have a large sample, by the law of large numbers, observed relative frequencies will converge on true probabilities. Thus, the observed relative frequencies should be close to the predicted probabilities, or else the model is making systematic mistakes.

In practice, each case may have its own unique predicted probability p, so it might not be possible to accumulate many cases with the same p and check the relative frequency among those cases. When that happens, one option is to look at all the cases where the predicted probability is in some small range $[p, p + \epsilon)$; the observed relative frequency had them better be in that range too. §12.7 below illustrates some of the relevant calculations.

A second option is to use what is called a **proper scoring rule**, which is a function of the outcome variables and the predicted probabilities that attains its minimum when, and only when, the predicted are calibrated. For binary outcomes, one proper scoring rule (historically the oldest) is the **Brier score**,

$$n^{-1} \sum_{i=1}^{n} (y_i - p_i)^2 \tag{12.21}$$

Another however is simply the (normalized) negative log-likelihood,

$$-n^{-1}\sum_{i=1}^{n} y_i \log p_i + (1-y_i)\log 1 - p_i \tag{12.22}$$

Of course, proper scoring rules are better evaluated out-of-sample, or, failing that, through cross-validation, than in-sample. Even an in-sample evaluation is better than nothing, however, which is too often what happens.

12.6 A Toy Example

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 $\beta_0 = -0.5$, $\beta = (-1, 1)$.

```
df <- data.frame(y=y.1,x1=x[,1],x2=x[,2])
(logr <- glm(y ~ x1 + x2, data=df, family="binomial"))
##
## Call:  glm(formula = y ~ x1 + x2, family = "binomial", data = df)
##
## Coefficients:
## (Intercept)           x1           x2
##      -0.677       -1.579        1.556
##
## Degrees of Freedom: 49 Total (i.e. Null);  47 Residual
## Null Deviance:     65.3
## Residual Deviance: 53.3  AIC: 59.3
```

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.[5] Here the saturated model would assign probability 1 to the observed outcomes[6], 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 β_0 and setting the rest of β to 0. The fitted model definitely improves on that.[7]

[5] 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. See Appendix G for the connection between log likelihood ratios and χ^2 distributions.

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

[7] AIC is of course the Akaike information criterion, $-2\ell + 2p$, with p being the number of parameters (here, $p = 3$). (Some people divide this through by n.) 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; see especially §2.9 for the connection between AIC and leave-one-out. [[TODO: AIC appendix]]

If we're interested in inferential statistics on the estimated model, we can see those with `summary`, as with `lm`:

```
summary(logr,digits=2,signif.stars=FALSE)
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial", data = df)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.778  -0.838  -0.513   0.992   1.817
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.677      0.343   -1.97    0.049 *
## x1            -1.579      0.613   -2.57    0.010 *
## x2             1.556      0.635    2.45    0.014 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 65.342  on 49  degrees of freedom
## Residual deviance: 53.265  on 47  degrees of freedom
## AIC: 59.26
##
## Number of Fisher Scoring iterations: 4
```

The fitted values of the logistic regression are the class probabilities; this next line gives us the (in-sample) mis-classification rate.

```
mean(ifelse(fitted(logr)<0.5,0,1) != df$y)
## [1] 0.26
```

An error rate of 26 % may sound bad, but notice from the contour lines in Figure 12.1 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. We'll use the same package for estimating the GAM, `mgcv`, that we used to fit the additive models in Chapter 9.

```
library(mgcv)
```

```
## Loading required package:  nlme
## This is mgcv 1.7-28.  For overview type 'help("mgcv-package")'.
```

```
(gam.1 <- gam(y~s(x1)+s(x2),data=df,family="binomial"))
##
## Family: binomial
```

```
# Simulate a fitted logistic regression and return a new data frame
# Inputs: data frame (df), fitted model (mdl)
# Outputs: new data frame
# Presumes: df contains columns with names for the covariates of mdl
simulate.from.logr <- function(df, mdl) {
  probs <- predict(mdl,newdata=df,type="response")
  df$y <- rbinom(n=nrow(x),size=1,prob=probs)
  return(df)
}
```

CODE EXAMPLE 29: *Code for simulating from an* estimated *logistic regression model. By default (*`type="link"`*),* `predict` *for logistic regressions returns predictions for the log odds; changing the* `type` *to* `"response"` *returns a probability.*

```
## Link function: logit
##
## Formula:
## y ~ s(x1) + s(x2)
##
## Estimated degrees of freedom:
## 1 1  total = 3
##
## UBRE score: 0.1853
```

This fits a GAM to the same data, using spline smoothing of both input variables. (Figure 12.2 shows the partial response functions.) The (in-sample) deviance is

```
signif(gam.1$deviance,3)
## [1] 53.3
```

which is lower than the logistic regression, so the GAM gives the data higher likelihood. 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.

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

```
# Simulate from an estimated logistic model, and refit both the logistic
  # regression and a generalized additive model
# Hard-codes the formula; better code would be more flexible
# Inputs: data frame with covariates (df), fitted logistic model (mdl)
# Output: difference in deviances
# Presumes: df has columns names x.1 and x.2.
delta.deviance.sim <- function (df,mdl) {
  sim.df <- simulate.from.logr(df,mdl)
  GLM.dev <- glm(y~x1+x2,data=sim.df,family="binomial")$deviance
  GAM.dev <- gam(y~s(x1)+s(x2),data=sim.df,family="binomial")$deviance
```

```
plot(gam.1,residuals=TRUE,pages=0)
```

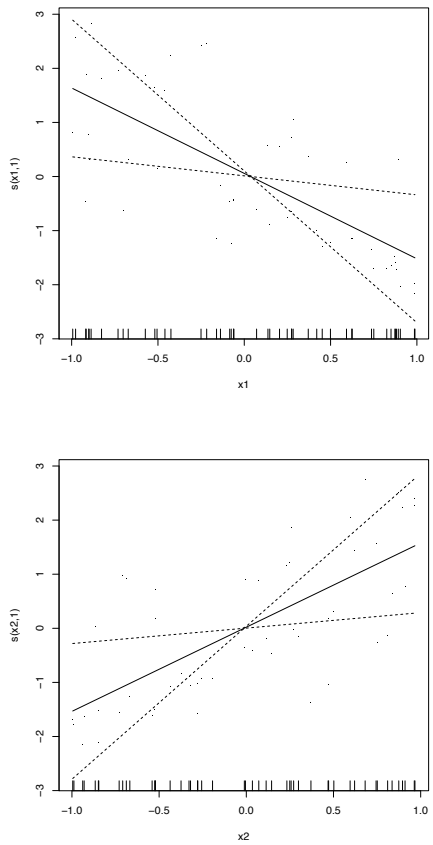

FIGURE 12.2: *Partial response functions estimated when we fit a GAM to the data simulated from a logistic regression. Notice that the vertical axes are on the logit scale.*

```
  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.observed <- logr$deviance - gam.1$deviance)
## [1] 7.723e-05
delta.dev <- replicate(100,delta.deviance.sim(df,logr))
mean(delta.dev.observed > delta.dev)
## [1] 0.26
```

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.

```
hist(delta.dev, main="",
       xlab="Amount by which GAM fits better than logistic regression")
abline(v=delta.dev.observed,col="grey",lwd=4)
```

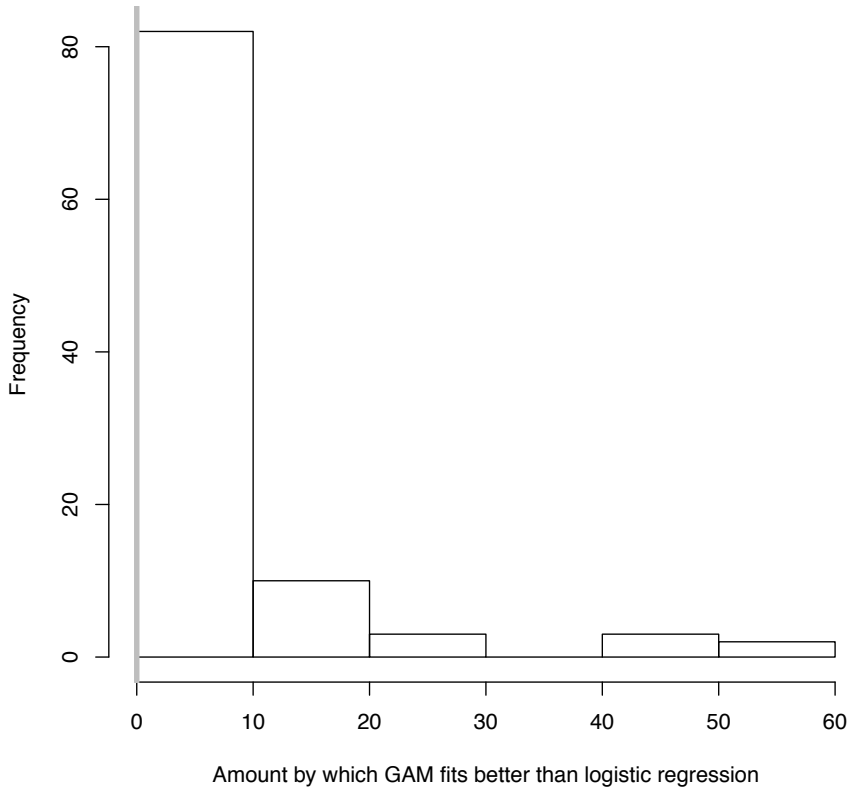

FIGURE 12.3: *Sampling distribution for the difference in deviance between a GAM and a logistic regression, on data generated from a logistic regression. The observed difference in deviances is shown by the grey vertical line.*

12.7 Weather Forecasting in Snoqualmie Falls

For our worked data example, we are going to build a simple weather forecaster. Our data consist of daily records, from the beginning of 1948 to the end of 1983, of precipitation at Snoqualmie Falls, Washington (Figure 12.4)[8]. Each row of the data file is a different year; each column records, for that day of the year, the day's precipitation (rain or snow), in units of $\frac{1}{100}$ inch. Because of leap-days, there are 366 columns, with the last column having an NA value for three out of four years.

```
# Read in the whole data set as one big vector, skipping the first line of
# the file (header information)
snoqualmie <- scan("http://www.stat.washington.edu/peter/book.data/set1",skip=1)
# Create a two-column data frame, today's precipitation vs. tomorrow's
snoq <- data.frame(tomorrow=c(tail(snoqualmie,-1),NA),
                   today=snoqualmie)
# Make the data more comprehensible by adding years and days within each year
# First, what years are we talking about?
years <- 1948:1983
# How many days to each year?
days.per.year <- rep(c(366,365,365,365),length.out=length(years))
# Add a "year" column
snoq$year <- rep(years, times=days.per.year)
# Add a day-within-the-year column
snoq$day <- rep(c(1:366,1:365,1:365,1:365),times=length(years)/4)
# Trim the last row to get rid of the NA
snoq <- snoq[-nrow(snoq),]
```

What we want to do is predict tomorrow's weather from today's. This would be of interest if we lived in Snoqualmie Falls, or if we operated one of the local hydroelectric power plants, or the tourist attraction of the Falls themselves. Examining the distribution of the data (Figures 12.5 and 12.6) shows that there is a big spike in the distribution at zero precipitation, and that days of no precipitation can follow days of any amount of precipitation but seem to be less common after heavy precipitation.

These facts suggest that "no precipitation" is a special sort of event which would be worth predicting in its own right (as opposed to just being when the precipitation happens to be zero), so we will attempt to do so with logistic regression. Specifically, the input variable X_i will be the amount of precipitation on the i^{th} day, and the response Y_i will be the indicator variable for whether there was any precipitation on day $i+1$ — that is, $Y_i = 1$ if $X_{i+1} > 0$, an $Y_i = 0$ if $X_{i+1} = 0$. We expect from Figure 12.6, as well as common experience, that the coefficient on X should be positive.[9]

The estimation is straightforward:

[8] I learned of this data set from Guttorp (1995); the data file is available from `http://www.stat.washington.edu/peter/stoch.mod.data.html`. Prof. Guttorp formatted it so that each year was a different row, which is rather inconvenient for our purposes; see `http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV/snoqualmie.R` for the commands used to reshape it.

[9] This does not attempt to model *how much* precipitation there will be tomorrow, if there is any. We could make that a separate model, if we can get this part right.

FIGURE 12.4: *Snoqualmie Falls, Washington, on a low-precipitation day. Photo by Jeannine Hall Gailey, from* `http://myblog.webbish6.com/2011/07/17-years-and-hoping-for-another-17.html`. *[[TODO: Get permission for photo use!]]*

```
hist(snoqualmie,n=50,probability=TRUE,xlab="Precipitation (1/100 inch)")
rug(snoqualmie,col="grey")
```

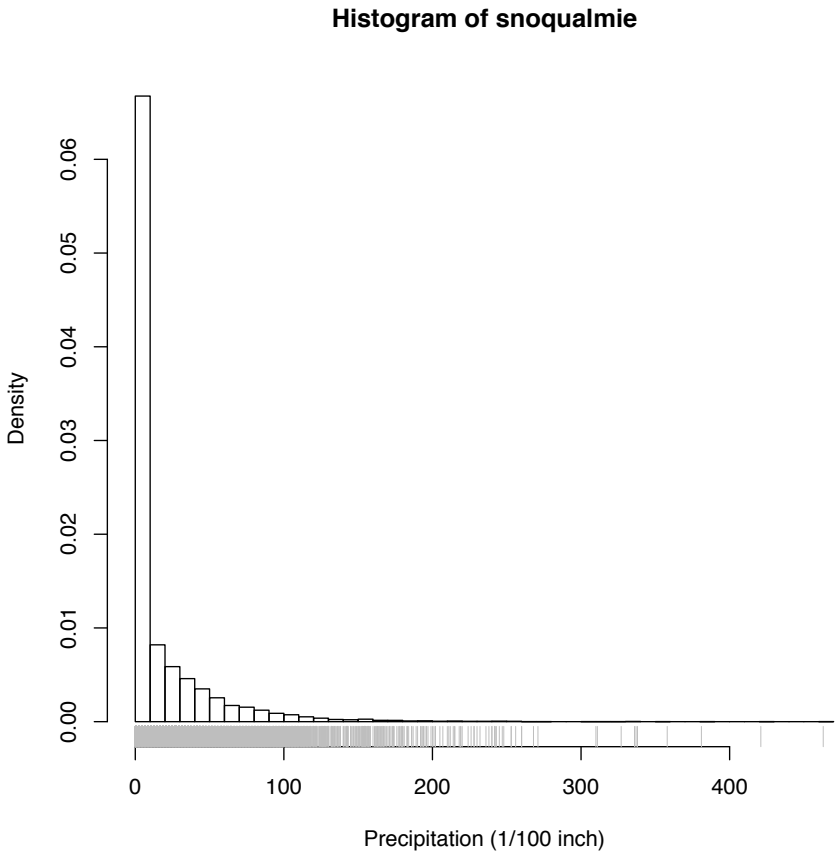

FIGURE 12.5: *Histogram of the amount of daily precipitation at Snoqualmie Falls*

```
plot(tomorrow~today,data=snoq,
     xlab="Precipitation today (1/100 inch)",
     ylab="Precipitation tomorrow (1/100 inch)",cex=0.1)
rug(snoq$today,side=1,col="grey")
rug(snoq$tomorrow,side=2,col="grey")
```

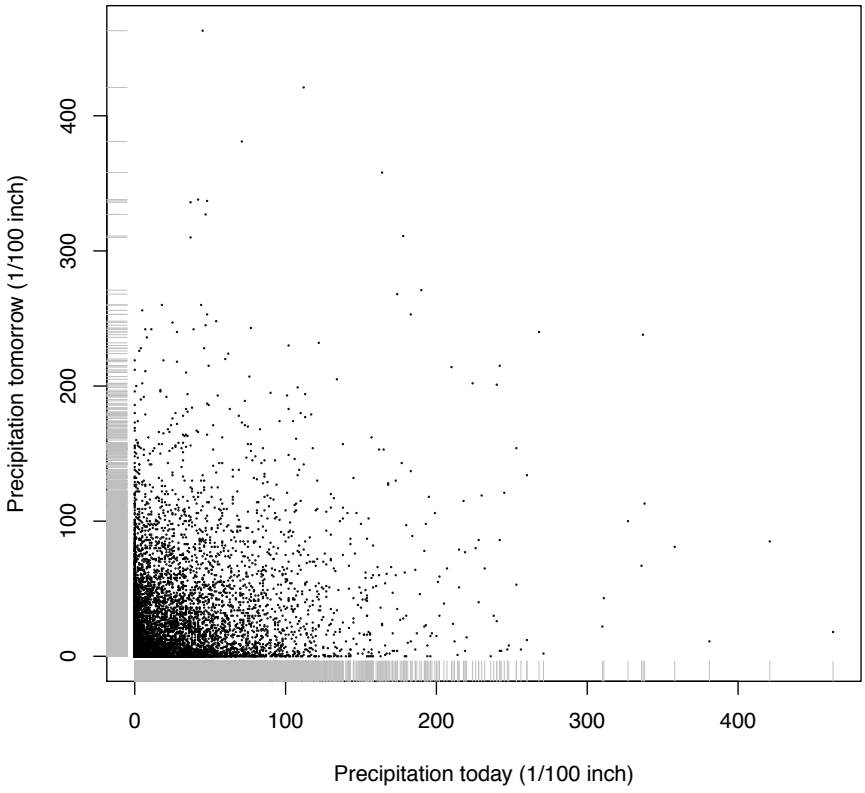

FIGURE 12.6: *Scatterplot showing relationship between amount of precipitation on successive days. Notice that days of no precipitation can follow days of any amount of precipitation, but seem to be more common when there is little or no precipitation to start with.*

```
snoq.logistic <- glm((tomorrow > 0) ~ today, data=snoq, family=binomial)
```

To see what came from the fitting, run `summary`:

```
print(summary(snoq.logistic),digits=3,signif.stars=FALSE)
##
## Call:
## glm(formula = (tomorrow > 0) ~ today, family = binomial, data = snoq)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.525  -0.999   0.167   1.170   1.367
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.43520    0.02163   -20.1   <2e-16
## today        0.04523    0.00131    34.6   <2e-16
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 18191  on 13147  degrees of freedom
## Residual deviance: 15896  on 13146  degrees of freedom
## AIC: 15900
##
## Number of Fisher Scoring iterations: 5
```

The coefficient on the amount of precipitation today is indeed positive, and (if we can trust R's assumptions) highly significant. There is also an intercept term, which is slight positive, but not very significant. We can see what the intercept term means by considering what happens when on days of no precipitation. The linear predictor is then just the intercept, -0.435, and the predicted probability of precipitation is 0.393. That is, even when there is no precipitation today, it's almost as likely as not that there will be some precipitation tomorrow.[10]

We can get a more global view of what the model is doing by plotting the data and the predictions (Figure 12.7). This shows a steady increase in the probability of precipitation tomorrow as the precipitation today increases, though with the leveling off characteristic of logistic regression. The (approximate) 95% confidence limits for the predicted probability are (on close inspection) asymmetric.

How well does this work? We can get a first sense of this by comparing it to a simple nonparametric smoothing of the data. Remembering that when Y is binary, $\Pr Y = 1|X = x = \mathbf{E}[Y|X = x]$, we can use a smoothing spline to estimate $\mathbf{E}[Y|X = x]$ (Figure 12.8). This would not be so great as a model — it ignores the fact that the response is a binary event and we're trying to estimate a probability, the fact that the variance of Y therefore depends on its mean, etc. — but it's at least indicative.

[10] For western Washington State, this is plausible — but see below.

```
plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)", ylab="Positive precip
rug(snoq$today,side=1,col="grey")
data.plot <- data.frame(today=(0:500))
pred.bands <- function(mdl,data,col="black",mult=1.96) {
    preds <- predict(mdl,newdata=data,se.fit=TRUE)
    lines(data[,1],ilogit(preds$fit),col=col)
    lines(data[,1],ilogit(preds$fit+mult*preds$se.fit),col=col,lty="dashed")
    lines(data[,1],ilogit(preds$fit-mult*preds$se.fit),col=col,lty="dashed")
}
pred.bands(snoq.logistic,data.plot)
```

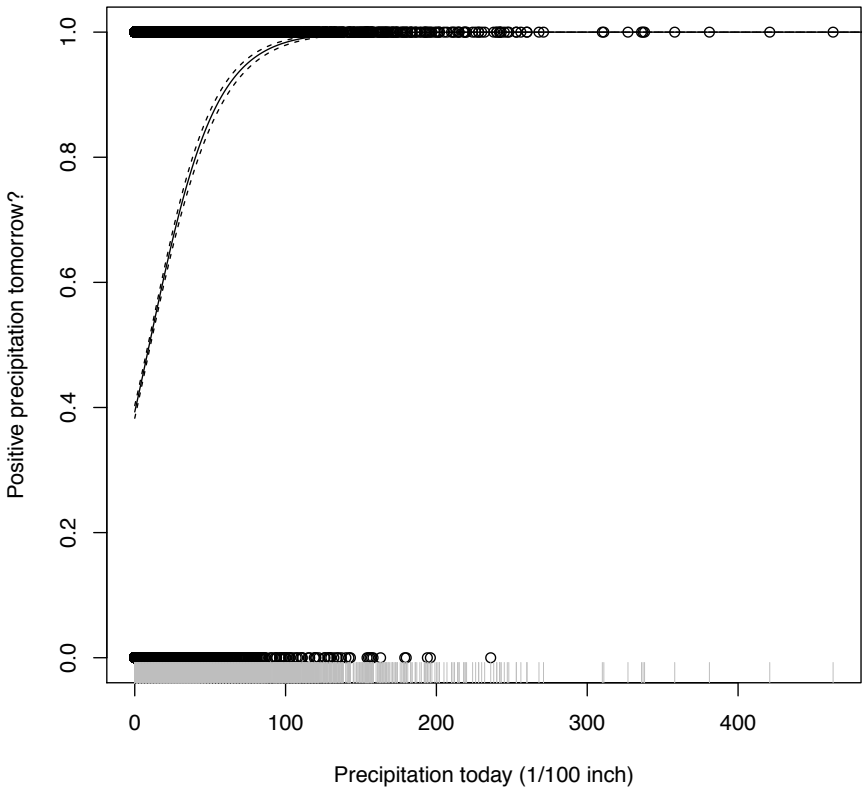

FIGURE 12.7: *Data (dots), plus predicted probabilities (solid line) and approximate 95% confidence intervals from the logistic regression model (dashed lines). Note that calculating standard errors for predictions on the logit scale, and then transforming, is better practice than getting standard errors directly on the probability scale.*

```
plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)", ylab="Positive precipitation tomo
rug(snoq$today,side=1,col="grey")
data.plot <- data.frame(today=(0:500))
pred.bands(snoq.logistic,data.plot)
snoq.spline <- smooth.spline(x=snoq$today,y=(snoq$tomorrow>0))
lines(snoq.spline,col="red")
```

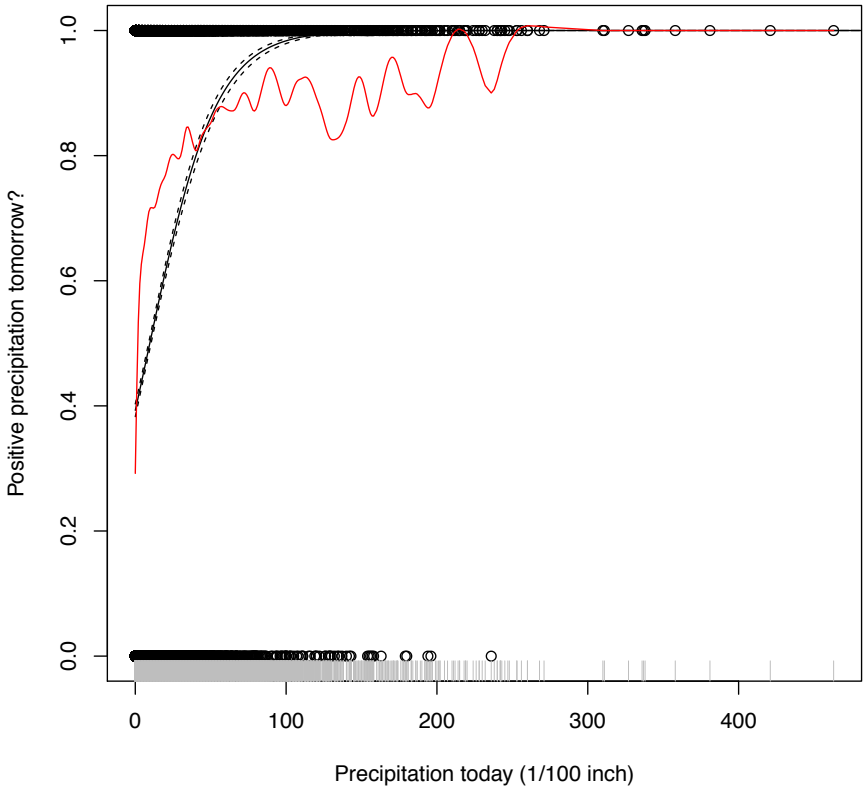

FIGURE 12.8: *As Figure 12.7, plus a smoothing spline (red).*

The result starts out notably above the logistic regression, then levels out and climbs much more slowly. It also has a bunch of dubious-looking wiggles, despite the cross-validation.

We can try to do better by fitting a generalized additive model. In this case, with only one predictor variable, this means using non-parametric smoothing to estimate the log odds — we're still using the logistic transformation, but only requiring that the log odds change smoothly with X, not that they be linear in X. The result (Figure 12.9) is initially similar to the spline, but has some more exaggerated undulations, and has confidence intervals. At the largest values of X, the latter span nearly the whole range from 0 to 1, which is not unreasonable considering the sheer lack of data there.

Visually, the logistic regression curve is hardly ever within the confidence limits of the non-parametric predictor. What can we say about the difference between the two models more quantiatively?

Numerically, the deviance is 1.5896×10^4 for the logistic regression, and 1.5122×10^4 for the GAM. We can go through the testing procedure outlined in §12.6. We need a simulator (which presumes that the logistic regression model is true), and we need to calculate the difference in deviance on simulated data many times.

```
# Simulate from the fitted logistic regression model for Snoqualmie
# Presumes: fitted values of the model are probabilities.
snoq.sim <- function(model=snoq.logistic) {
  fitted.probs=fitted(model)
  return(rbinom(n=length(fitted.probs),size=1,prob=fitted.probs))
}
```

A quick check of the simulator against the observed values:

```
summary(ifelse(snoq[,1]>0,1,0))
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   1.000   0.526   1.000   1.000
summary(snoq.sim())
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   1.000   0.525   1.000   1.000
```

This suggests that the simulator is not acting crazily.

Now for the difference in deviances:

```
# Simulate from fitted logistic regression, re-fit logistic regression and
# GAM, calculate difference in deviances
diff.dev <- function(model=snoq.logistic,x=snoq[,"today"]) {
  y.new <- snoq.sim(model)
  GLM.dev <- glm(y.new ~ x,family=binomial)$deviance
  GAM.dev <- gam(y.new ~ s(x),family=binomial)$deviance
  return(GLM.dev-GAM.dev)
}
```

A single run of this takes about 0.6 seconds on my computer.

Finally, we calculate the distribution of difference in deviances under the null (that the logistic regression is properly specified), and the corresponding p-value:

```
library(mgcv)
plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)", ylab="Positive precipitation tomo
rug(snoq$today,side=1,col="grey")
pred.bands(snoq.logistic,data.plot)
lines(snoq.spline,col="red")
snoq.gam <- gam((tomorrow>0)~s(today),data=snoq,family=binomial)
pred.bands(snoq.gam,data.plot,"blue")
```

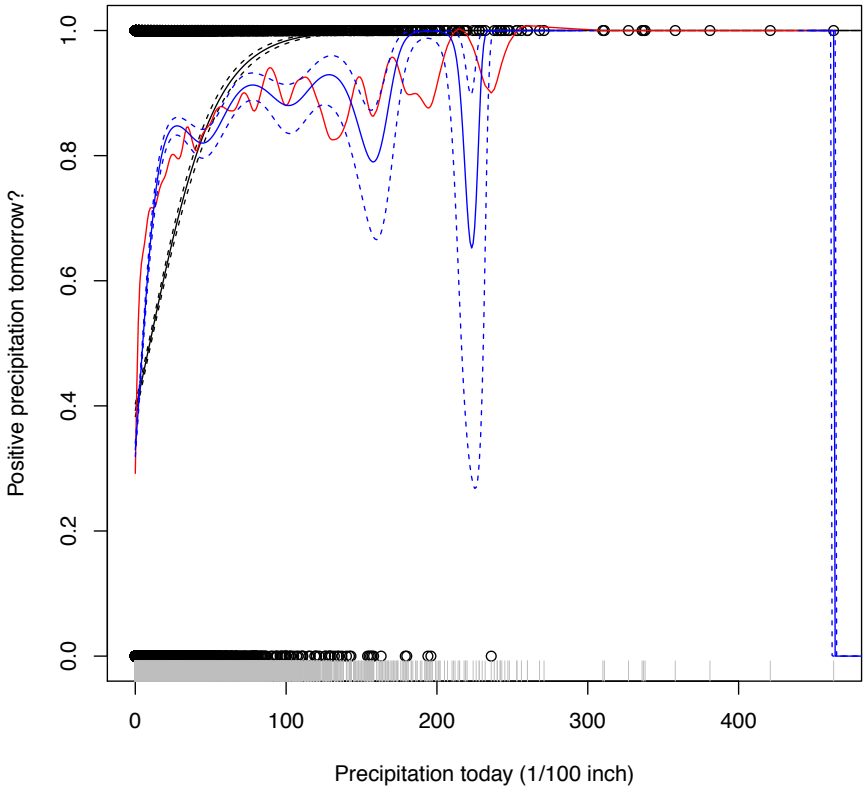

FIGURE 12.9: *As Figure 12.8, but with the addition of a generalized additive model (blue line) and its confidence limits (dashed blue lines).*

```
diff.dev.obs <- snoq.logistic$deviance - snoq.gam$deviance
null.dist.of.diff.dev <- replicate(100,diff.dev())
p.value <- (1+sum(null.dist.of.diff.dev > diff.dev.obs))/(1+length(null.dist.of.diff.dev))
```

Using a thousand replicates takes about 67 seconds, or a bit over a minute; it gives a p-value of $< 1/101$. (A longer run of 1000 replicates, not shown, gives a p-values of $< 10^{-3}$.)

Having detected that there is a problem with the logistic model, we can ask where it lies. We *could* just use the GAM, but it's more interesting to try to diagnose what's going on.

In this respect Figure 12.9 is actually a little misleading, because it leads the eye to emphasize the disagreement between the models at large X, when actually there are very few data points there, and so even large differences in predicted probabilities there contribute little to the over-all likelihood difference. What is actually more important is what happens at $X = 0$, which contains a very large number of observations (about 47% of all observations), and which we have reason to think is a special value anyway.

Let's try introducing a dummy variable for $X = 0$ into the logistic regression, and see what happens. It will be convenient to augment the data frame with an extra column, recording 1 whenever $X = 0$ and 0 otherwise.

```
snoq2 <- data.frame(snoq,dry=ifelse(snoq$today==0,1,0))
snoq2.logistic <- glm((tomorrow > 0) ~ today + dry,data=snoq2,family=binomial)
snoq2.gam <- gam((tomorrow > 0) ~ s(today) + dry,data=snoq2,family=binomial)
```

Notice that I allow the GAM to treat zero as a special value as well, by giving it access to that dummy variable. In principle, with enough data it can decide whether or not that is useful on its own, but since we have guessed that it is, we might as well include it. The new GLM has a deviance of 1.4955×10^4, lower than even the GAM before, and the new GAM has a deviance of 1.4842×10^4. I will leave repeating the specification test as an exercise. Figure 12.10 shows the data and the two new models. These are *extremely* close to each other at low percipitation, and diverge thereafter. The new GAM is the smoothest model we've nonparametric model we've seen yet, which suggests that before the it was being under-smoothed to help capture the special value at zero.

Let's turn now to looking at calibration. The actual fraction of no-precipitation days which are followed by precipitation is

```
signif(mean(snoq$tomorrow[snoq$today==0]>0),3)
## [1] 0.287
```

What does the new logistic model predict?

```
signif(predict(snoq2.logistic,
               newdata=data.frame(today=0,dry=1),type="response"),3)
##     1
## 0.287
```

```
plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)",
  ylab="Positive precipitation tomorrow?")
rug(snoq$today,side=1,col="grey")
data.plot=data.frame(data.plot,dry=ifelse(data.plot$today==0,1,0))
lines(snoq.spline,col="red")
pred.bands(snoq2.logistic,data.plot)
pred.bands(snoq2.gam,data.plot,"blue")
```

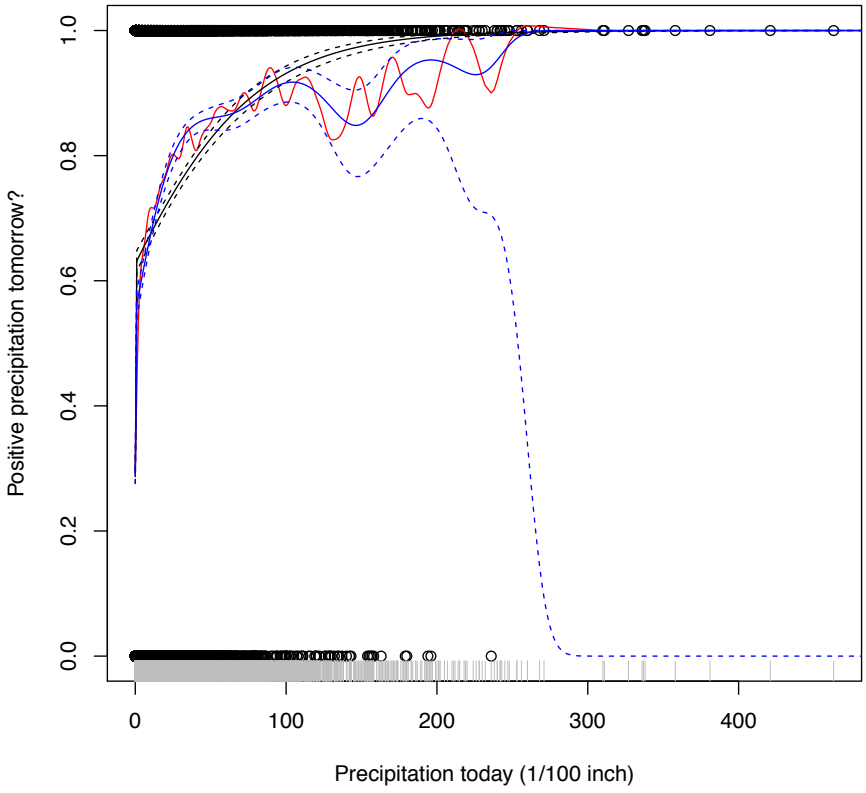

FIGURE 12.10: *As Figure 12.9, but allowing the two models to use a dummy variable indicating when today is completely dry ($X = 0$).*

This should not be surprising — we've given the model a special parameter dedicated to getting this one probability exactly right! The hope however is that this will change the predictions made on days *with* precipitation so that they are better.

Looking at a histogram of fitted values (`hist(fitted(snoq2.logistic))`) shows a gap in the distribution of predicted probabilities below 0.63, so we'll look first at days where the predicted probability is between 0.63 and 0.64.

```
signif(mean(snoq$tomorrow[(fitted(snoq2.logistic) >= 0.63) &
                          (fitted(snoq2.logistic) < 0.64)] > 0),3)
## [1] 0.526
```

Not bad — but a bit painful to write out. Let's write a function:

```
frequency.vs.probability <- function(p.lower,p.upper=p.lower+0.01,
  model=snoq2.logistic,events=(snoq$tomorrow>0)) {
  fitted.probs <- fitted(model)
  indices <- (fitted.probs >= p.lower) & (fitted.probs < p.upper)
  ave.prob <- mean(fitted.probs[indices])
  frequency <- mean(events[indices])
  se <- sqrt(ave.prob*(1-ave.prob)/sum(indices))
  return(c(frequency=frequency,ave.prob=ave.prob,se=se))
}
```

I have added a calculation of the average predicted probability, and a crude estimate of the standard error we should expect if the observations really are binomial with the predicted probabilities[11]. Try the function out before doing anything rash:

```
frequency.vs.probability(0.63)
## frequency  ave.prob        se
##   0.52603   0.63415   0.01586
```

This agrees with our previous calculation.

Now we can do this for a lot of probability brackets:

```
f.vs.p <- sapply(c(0.28,(63:100)/100),frequency.vs.probability)
```

This comes with some unfortunate R cruft, removable thus

```
f.vs.p <- data.frame(frequency=f.vs.p["frequency",],
  ave.prob=f.vs.p["ave.prob",],se=f.vs.p["se",])
```

and we're ready to plot (Figure 12.11). The observed frequencies are generally reasonably near the predicted probabilities. While I wouldn't want to say this was the last word in weather forecasting[12], it's surprisingly good for such a simple model. I will leave calibration checking for the GAM as another exercise.

[11] This could be improved by averaging predicted variances for each point, but using probability ranges of 0.01 makes it hardly worth the effort.

[12] There is an extensive discussion of this data in Guttorp (1995, ch. 2), including many significant refinements, such as dependence across multiple days.

```
plot(frequency~ave.prob,data=f.vs.p,xlim=c(0,1),ylim=c(0,1),
  xlab="Predicted probabilities",ylab="Observed frequencies")
rug(fitted(snoq2.logistic),col="grey")
abline(0,1,col="grey")
segments(x0=f.vs.p$ave.prob,y0=f.vs.p$ave.prob-1.96*f.vs.p$se,
  y1=f.vs.p$ave.prob+1.96*f.vs.p$se)
```

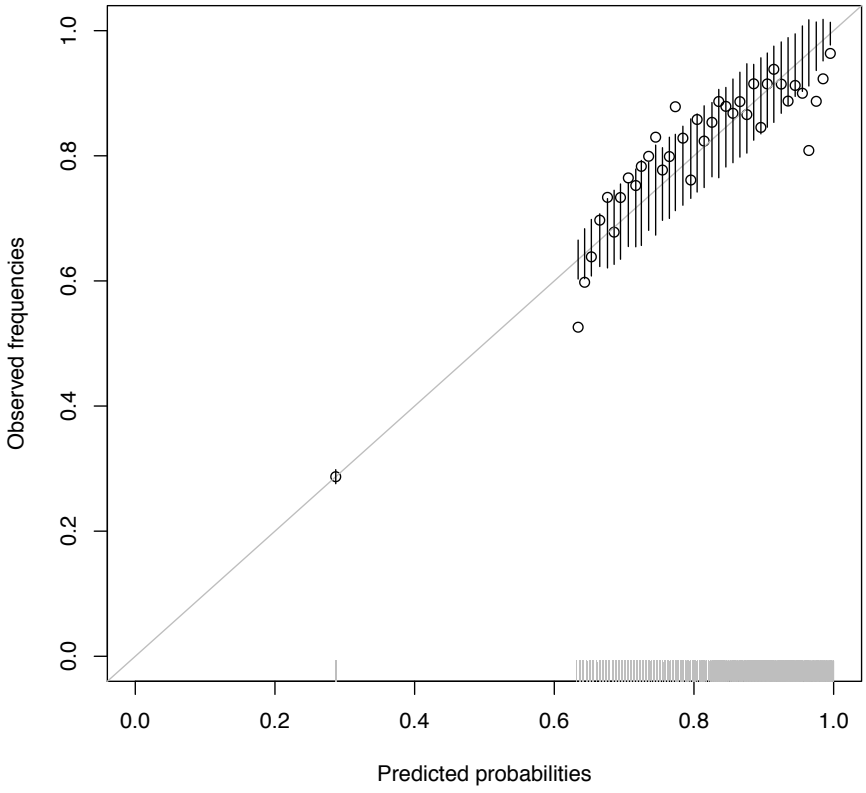

FIGURE 12.11: *Calibration plot for the modified logistic regression model* `snoq2.logistic`. *Points show the actual frequency of precipitation for each level of predicted probability. Vertical lines are (approximate) 95% sampling intervals for the frequency, given the predicted probability and the number of observations.*

12.8 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

$$\Pr\left(Y = c|\vec{X} = x\right) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}} \tag{12.23}$$

You can check that when there are only two classes (say, 0 and 1), equation 12.23 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 (Exercise 2)[13].

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

12.9 Exercises

1. "*We minimize the mis-classification rate by predicting whichever class is more likely*": Let $\hat{Y}(x)$ be our predicted class, either 0 or 1. Our error rate is then $\Pr\left(Y \neq \hat{Y}\right)$. Show that $\Pr\left(Y \neq \hat{Y}\right) = \mathbf{E}\left[(Y - \hat{Y})^2\right]$. Further show that $\mathbf{E}\left[(Y - \hat{Y})^2 \,|\, X = x\right] = \Pr(Y = 1|X = x)(1 - 2\hat{Y}(x)) + \hat{Y}^2(x)$. Conclude by showing that if $\Pr(Y = 1|X = x) > 0.5$, the risk of mis-classification is minimized by taking $\hat{Y} = 1$, that if $\Pr(Y = 1|X = x) < 0.5$ the risk is minimized by taking $\hat{Y} = 0$, and that when $\Pr(Y = 1|X = x) = 0.5$ both predictions are equally risky.

2. A multiclass logistic regression, as in Eq. 12.23, 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.

3. 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.

[13] Since we can arbitrarily chose which class's parameters to "zero out" without affecting the predicted probabilities, strictly speaking the model in Eq. 12.23 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.