Chapter 13

Generalized Linear Models and Generalized Additive Models

13.1 Generalized Linear Models and Iterative Least Squares

Logistic regression is a particular instance of a broader kind of model, called a **generalized linear model** (GLM). You are familiar, of course, from your regression class with the idea of transforming the response variable, what we've been calling Y, and then predicting the transformed variable from X. This was *not* what we did in logistic regression. Rather, we transformed the conditional expected value, and made that a linear function of X. This seems odd, because it *is* odd, but it turns out to be useful.

Let's be specific. Our usual focus in regression modeling has been the conditional expectation function, $r(x) = \mathbf{E}[Y|X = x]$. In plain linear regression, we try to approximate $r(x)$ by $\beta_0 + x \cdot \beta$. In logistic regression, $r(x) = \mathbf{E}[Y|X = x] = \Pr(Y = 1|X = x)$, and it is a transformation of $r(x)$ which is linear. The usual notation says

$$\eta(x) = \beta_0 + x\dot{c}\dot{\beta} \tag{13.1}$$

$$\eta(x) = \log \frac{r(x)}{1 - r(x)} \tag{13.2}$$

$$= g(r(x)) \tag{13.3}$$

defining the logistic **link function** by $g(m) = \log m/(1 - m)$. The function $\eta(x)$ is called the **linear predictor**.

Now, the naive strategy for estimating this model would be to all the transformation g to the response. But Y is always zero or one, so $g(Y) = \pm\infty$, and regression will not be helpful here. The standard strategy is instead to use (what else?) Taylor expansion. Specifically, we try expanding $g(Y)$ around $r(x)$, and stop at first order:

$$g(Y) \approx g(r(x)) + (Y - r(x))g'(r(x)) \tag{13.4}$$

$$= \eta(x) + (Y - r(x))g'(r(x)) \equiv z \tag{13.5}$$

We *define* this to be our effective response after transformation. Notice that if there were no noise, so that y was always equal to its conditional mean $r(x)$, then regressing z on x would give us back the coefficients β_0, β. What this suggests is that we can *estimate* those parameters by regressing z on x.

The term $Y - r(x)$ always has expectation zero, so it acts like the noise, with the factor of g' telling us about how the noise is scaled by the transformation. This lets us work out the variance of z:

$$\begin{aligned} \mathrm{Var}[Z|X=x] &= \mathrm{Var}[\eta(x)|X=x] + \mathrm{Var}\left[(Y-r(x))g'(r(x))|X=x\right] && (13.6) \\ &= 0 + (g'(r(x)))^2 \mathrm{Var}[Y|X=x] && (13.7) \end{aligned}$$

For logistic regression, with Y binary, $\mathrm{Var}[Y|X=x] = r(x)(1-r(x))$. On the other hand, with the logistic link function, $g'(r(x)) = \frac{1}{r(x)(1-r(x))}$. Thus, for logistic regression, $\mathrm{Var}[Z|X=x] = [r(x)(1-r(x))]^{-1}$.

Because the variance of Z changes with X, this is a heteroskedastic regression problem. As we saw in chapter 6, the appropriate way of dealing with such a problem is to use weighted least squares, with weights inversely proportional to the variances. This means that the weight at x should be proportional to $r(x)(1-r(x))$. Notice two things about this. First, the weights depend on the current guess about the parameters. Second, we give little weight to cases where $r(x) \approx 0$ or where $r(x) \approx 1$, and the most weight when $r(x) = 0.5$. This focuses our attention on places where we have a lot of potential information — the distinction between a probability of 0.499 and 0.501 is just a lot easier to discern than that between 0.000 and 0.002!

We can now put all this together into an estimation strategy for logistic regression.

1. Get the data $(x_1, y_1), \ldots (x_n, y_n)$, and some initial guesses β_0, β.
2. until β_0, β converge
   (a) Calculate $\eta(x_i) = \beta_0 + x_i \cdot \beta$ and the corresponding $r(x_i)$
   (b) Find the effective transformed responses $z_i = \eta(x_i) + \frac{y_i - r(x_i)}{r(x_i)(1-r(x_i))}$
   (c) Calculate the weights $w_i = r(x_i)(1-r(x_i))$
   (d) Do a weighted linear regression of z_i on x_i with weights w_i, and set β_0, β to the intercept and slopes of this regression

Our initial guess about the parameters tells us about the heteroskedasticity, which we use to improve our guess about the parameters, which we use to improve our guess about the variance, and so on, until the parameters stabilize. This is called **iterative reweighted least squares** (or "iterative weighted least squares", "iteratively weighted least squares", "iterated reweighted least squares", etc.), abbreviated IRLS, IRWLS, IWLS, etc. As mentioned in the last chapter, this turns out to be *almost* equivalent to Newton's method, at least for this problem.

13.1.1 GLMs in General

The set-up for an arbitrary GLM is a generalization of that for logistic regression. We need

- A **linear predictor**, $\eta(x) = \beta_0 + x\dot{c}\dot{\beta}$
- A **link function** g, so that $\eta(x) = g(r(x))$. For logistic regression, we had $g(r) = \log r/(1-r)$.
- A **dispersion scale function** V, so that $\mathrm{Var}[Y|X=x] = \sigma^2 V(r(x))$. For logistic regression, we had $V(r) = r(1-r)$, and $\sigma^2 = 1$.

With these, we know the conditional mean and conditional variance of the response for each value of the input variables x.

As for estimation, basically everything in the IRWLS set up carries over unchanged. In fact, we can go through this algorithm:

1. Get the data $(x_1, y_1), \ldots (x_n, y_n)$, fix link function $g(r)$ and dispersion scale function $V(r)$, and make some initial guesses β_0, β.
2. Until β_0, β converge
   (a) Calculate $\eta(x_i) = \beta_0 + x_i \cdot \beta$ and the corresponding $r(x_i)$
   (b) Find the effective transformed responses $z_i = \eta(x_i) + \frac{y_i - r(x_i)}{g}'(r(x_i))$
   (c) Calculate the weights $w_i = [(g'(r(x_i))^2 V(r(x_i))]^{-1}$
   (d) Do a weighted linear regression of z_i on x_i with weights w_i, and set β_0, β to the intercept and slopes of this regression

Notice that even if we don't know the over-all variance scale σ^2, that's OK, because the weights just have to be *proportional* to the inverse variance.

13.1.2 Example: Vanilla Linear Models as GLMs

To re-assure ourselves that we are not doing anything crazy, let's see what happens when $g(r) = r$ (the "identity link"), and $\mathrm{Var}[Y|X=x] = \sigma^2$, so that $V(r) = 1$. Then $g' = 1$, all weights $w_i = 1$, and the effective transformed response $z_i = y_i$. So we just end up regressing y_i on x_i with no weighting at all — we do ordinary least squares. Since neither the weights nor the transformed response will change, IRWLS will converge exactly after one step. So if we get rid of all this nonlinearity and heteroskedasticity and go all the way back to our very first days of doing regression, we get the OLS answers we know and love.

13.1.3 Example: Binomial Regression

In many situations, our response variable y_i will be an integer count running between 0 and some pre-determined upper limit n_i. (Think: number of patients in a hospital ward with some condition, number of children in a classroom passing a test, number

of widgets produced by a factory which are defective, number of people in a village with some genetic mutation.) One way to model this would be as a binomial random variable, with n_i trials, and a success probability p_i which was a logistic function of predictors x. The logistic regression we have done so far is the special case where $n_i = 1$ always. I will leave it as an EXERCISE (1) for you to work out the link function and the weights for general binomial regression, where the n_i are treated as known.

One implication of this model is that each of the n_i "trials" aggregated together in y_i is independent of all the others, at least once we condition on the predictors x. (So, e.g., whether any student passes the test is independent of whether any of their classmates pass, once we have conditioned on, say, teacher quality and average previous knowledge.) This may or may not be a reasonable assumption. When the successes or failures are dependent, even after conditioning on the predictors, the binomial model will be mis-specified. We can either try to get more information, and hope that conditioning on a richer set of predictors makes the dependence go away, or we can just try to account for the dependence by modifying the variance ("overdispersion" or "underdispersion"); we'll return to both topics later.

13.1.4 Poisson Regression

Recall that the Poisson distribution has probability mass function

$$p(y) = frac{e^{-\mu}\mu^y}{y!} \tag{13.8}$$

with $\mathbf{E}[Y] = \mathrm{Var}[Y] = \mu$. As you remember from basic probability, a Poisson distribution is what we get from a binomial if the probability of success per trial shrinks towards zero but the number of trials grows to infinity, so that we keep the mean number of successes the same:

$$\mathrm{Binom}(n, \mu/n) \rightsquigarrow \mathrm{Pois}(\mu) \tag{13.9}$$

This makes the Poisson distribution suitable for modeling counts with no fixed upper limit, but where the probability that any one of the many individual trials is a success is fairly low. If μ is allowed to be depend on the predictor variables, we get Poisson regression. Since the variance is equal to the mean, Poisson regression is always going to be heteroskedastic.

Since μ has to be non-negative, a natural link function is $g(\mu) = \log\mu$. This produces $g'(\mu) = 1/\mu$, and so weights $w = \mu$. When the expected count is large, so is the variance, which normally would reduce the weight put on an observation in regression, but in this case large expected counts also provide more information about the coefficients, so they end up getting increasing weight.

13.1.5 Uncertainty

Standard errors for coefficients can be worked out as in the case of weighted least squares for linear regression. Confidence intervals for the coefficients will be approximately Gaussian in large samples, for the usual likelihood-theory reasons, when the

model is properly specified. One can, of course, also use either a parametric bootstrap, or resampling of cases/data-points to assess uncertainty.

Resampling of residuals can be trickier, because it is not so clear what counts as a residual. When the response variable is continuous, we can get "standardized" or "Pearson" residuals, $\hat{\epsilon}_i = \frac{y_i - \widehat{\mu}(x_i)}{\sqrt{\widehat{V(\mu(x_i))}}}$, resample them to get $\tilde{\epsilon}_i$, and then add $\tilde{\epsilon}_i \sqrt{\widehat{V(\mu(x_i))}}$ to the fitted values. This does not really work when the response is discrete-valued, however.

13.2 Generalized Additive Models

In the development of generalized linear models, we use the link function g to relate the conditional mean $\mu(x)$ to the linear predictor $\eta(x)$. But really nothing in what we were doing required η to be *linear* in x. In particular, it all works perfectly well if η is an additive function of x. We form the effective responses z_i as before, and the weights w_i, but now instead of doing a linear regression on x_i we do an additive regression, using backfitting (or whatever). This gives us a generalized additive model (GAM).

Essentially everything we know about the relationship between linear models and additive models carries over. GAMs converge somewhat more slowly as n grows than do GLMs, but the former have less bias, and strictly include GLMs as special cases. The transformed (mean) response is related to the predictor variables not just through coefficients, but through whole partial response functions. If we want to test whether a GLM is well-specified, we can do so by comparing it to a GAM, and so forth.

In fact, one could even make $\eta(x)$ an arbitrary smooth function of x, to be estimated through (say) kernel smoothing of z_i on x_i. This is rarely done, however, partly because of curse-of-dimensionality issues, but also because, if one is going to go that far, one might as well just use kernels to estimate conditional distributions, as we will see in Chapter 15.

13.3 Weather Forecasting in Snoqualmie Falls

To make the use of logistic regression and GLMs concrete, 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 13.1)[1]. 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.

```
snoqualmie <- read.csv("snoqualmie.csv",header=FALSE)
# Turn into one big vector without year breaks
snoqualmie <- unlist(snoqualmie)
# Remove NAs from non-leap-years
snoqualmie <- na.omit(snoqualmie)
```

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 either one of the local hydroelectric power plants, or the tourist attraction of the Falls themselves. Examining the distribution of the data (Figures 13.2 and 13.3) 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 13.3, as well as common experience, that the coefficient on X should be positive.[2]

Before fitting the logistic regression, it's convenient to re-shape the data:

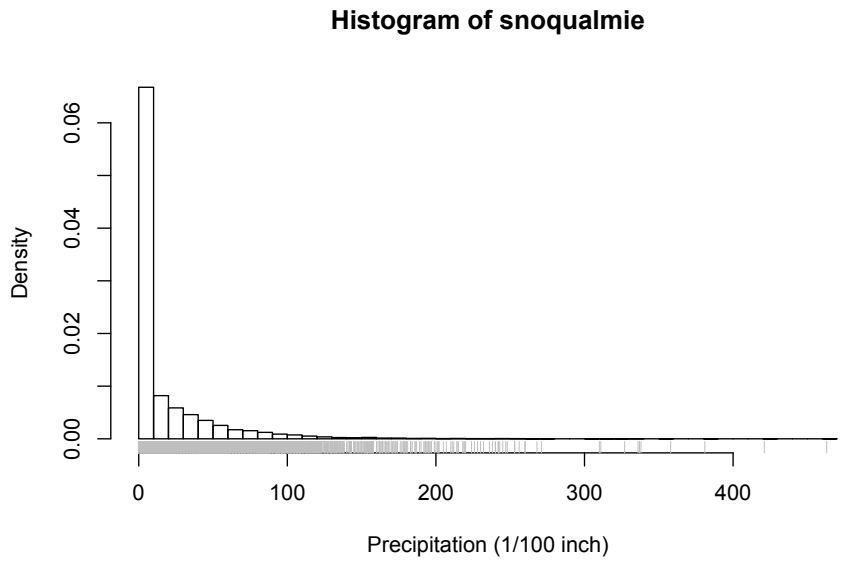
```
vector.to.pairs <- function(v) {
  v <- as.numeric(v)
  n <- length(v)
  return(cbind(v[-1],v[-n]))
}
snoq.pairs <- vector.to.pairs(snoqualmie)
colnames(snoq.pairs) <- c("tomorrow","today")
snoq <- as.data.frame(snoq.pairs)
```

This creates a two-column array, where the first column is the precipitation on day $i+1$, and the second column is the precipitation on day i (hence the column names). Finally, I turn the whole thing into a data frame.

[1] I learned of this data set from Peter Guttorp's *Stochastic Modeling of Scientific Data*; the data file is available from `http://www.stat.washington.edu/peter/stoch.mod.data.html`.

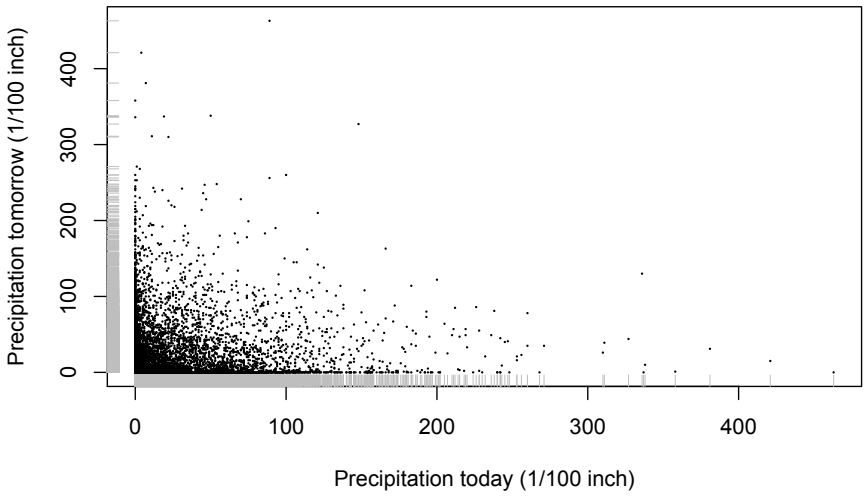
[2] This does not attempt to model *how much* precipitation there will be tomorrow, if there is any. We could make that a separate model, if we can get this part right.

Figure 13.1: Snoqualmie Falls, Washington, on a sunny 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!]]




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

Figure 13.2: Histogram of the amount of daily precipitation at Snoqualmie Falls




```
plot(snoqualmie[-length(snoqualmie)],snoqualmie[-1],
     xlab="Precipitation today (1/100 inch)",
     ylab="Precipitation tomorrow (1/100 inch)",cex=0.1)
rug(snoqualmie[-length(snoqualmie)],side=1,col="grey")
rug(snoqualmie[-1],side=2,col="grey")
```

Figure 13.3: 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.

Now fitting is straightforward:

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

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

```
> summary(snoq.logistic)

Call:
glm(formula = (tomorrow > 0) ~ today, family = binomial, data = snoq)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3713  -1.1805   0.9536   1.1693   1.1744

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) 0.0071899  0.0198430   0.362    0.717
today       0.0059232  0.0005858  10.111   <2e-16 ***
---
```

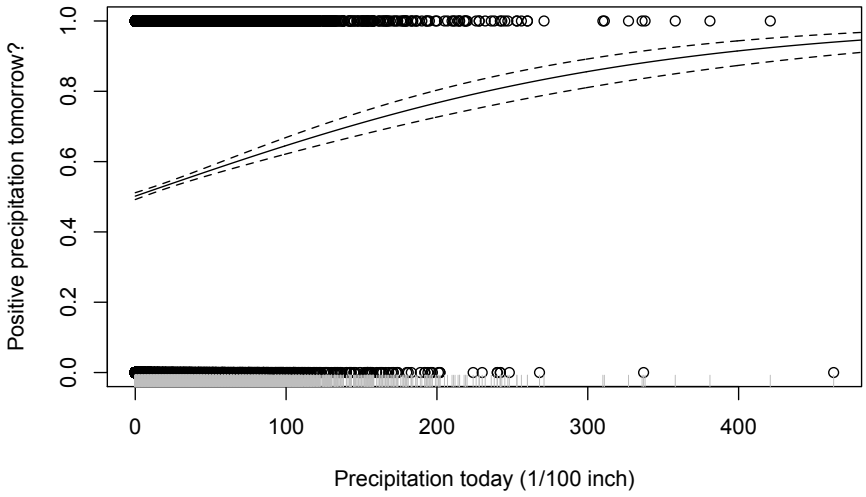
(I have cut off some uninformative bits of the output.) The coefficient on X, the amount of precipitation today, is indeed positive, and (if we can trust R's calculations) highly significant. There is also an intercept term, which is slight positive, but not very significant. We can see what the intercept term means by considering what happens when $X = 0$, i.e., on days of no precipitation. The linear predictor is then $0.0072 + 0 * (0.0059) = 0.0072$, and the predicted probability of precipitation is $e^{0.0072}/(1 + e^{0.0072}) = 0.502$. That is, even when there is no precipitation today, we predict that it is slightly more probable than not that there will be some precipitation tomorrow.[3]

We can get a more global view of what the model is doing by plotting the data and the predictions (Figure 13.4). 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, and actually slightly narrower at the far right than at intermediate values of X (Figure 13.3).

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 13.6). 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.

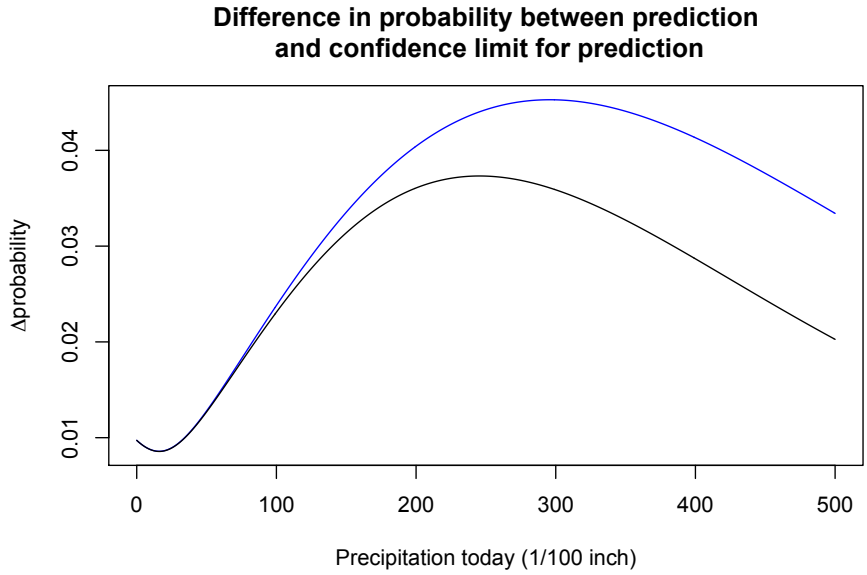
The result is in not-terribly-bad agreement with the logistic regression up to about 1.2 or 1.3 inches of precipitation, after which it runs significantly below the logistic

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



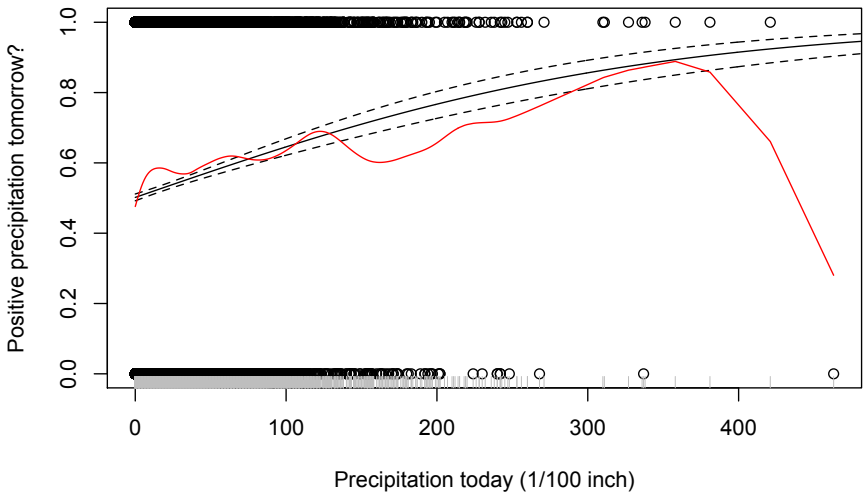

```
plot((tomorrow>0)~today,data=snoq,xlab="Precipitation today (1/100 inch)",
     ylab="Positive precipitation tomorrow?")
rug(snoq$today,side=1,col="grey")

data.plot <- data.frame(today=(0:500))
logistic.predictions <- predict(snoq.logistic,newdata=data.plot,se.fit=TRUE)
lines(0:500,ilogit(logistic.predictions$fit))
lines(0:500,ilogit(logistic.predictions$fit+1.96*logistic.predictions$se.fit),
      lty=2)
lines(0:500,ilogit(logistic.predictions$fit-1.96*logistic.predictions$se.fit),
      lty=2)
```

Figure 13.4: 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(0:500,ilogit(logistic.predictions$fit)
           -ilogit(logistic.predictions$fit-1.96*logistic.predictions$se.fi
    type="l",col="blue",xlab="Precipitation today (1/100 inch)",
    main="Difference in probability between prediction\n
and confidence limit for prediction",
    ylab = expression(paste(Delta,"probability")))
lines(0:500,ilogit(logistic.predictions$fit+1.96*logistic.predictions$se.fi
            -ilogit(logistic.predictions$fit))
```

Figure 13.5: Distance from the fitted probability to the upper (black) and lower (blue) confidence limits. Notice that the two are not equal, and somewhat smaller at very large values of X than at intermediate ones. (Why?)




```
snoq.spline <- smooth.spline(x=snoq$today,y=(snoq$tomorrow>0))
lines(snoq.spline,col="red")
```

Figure 13.6: As Figure 13.4, plus a smoothing spline (red).

regression, rejoins it around 3.5 inches of precipitation, and then (as it were) falls off a cliff.

We can 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 13.7) is actually quite similar to the spline, but a bit better behaved, 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 usually but not always 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 18079.69 for the logistic regression, and 18036.77 for the GAM. We can go through the testing procedure outlined in the notes for lecture 14. 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)
  n <- length(fitted.probs)
  new.binary <- rbinom(n,size=1,prob=fitted.probs)
  return(new.binary)
}
```

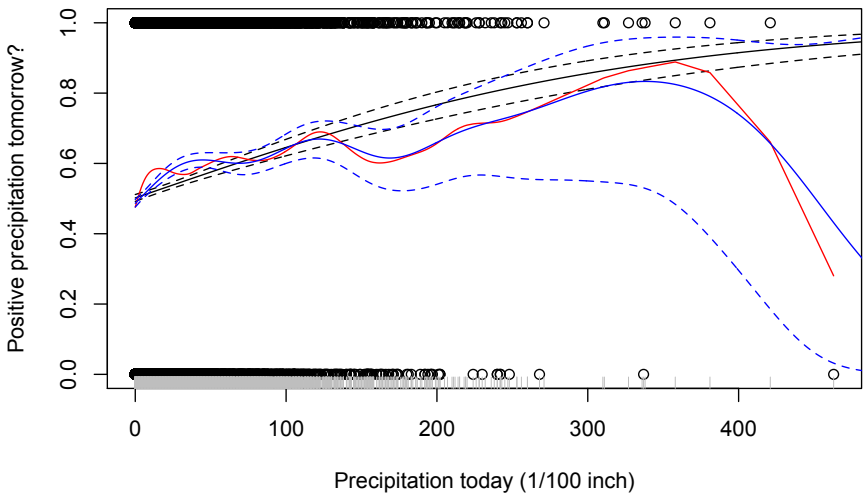
A quick check of the simulator against the observed values:

```
> summary(ifelse(snoq[,1]>0,1,0))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  1.0000  0.5262  1.0000  1.0000
> summary(snoq.sim())
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  1.0000  0.5264  1.0000  1.0000
```

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[,2]) {
  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)
}
```




```
library(mgcv)
snoq.gam <- gam((tomorrow>0)~s(today),data=snoq,family=binomial)
gam.predictions <- predict.gam(snoq.gam,newdata=data.plot,se.fit=TRUE)
lines(0:500,ilogit(gam.predictions$fit),col="blue")
lines(0:500,ilogit(gam.predictions$fit+1.96*gam.predictions$se.fit),
      col="blue",lty=2)
lines(0:500,ilogit(gam.predictions$fit-1.96*gam.predictions$se.fit),
      col="blue",lty=2)
```

Figure 13.7: As Figure 13.6, but with the addition of a generalized additive model (blue line) and its confidence limits (dashed blue lines). *Note:* the `predict` function in the `gam` package does not allow one to calculate standard errors for new data. You may need to un-load the `gam` library first, with `detach(package:gam)`.

A single run of this takes about 1.5 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:

```
diff.dev.obs <- snoq.logistic$deviance - snoq.gam$deviance
null.dist.of.diff.dev <- replicate(1000,diff.dev())
p.value <- (1+sum(null.dist.of.diff.dev > diff.dev.obs))/(1+length(null.dis
```

Using a thousand replicates takes about 1500 seconds, or roughly 25 minutes, which is substantial, but not impossible; it gave a p-value of $< 10^{-3}$, and the following sampling distribution:

```
> summary(null.dist.of.diff.dev)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
 0.000097  0.002890  0.016770  2.267000  2.897000 29.750000
```

(A preliminary trial run of only 100 replicates, taking a few minutes, gave

```
> summary(null.dist.of.diff.dev)
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
 0.000291  0.002681  0.013700  2.008000  2.121000 27.820000
```

which implies a p-value of < 0.01. This would be good enough for many practical purposes.)

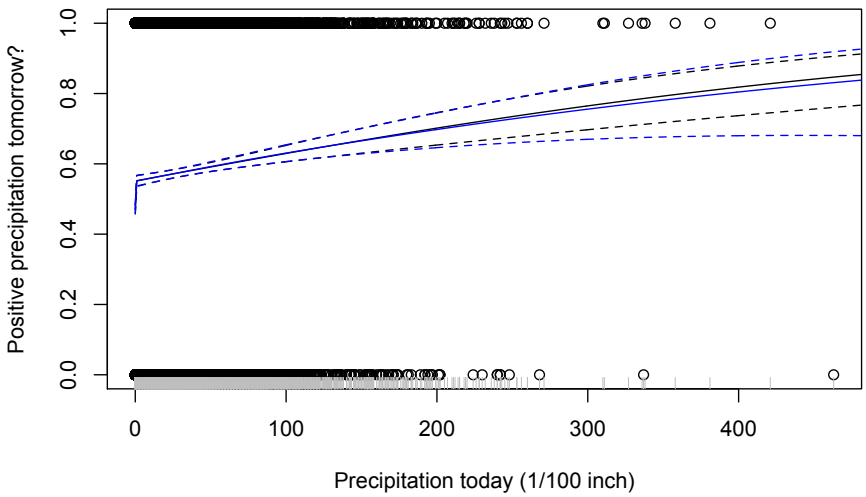
Having detected that there is a problem with the GLM, we can ask where it lies. We *could* just use the GAM, but it's more interesting to try to diagnose what's going on.

In this respect Figure 13.7 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=binomi
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. Figure 13.8 shows the data and the two new models. These are *extremely* close to each other. The new GLM has a deviance of 18015.65, lower than even the GAM before, and the new GAM has a deviance of 18015.21. The p-value is essentially 1




```
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$today==0,1,0))
logistic.predictions2 <- predict(snoq2.logistic,newdata=data.plot,se.fit=TRUE)
lines(0:500,ilogit(logistic.predictions2$fit))
lines(0:500,ilogit(logistic.predictions2$fit+1.96*logistic.predictions2$se.fit),
      lty=2)
lines(0:500,ilogit(logistic.predictions2$fit-1.96*logistic.predictions2$se.fit),
      lty=2)
gam.predictions2 <- predict.gam(snoq2.gam,newdata=data.plot,se.fit=TRUE)
lines(0:500,ilogit(gam.predictions2$fit),col="blue")
lines(0:500,ilogit(gam.predictions2$fit+1.96*gam.predictions2$se.fit),
      col="blue",lty=2)
lines(0:500,ilogit(gam.predictions2$fit-1.96*gam.predictions2$se.fit),
      col="blue",lty=2)
```

Figure 13.8: As Figure 13.7, but allowing the two models to use a dummy variable indicating when today is completely dry ($X = 0$).

— and yet we know that the test, with this test, *does* have power to detect departures from the parametric model. This is very promising.

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

```
> mean(snoq$tomorrow[snoq$today==0]>0)
[1] 0.4702199
```

What does the new logistic model predict?

```
> predict(snoq2.logistic,newdata=data.frame(today=0,dry=1),type="response")
        1
0.4702199
```

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.

We'll tackle this through calibration. Looking at a histogram of fitted values (`hist(fitted(snoq2.logistic))`) shows a gap in the distribution of predicted probabilities between 0.47 and about 0.55, so we'll look first at days where the predicted probability is between 0.55 and 0.56.

```
> mean(snoq$tomorrow[(fitted(snoq2.logistic) >= 0.55)
                     & (fitted(snoq2.logistic) < 0.56)] > 0)
[1] 0.5474882
```

Not bad — but a bit painful to write out. Let's write a function[4].

```
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))
  out <- list(frequency=frequency,ave.prob=ave.prob,se=se)
  return(out)
}
```

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[5]. Try the function out before doing anything rash:

```
> frequency.vs.probability(0.55)
$frequency
```

[4] Thanks to Terra Mack for comments on an earlier version.

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

```
[1] 0.5474882

$ave.prob
[1] 0.5548081

$se
[1] 0.00984567
```

This agrees with our previous calculation.

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

```
f.vs.p <- sapply((55:74)/100,frequency.vs.probability)
```

This comes with some unfortunate R cruft, removable thus

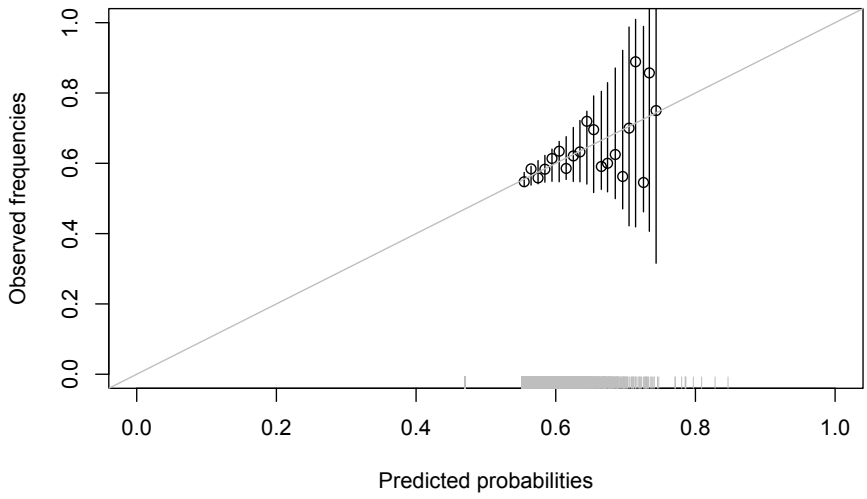
```
f.vs.p <- data.frame(frequency=unlist(f.vs.p["frequency",]),
                     ave.prob=unlist(f.vs.p["ave.prob",]),
                     se=unlist(f.vs.p["se",]))
```

and we're ready to plot (Figure 13.9). The observed frequencies are generally quite near to the predicted probabilites, especially when the number of observations is large and so the sample frequency *should* be close to the true probability. While I wouldn't want to say this was the last word in weather forecasting[6], it's surprisingly good for such a simple model.

13.4 Exercises

1. In binomial regression, we have $Y|X = x$ Binom$(n, p(x))$, where $p(x)$ follows a logistic model. Work out the link function $g(\mu)$, the variance function $V(\mu)$, and the weights w, assuming that n is known and not random.
2. Homework 4, on predicting the death rate in Chicago, is a good candidate for using Poisson regression. Repeat the exercises in that problem set with Poisson-response GAMs. How do the estimated functions change? Why is this any different from just taking the log of the death counts, as we did in the homework?

[6] There is an extensive discussion of this data in chapter 2 of Guttorp's book, including many significant refinements, such as dependence across multiple days.




```
plot(f.vs.p$ave.prob,f.vs.p$frequency,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 13.9: 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.