Chapter 11

DJM

20 March 2016

Chapter 11

- Here we finally learn about **logistic regression**.
- This is one of *many* "classifiers", ways to predict binary outcomes.
- We've waited so long to introduce it for two reasons:
  1. The algorithm for estimating the coefficients is weighted least squares
  2. It is easy to do as a GAM, so learn all this just once

What is logistic regression

- Let Y be the outcome of a random (unfair) coin with $P(heads) = p$, then the pmf of Y is

$$P(Y = y; p) = p^y (1-p)^{1-y}.$$

- This has $\mathbb{E}[Y] = p$ and $\mathbb{V}[Y] = p(1-p)$.
- Now, $Y = 1$ or $Y = 0$, and linear models don't predict discrete things very well, so let's try to predict $\mathbb{E}[Y] = p$ instead.
- If $P(heads) = p = x^\top \beta$, then

$$P(Y = y; p) = p^y (1-p)^{1-y} = (x^\top \beta)^y (1 - x^\top \beta)^{1-y}$$

  which doesn't help us solve for β.

Why this?

- For OLS, we usually assume that Y is **real valued**.
- This means that $Y - X^\top \beta = \epsilon$ is real valued
- We spent a lot of time examining the residuals of the linear model
  1. Are they symmetric?
  2. Do they have mean zero? For all x?
  3. Does the variance seem constant across x values?
  4. Does the distribution of the residuals appear normal?
- We looked at ways of **transforming** Y or X to make this happen (take logs, add polynomials, use smoothers, interactions, etc.)
- If Y takes values 0 or 1, and we predict 0 or 1, then the residuals must be either 0 or 1 (can't be normal for sure).
- Logistic regression is a **transformation** that helps us get symmetric, real valued, potentially normal residuals:
  - We don't predict Y, we predict $\text{logit}(Y) = \log \frac{P(Y=1)}{1-P(Y=1)} = \log \frac{P(Y=1)}{P(Y=0)}$.
  - Unfortunately, we don't **see** $P(Y = 1)$, just Y, so we let the computer do some numerics

Estimation

- This is easy, use `glm(y~x1+x2, family=binomial)`
- The numerics are also easy and detailed in the text.
- There's a lot to unpack here, so we'll look at GLMs over 2 weeks.
- This week, we'll concentrate on simply estimating and an example with Logistic Regression.
- Next week, we'll look at the numeric implementation in detail, Taylor expansion, as well as examples of other GLMs.

Let's look at a simulation (bivariate x)

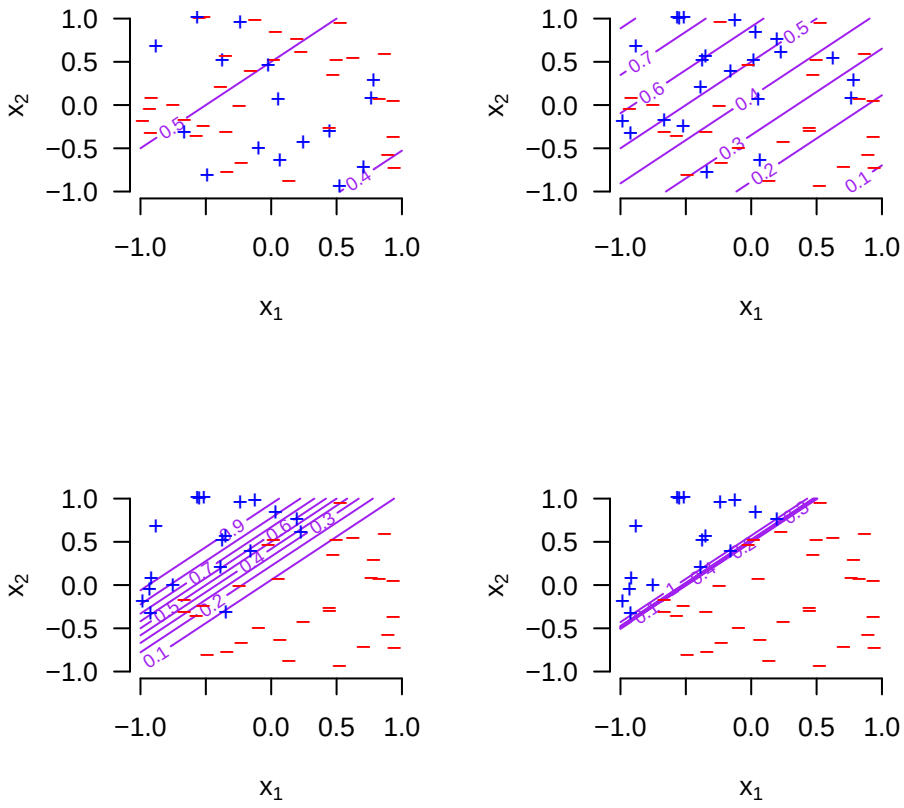
```
logit <- function(z){ # can this take in any z?
  log(z)-log(1-z)
}
ilogit <- function(z){ # what about this one?
  exp(z)/(1+exp(z))
}

# slightly modified from the text
sim.logistic <- function(X, beta.0, beta) {
  # Note: I'm passing in an x matrix => I need ncol(x)==length(beta)
  linear.parts = beta.0 + X%*%beta
  y = rbinom(nrow(X), size=1, prob=ilogit(linear.parts))
  return(data.frame(y,X))
}

plot.bilogistic.sim <- function(X, beta.0, beta, n.grid=50, col="purple", ...) {
  grid.seq <- seq(from=-1,to=1,length.out=n.grid)
  plot.grid <- as.matrix(expand.grid(grid.seq,grid.seq))
  p <- matrix(ilogit(beta.0 + (plot.grid %*% beta )), nrow=n.grid)
  contour(x=grid.seq,y=grid.seq,z=p, xlab=expression(x[1]), zlim=c(0,1),
          ylab=expression(x[2]),main="",col=col,...)
  df <- sim.logistic(X,beta.0,beta)
  points(X[,1],X[,2],pch=ifelse(df$y==1,"+","-"),col=ifelse(df$y==1,"blue","red"))
  invisible(df)
}
```

Plotting

```
X <- matrix(runif(50*2, min=-1,max=1),ncol=2)
par(mfrow=c(2,2),las=1,bty='n')
plot.bilogistic.sim(X,beta.0=-0.1,beta=c(-0.2,0.2))
df = plot.bilogistic.sim(X,beta.0=-0.5,beta=c(-1,1)) # for use in a minute
plot.bilogistic.sim(X,beta.0=-2.5,beta=c(-5,5))
plot.bilogistic.sim(X,beta.0=-2.5e2,beta=c(-5e2,5e2))
```



Estimating

```
logr <- glm(y ~ X1 + X2, data=df, family=binomial)
summary(logr,digits=2,signif.stars=FALSE)
```

```
##
## Call:
## glm(formula = y ~ X1 + X2, family = binomial, data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8890  -0.9076  -0.3915   0.9681   1.8738
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.3269     0.3360  -0.973  0.33066
## X1           -1.1332     0.5854  -1.936  0.05288 .
## X2            1.7711     0.6477   2.735  0.00624 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 68.994  on 49  degrees of freedom
## Residual deviance: 55.167  on 47  degrees of freedom
## AIC: 61.167
##
```

```
## Number of Fisher Scoring iterations: 4
```

```
table(ifelse(fitted(logr)<0.5, 0, 1), df$y)
```

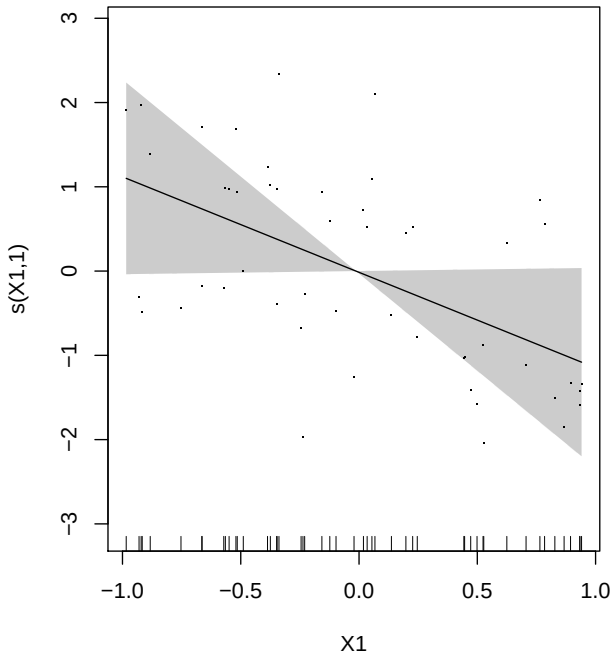
```
##
##       0  1
##   0 21  7
##   1  6 16
```

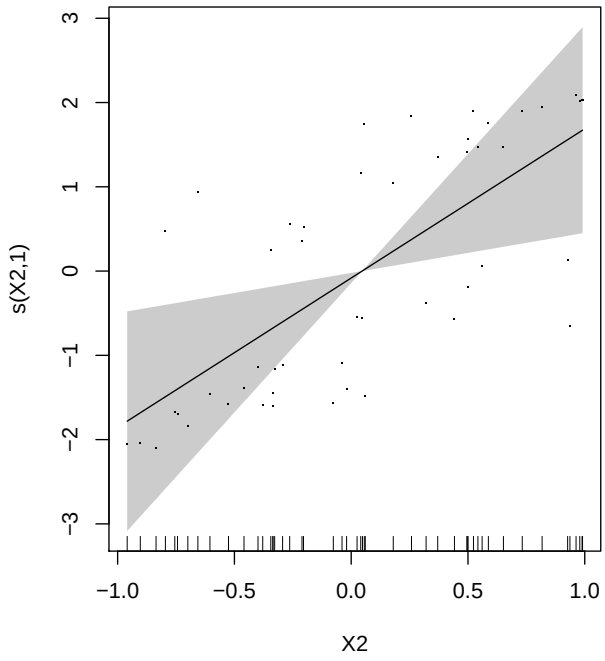
A GAM

```
library(mgcv)
(gam.1 <- gam(y~s(X1)+s(X2), data=df, family=binomial))
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## y ~ s(X1) + s(X2)
##
## Estimated degrees of freedom:
## 1 1  total = 3
##
## UBRE score: 0.2233347
```

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






```
table(ifelse(fitted(gam.1)<0.5,0,1),df$y)
```

```
##
##       0  1
##   0 21  7
```

```
##   1  6 16
```

Real data example

This dataset comes from a study investigating the causes of civil wars.

Every row of the data represents a combination of a country and a five-year interval: for example, the first row is `Afghanistan, 1960`, which really means Afghanistan, 1960-1965.

- The variables are:
  1. Country name
  2. The year
  3. In indicator (1 if yes, 0 if no) for whether a civil war **began** during the period. `NA` means an ongoing war.
  4. A measure of dependence of the country's dependence on commodity exports.
  5. % of males enrolled in secondary school (occasionally exceeds 100 for unknown reasons)
  6. Annual GDP growth rate
  7. A measure of geographic concentration of the population (0 if evenly spread, 1 if everyone lives in one city).
  8. The number of months since the country's last war (or WWII)
  9. log(population)
  10. An index measuring how fractured the country is along ethnic/religious lines
  11. An index of ethnic dominance (does one group run the country?)

  Goal: Predict civil war starts. Find important variables for making such predictions.

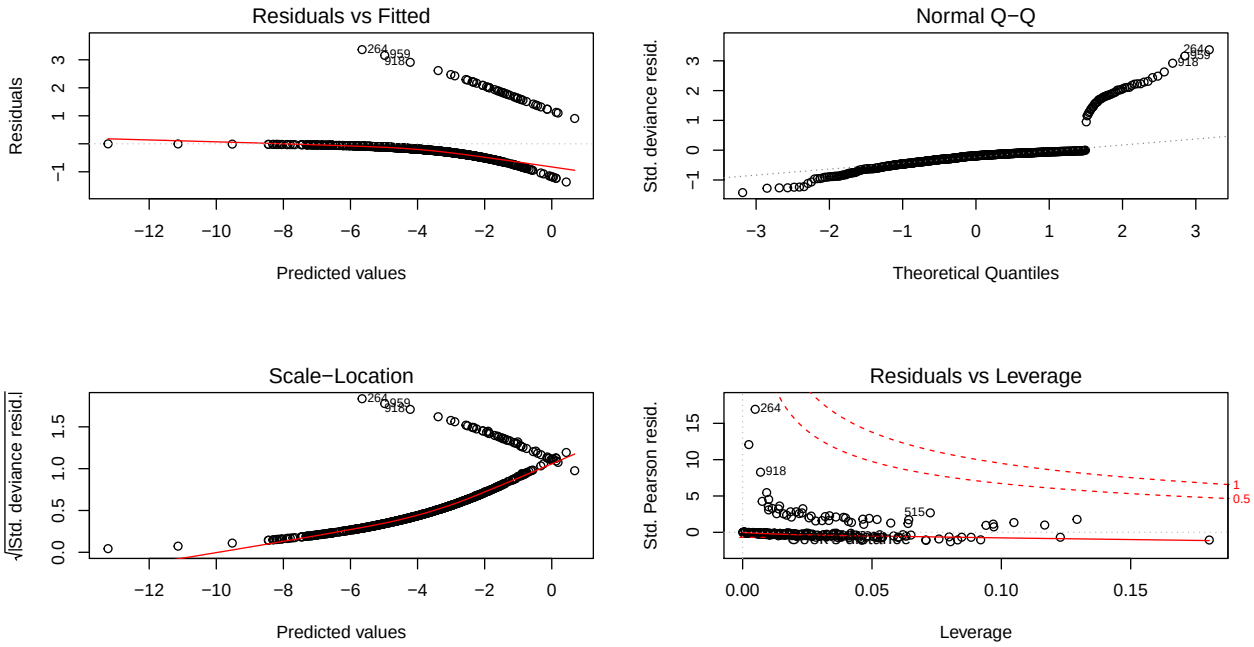
Estimate a model

```
# Read in some data
ch <- read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/15/hw/06/ch.csv")
ch <- ch[,-1] # first column is just row numbers
ch.clean <- na.omit(ch) # make a version excluding missing values (for later)
lr.ch <- glm(start ~ . -country -year +I(exports^2), data=ch, family="binomial")
```

- Why exclude `country` and `year`?
- How should I decide if I need the quadratic term?

Examine the model

```
par(mfrow=c(2,2), oma=c(0,0,2,0))
plot(lr.ch, panel=panel.smooth)
```



- What am I looking at here?

Coefficients and SEs

```
fortable1 <- summary(lr.ch)$coefficients
kable(fortable1, digits=3)
```

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-13.073	2.795	-4.677	0.000
exports	18.937	5.865	3.229	0.001
schooling	-0.032	0.010	-3.225	0.001
growth	-0.115	0.043	-2.675	0.007
peace	-0.004	0.001	-3.397	0.001
concentration	-2.487	1.005	-2.474	0.013
lnpop	0.768	0.166	4.632	0.000
fractionalization	0.000	0.000	-2.345	0.019
dominance	0.670	0.354	1.896	0.058
I(exports^2)	-29.443	11.781	-2.499	0.012

- Which are significant?
- What is the interpretation of the coefficient on `schooling`?
- What is the interpretation of the coefficient on `exports^2`?
- Remember that these are log $\frac{P(Y=1)}{P(Y=0)}$.

Making predictions

- What is the model's predicted probability for a civil war in India in the period beginning 1975?

```
india1975 <- ch[(ch$country=="India") & (ch$year==1975),]
signif(p.india75 <- predict(lr.ch,type="response",newdata=india1975),3)
```

```
##  500
## 0.35
```

- What probability would it predict for a country just like India in 1975, except that its male secondary school enrollment rate was 30 points higher?

```
india1975.sch <- india1975
india1975.sch$schooling <- india1975$schooling+30
signif(p.india75sch <- predict(lr.ch,type="response",newdata=india1975.sch),3)
```

```
##   500
## 0.173
```

- What probability would it predict for a country just like India in 1975, except that the ratio of commodity exports to GDP was 0.1 higher?

```
india1975.exp <- india1975
india1975.exp$exports <- india1975$exports+0.1
signif(p.india75exp <- predict(lr.ch, type="response", newdata=india1975.exp),3)
```

```
##   500
## 0.696
```

Note: No civil war began in India in 1975-1980.

- What about Nigeria in 1965 (code hidden)?

```
##   802
## 0.171
```

```
##    802
## 0.0741
```

```
##   802
## 0.331
```

Note: A civil war began in Nigeria in 1967 (the Biafran War)

What is different?

- In the previous slide, we changed the same predictor variables by the same amounts:
- The changes in the predicted probablities for India at 1975 are -0.177, 0.346 while for Nigeria in 1965 the change is -0.0969, 0.16.
- In a linear regression model, changing the same predictor variable by the same amount always produces the same change in the predictor. (This is the basis for the formula about how "a one unit change in x_j leads to a change of β_j units in the expected response.") This is not happening here.
- To see why it does not happen, change one of the predictor variables, say x_1, by a units, and look at the change in the predicted probability:

$$\frac{\exp\left(\beta_0+\beta_1(x_1+a)+\beta_{\mathbf{rest}}\mathbf{x}\right)}{1+\exp\left(\beta_0+\beta_1(x_1+a)+\beta_{\mathbf{rest}}\mathbf{x}\right)}-\frac{\exp\left(\beta_0+\beta_1x_1+\beta_{\mathbf{rest}}\mathbf{x}\right)}{1+\exp\left(\beta_0+\beta_1x_1+\beta_{\mathbf{rest}}\mathbf{x}\right))}$$

- If we define $c = \exp(\beta_0 + \beta_1 x_1 + \beta_{\mathbf{rest}}\mathbf{x})$, then the difference becomes

$$\frac{c \times \exp(\beta_1 a)}{1 + c \times \exp(\beta_1 a)} - \frac{c}{1+c}$$

- The change depends on the current or baseline values of all the predictor variables, not just the amount by which x_1 is changed. This is part of what it means for logistic regression to be a non-linear model.
- Notice, however, that changing the same predictor variable by the same amount will always change the **log odds** equally.

Confusion matrix

- How well is our classifier working?

```
pred.prob <- predict(lr.ch, type='response') # use later, without the flag, returns log-odds
(confusion <- table(ch.clean$start, as.integer(pred.prob>0.5), dnn=c('reality','prediction')))
```

```
##         prediction
## reality   0   1
##       0 637   5
##       1  43   3
```

- What fraction of predictions are correct?

```
signif(sum(diag(confusion))/sum(confusion),3)
```

```
## [1] 0.93
```

- Consider a "foolish pundit" who always predicts **no war**. How many predictions does he get right on the whole dataset?

```
signif(mean(ch$start==0, na.rm=TRUE),3)
```

```
## [1] 0.933
```

- What about if we restrict to when the regression makes a prediction?

```
signif(mean(ch.clean$start==0),3)
```

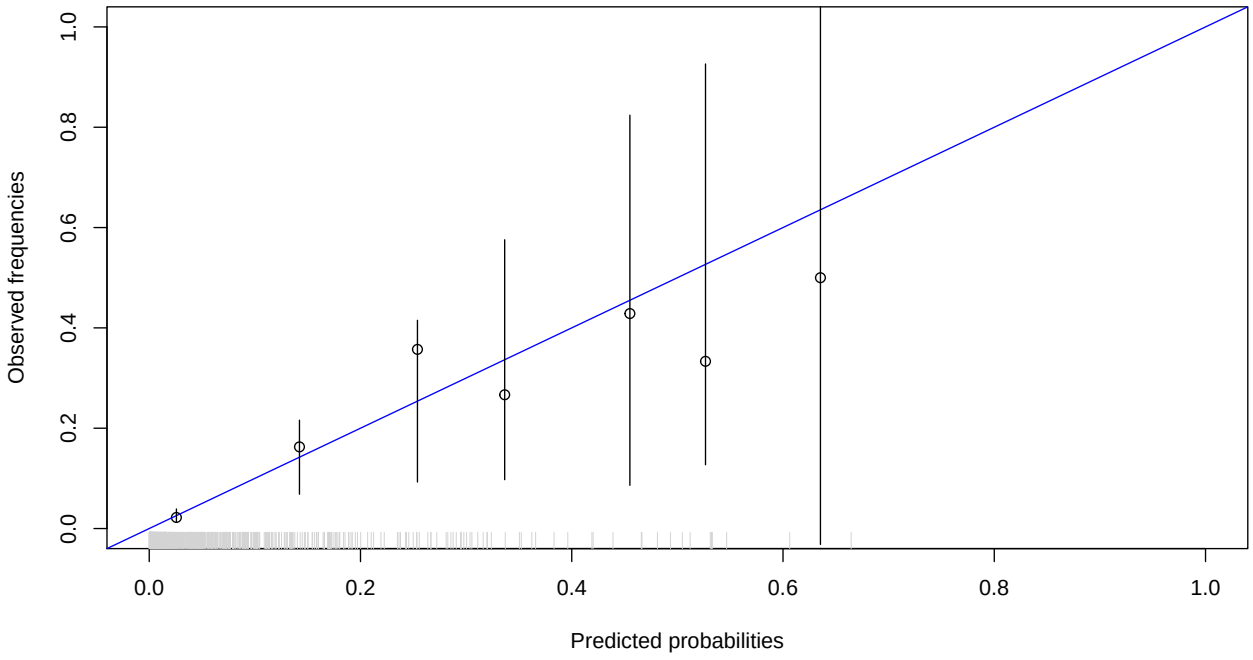
```
## [1] 0.933
```

Calibration

- To check whether a model is properly calibrated, we want to know whether the probability reflects reality.
- For example, if we look back at a meteorologist's forecasts for rain, on those days s/he said "20%" chance, did it rain 20% of the time?
- We can alter the code from the notes (slightly):

```
frequency.vs.probability <- function(p.lower, p.increment=0.01, # changed this
                                     model, events) {
    fitted.probs <- fitted(model)
    indices <- (fitted.probs >= p.lower) & (fitted.probs < p.lower+p.increment) # a change here, too
    ave.prob <- mean(fitted.probs[indices])
    frequency <- mean(events[indices])
    se <- sqrt(ave.prob * (1 - ave.prob)/sum(indices))
```

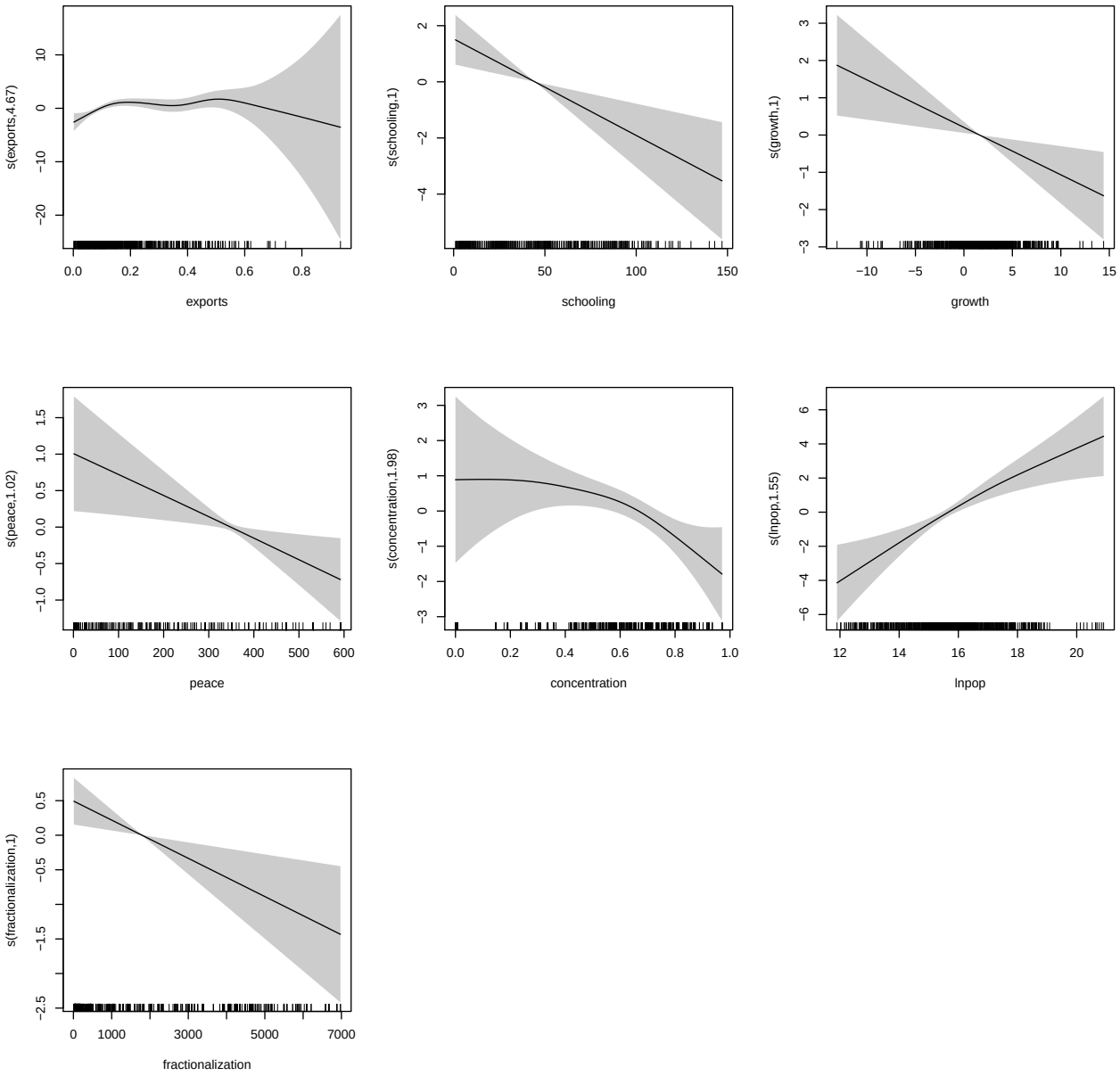
```
    return(c(frequency = frequency, ave.prob = ave.prob, se = se))
}
f.vs.p <- sapply(0:6/10,frequency.vs.probability, p.increment=0.1,
                 model=lr.ch, events=ch.clean$start)
f.vs.p <- data.frame(frequency=unlist(f.vs.p["frequency",]),
                     ave.prob=unlist(f.vs.p["ave.prob",]),
                     se=unlist(f.vs.p["se",]))
plot(frequency~ave.prob, data=f.vs.p, xlab="Predicted probabilities",
     ylab="Observed frequencies", xlim=c(0,1), ylim=c(0,1))
rug(fitted(lr.ch),col="lightgrey")
abline(0,1,col="blue")
segments(x0=f.vs.p$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)
```



- Calibration is not great, especially at the high-probability end.

Let's do a GAM

```
library(mgcv)
gam.ch <- gam(start~s(exports)+s(schooling)+s(growth)+s(peace)
          +s(concentration)+s(lnpop)+s(fractionalization)+dominance, # dominance is binary (no smoothi
          data=ch, family="binomial")
plot(gam.ch, scale=0, se=2, pages=1, shade=TRUE)
```



GAM confusion

```
pred.prob.gam <- predict(gam.ch, type="response")
(confusion.gam <- table(ch.clean$start,as.integer(pred.prob.gam>0.5), dnn=c("reality", "prediction")))
```

```
##         prediction
## reality   0   1
##       0 638   4
##       1  43   3
```

- We can calculate the percentage of correct predictions:

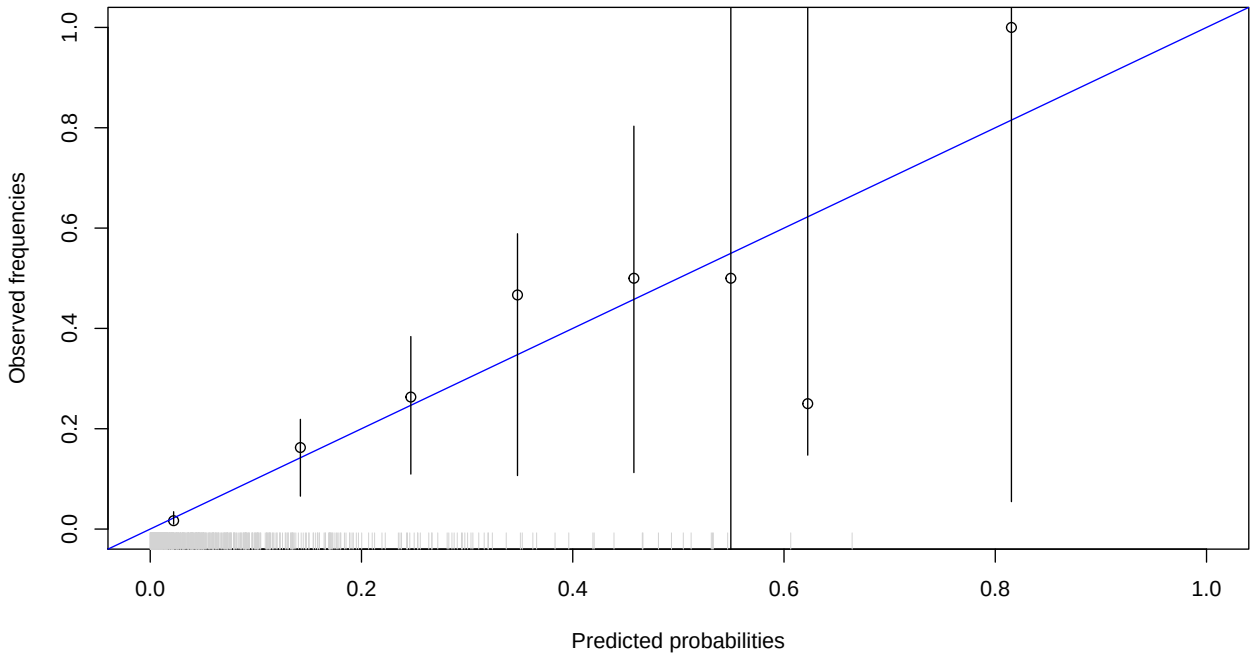
```
signif(sum(diag(confusion.gam))/sum(confusion.gam),3)
```

```
## [1] 0.932
```

- This is better than the `glm` (one more correct prediction) but not as good as the pundit.

GAM calibration

```
f.vs.p <- sapply(0:8/10,frequency.vs.probability,p.increment=0.1,
                 model=gam.ch, events=ch.clean$start)
f.vs.p <- data.frame(frequency=unlist(f.vs.p["frequency",]),
                     ave.prob=unlist(f.vs.p["ave.prob",]),
                     se=unlist(f.vs.p["se",]))
plot(frequency~ave.prob, data=f.vs.p, xlab="Predicted probabilities",
     ylab="Observed frequencies", xlim=c(0,1), ylim=c(0,1))
rug(fitted(lr.ch),col="lightgrey")
abline(0,1,col="blue")
segments(x0=f.vs.p$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)
```



- This is calibrated a bit better, maybe

Testing logistic specifications

- Suppose we want to know whether our logistic regression is "good enough".
- We can try to test the null hypothesis,

$$H_0 : \text{logit generated the data}$$

  against the alternative

$$H_1 : \text{we need a GAM}$$

- With "nice" hypothesis tests, we find a test statistic (like the t-test), and we know it's distribution under the null.

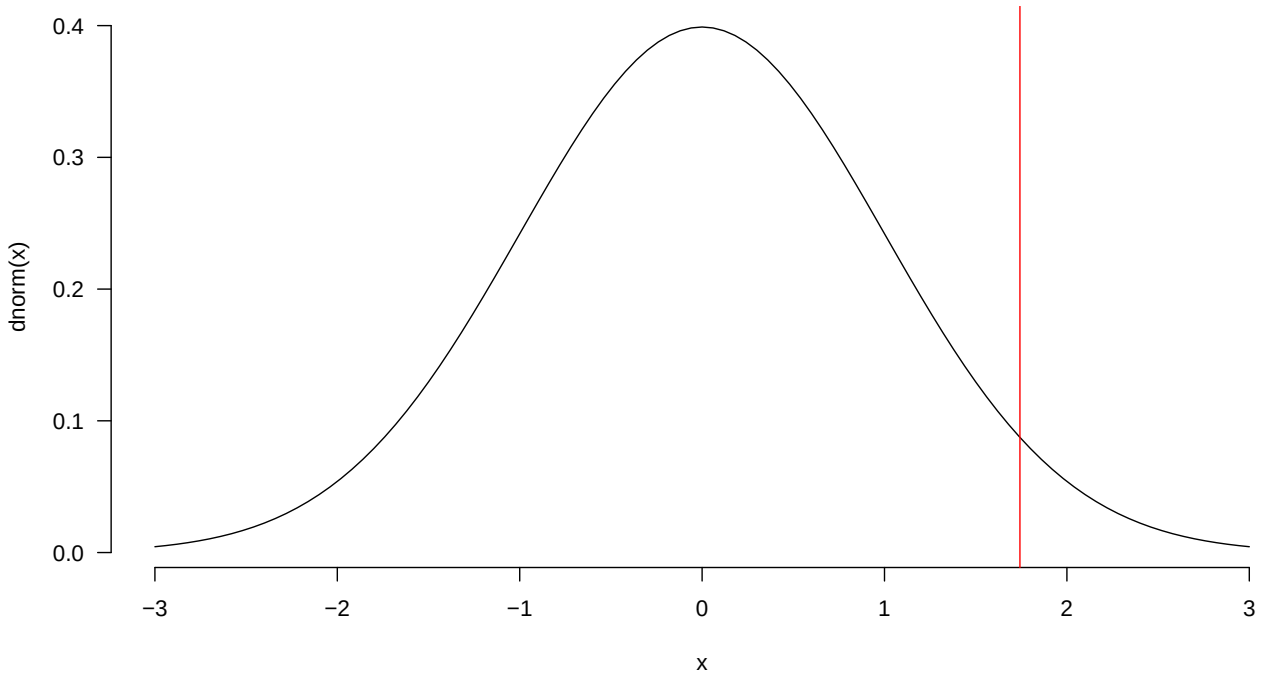
- Then we compare the observed test statistic to that distribution.

Example

- We observe $n = 100$ data points $x_1, \ldots, x_n$
- Null hypothesis: $X_i \sim \mathrm{N}(10, 1)$
- Alternative hypothesis: $X \sim \mathrm{N}(\mu_0, 1)$, $\mu_0 > 10$
- Test statistic: $Z = \sqrt{100}(\bar{X}_{100} - 10)/1 = 10(\bar{X}_{100} - 10)$
- Distribution of test statistic (if the null is true): $Z \sim \mathrm{N}(0, 1)$
- Procedure: calculate z, find the p-value by using the known distribution

Example in R

```
set.seed(04072016)
n = 100
mu = 10
x = rnorm(n, mean=mu+.1, sd=1) # get some data
z = sqrt(n)*(mean(x)-mu) # calculate the test statistic
curve(dnorm(x), -3, 3, las=1, bty='n') # the density of the test statistic under the null
abline(v=z, col=2) # the statistic we observed
```



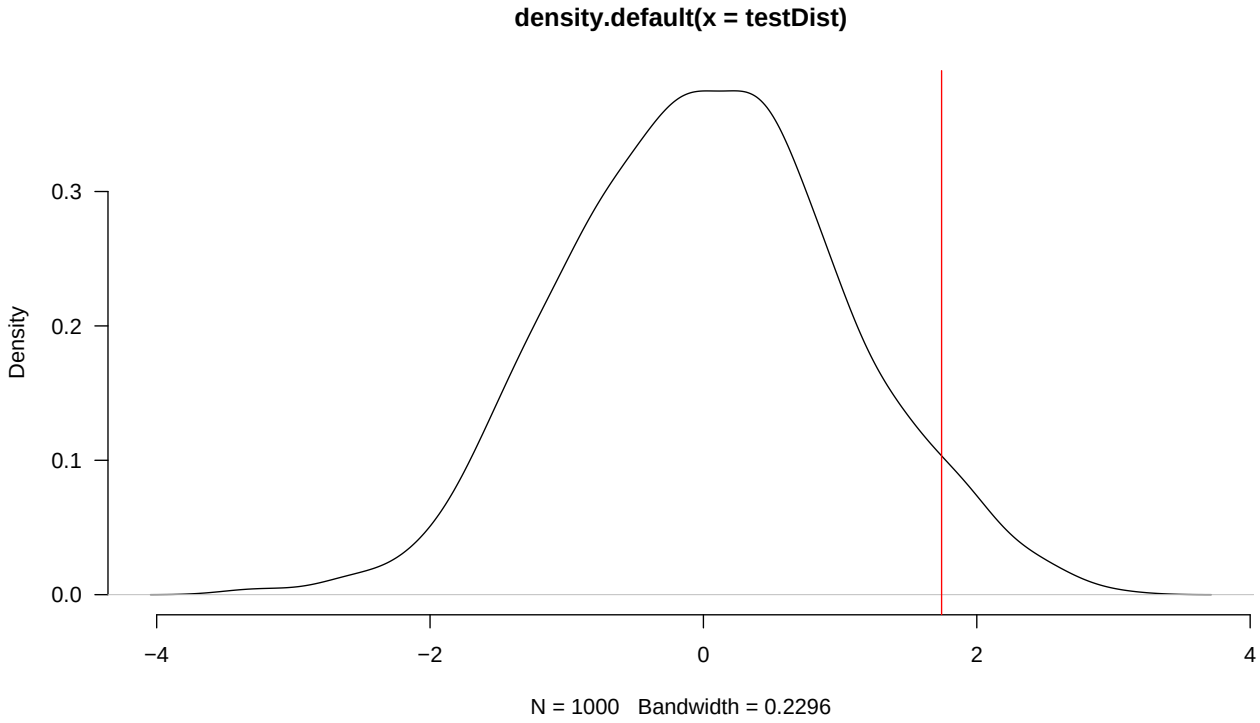

```
(pval = pnorm(z, lower.tail = FALSE)) # the probability of seeing data more extreme
```

```
## [1] 0.04071662
```

Example again

- What if I don't know the distribution of the test statistic?
- I can just generate lots of test statistics from the null hypothesis

```
testStat <- function(n, mu){
  x = rnorm(n, mu) # generate a new data set from the null
  z = sqrt(n)*(mean(x)-mu) # calculate the test statistic
  return(z)
}
testDist = replicate(1000, testStat(n, mu)) # generate 1000 test statistics
plot(density(testDist),las=1, bty='n') # the simulated density of the test statistic
abline(v=z, col=2) # the statistic we observed
```




```
pval2 = mean(testDist>z) # the probability of seeing data more extreme
c(pval, pval2)
```

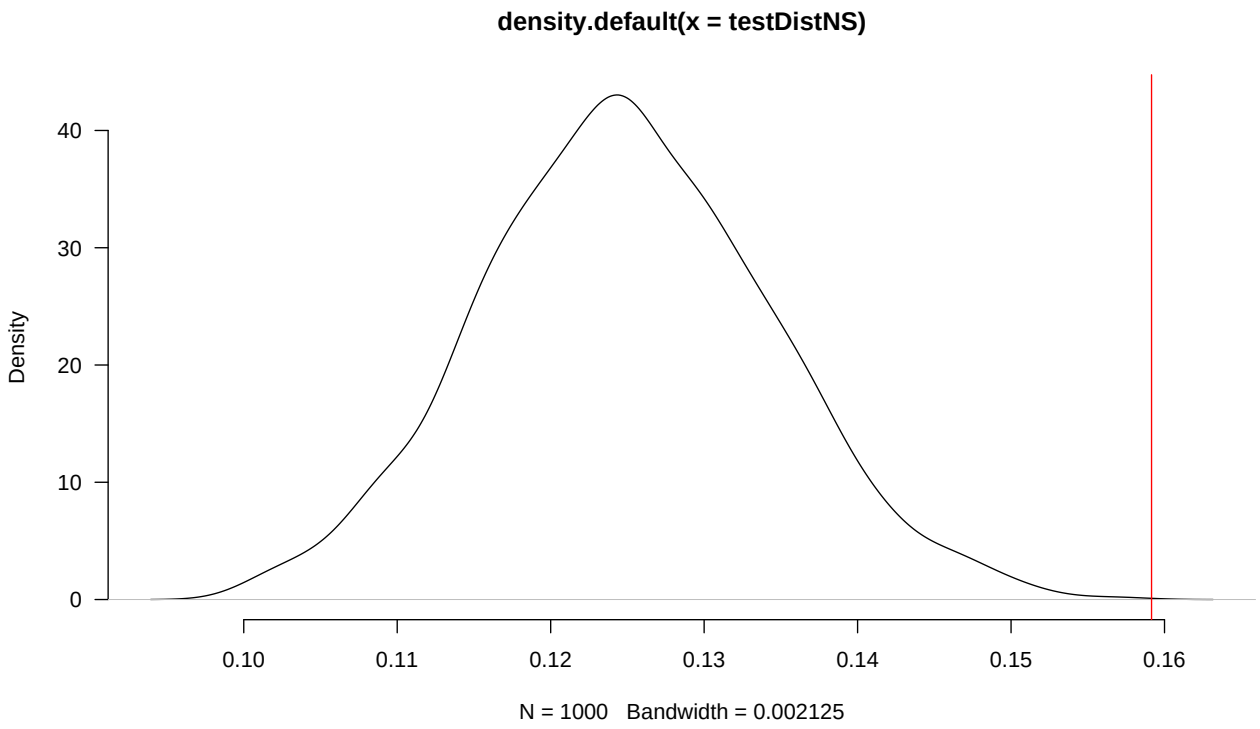
```
## [1] 0.04071662 0.05200000
```

The same idea works for any hypothesis test

- I have a new statistic: $NS = \frac{1}{n}\sum \sin\left(\frac{|x_i - \overline{x}|}{2\pi}\right)$
- I don't know it's distribution. I want to test the null $H_0 : NS = 1/2\pi$ against the alternative $H_1 : NS < 1/2\pi$.

```
testStatNS <- function(n){
  x = rnorm(n)
  z = mean(sin(abs(x-mean(x))/(2*pi)))
  return(z)
```

```
}
testDistNS = replicate(1000, testStatNS(n))
plot(density(testDistNS), las=1, bty='n')
abline(v=1/(2*pi), col=2)
```




```
(pvalNS = 1-mean(testDistNS<1/(2*pi)))
```

```
## [1] 0
```

Logistic regression specification testing

- The appropriate test statistic to look at is the **deviance**
- Deviance is much like using the RSS and doing F-tests.
- The deviance is automatically calculated by `glm` and `gam` using `mod$deviance` where `mod` is the fitted model.

Lower deviance

- Just like comparing RSS in linear regressions, a GAM will always have lower deviance than a logistic regression.
- It has more degrees of freedom and "nests" the logistic regression.
- For nested models, we used F-tests (from `anova`) to decide whether that improvement is significant.
- Then we knew the distribution under the null hypothesis, here we don't.
- So we use our simulation procedure to perform the test.

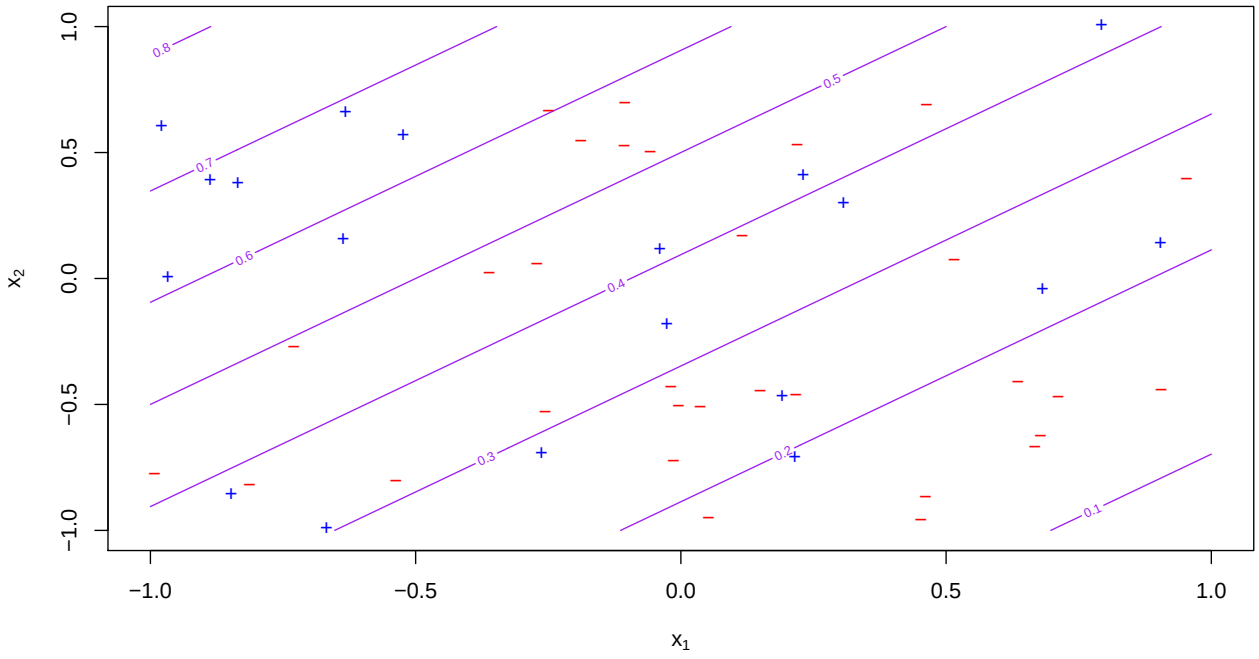
Helper functions

```
simulate.from.logr <- function(df, mdl) { # altered to work with any glm output
  probs <- predict(mdl, type="response") # don't want newdata argument due to factors
  newy <- rbinom(n=nrow(df), size=1,prob=probs)
  df[[names(mdl$model)[1]]] <- newy # the names part, gets the response from the df
  return(df)
}

# Simulate from an estimated logistic model, and refit both the logistic
  # regression and a generalized additive model
# Better code than in the textbook
# Inputs: data frame with covariates (df), fitted logistic model (logr), fitted gam (gamr)
# Output: difference in deviances
delta.deviance.sim <- function (df, logr, gamr) {
  sim.df <- simulate.from.logr(df, logr)
  GLM.dev <- glm(logr$formula,data=sim.df,family="binomial")$deviance # used formulas instead
  GAM.dev <- gam(gamr$formula,data=sim.df,family="binomial")$deviance
  return(GLM.dev - GAM.dev)
}
```

Simulated data

```
require(mgcv)
n = 50
p = 2
set.seed(2018-03-20)
x = matrix(runif(n*p,-1,1), n, p)
df = plot.bilogistic.sim(x, -0.5, c(-1,1))
```



```r
logr = glm(y~X1+X2, data=df, family=binomial) # they are these.
gamr = gam(y~s(X1)+s(X2), data=df, family=binomial)
```

Do the test

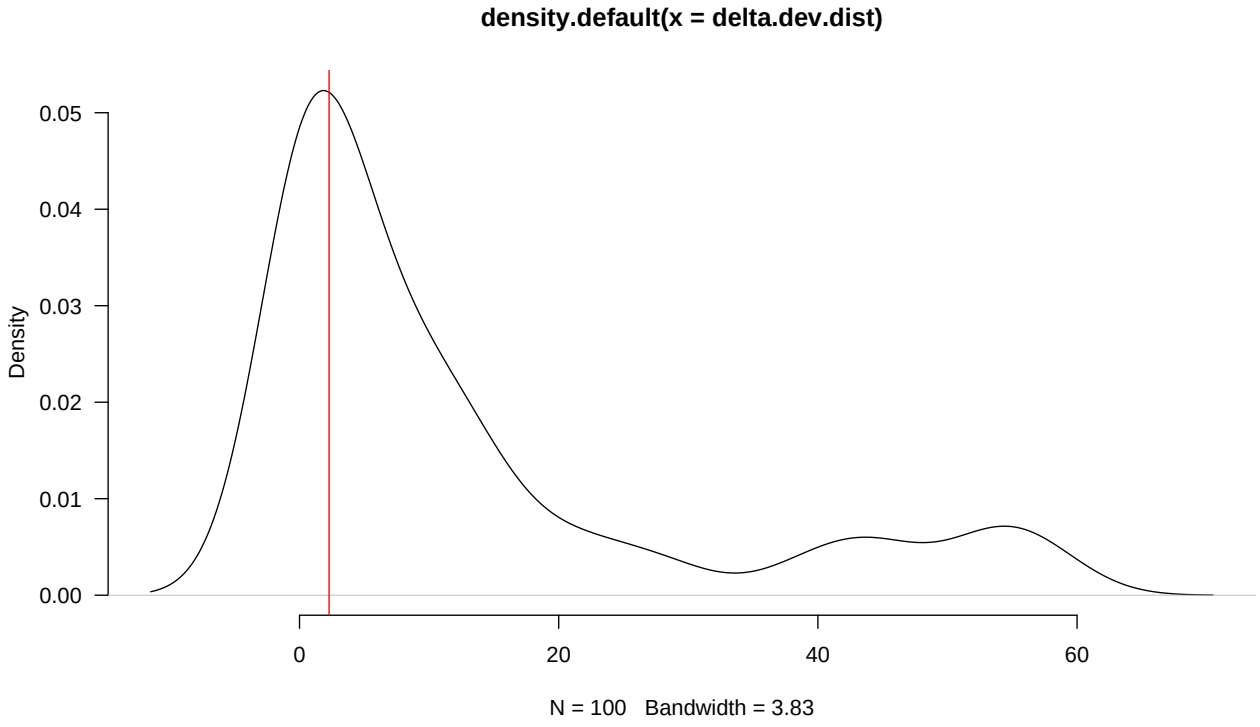
```r
(delta.observed = logr$deviance-gamr$deviance)
```

```
## [1] 2.280674
```

```r
delta.dev.dist = replicate(100, delta.deviance.sim(df, logr, gamr))
mean(delta.observed <= delta.dev.dist) # % of sims that fit better than what we saw
```

```
## [1] 0.61
```

```r
plot(density(delta.dev.dist), las=1, bty='n')
abline(v=delta.observed, col=2)
```

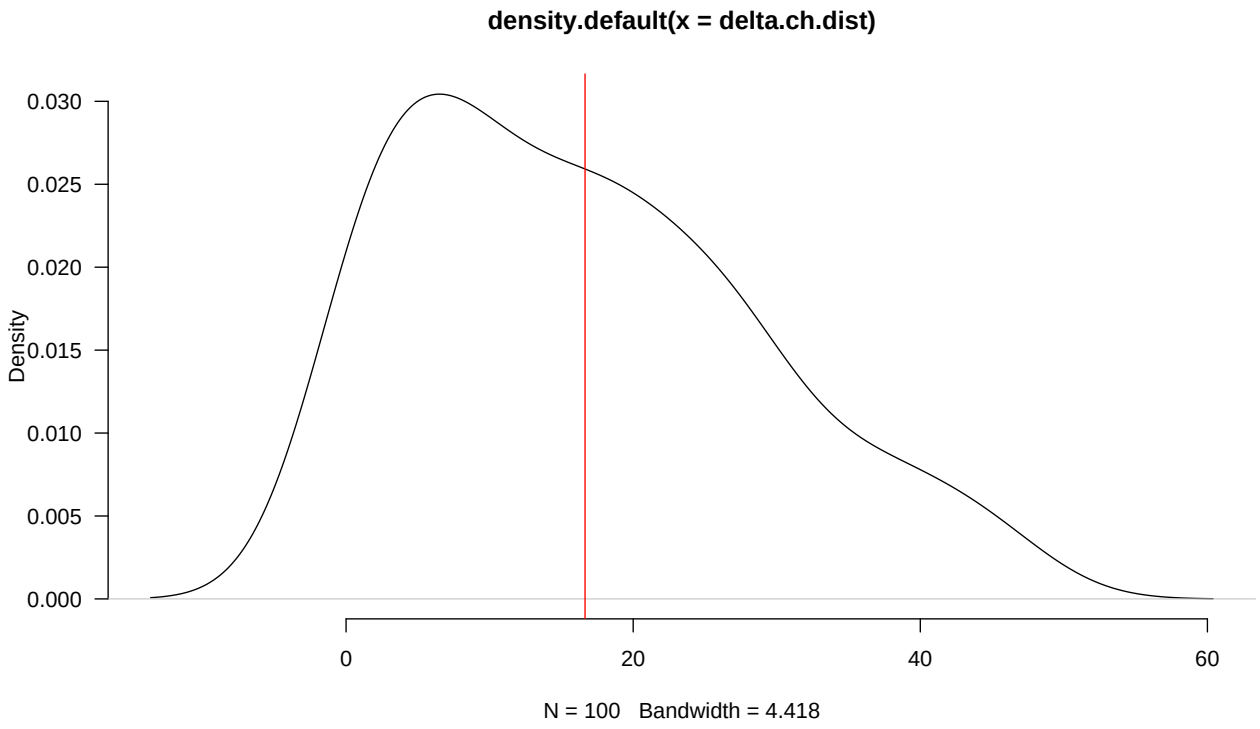


Let's do our data from earlier

```r
obs.ch = lr.ch$deviance - gam.ch$deviance
delta.ch.dist = replicate(100, delta.deviance.sim(ch, lr.ch, gam.ch))
mean(obs.ch <= delta.ch.dist)
```

```
## [1] 0.44
```

```r
plot(density(delta.ch.dist), las=1, bty='n')
abline(v=obs.ch, col=2)
```



CV for GLMs and GAMs

- With logstic regression (or any GLM), the residuals are on the scale of the transformation (by default):

```
head(residuals(logr))
```

```
##          1          2          3          4          5          6
## -0.5967852 -0.6340570  1.5406130 -0.9204301 -0.9966595 -1.0385943
```

- The same is true for the GAM.
- I told you when we did LooCV for `lm` that we could use a trick if the model is "nice" to calculate CV more easily.
- The trick doesn't quite work for GLMs (though it's not a terrible approximation)

```
cv.glm = function(glmObj) mean((residuals(glmObj)/(1-hatvalues(glmObj)))^2) # same as for lm
cv.gam = function(gamObj) mean((residuals(gamObj)/(1-gamObj$hat))^2) # almost the same

(cv.glm(logr))
```

```
## [1] 1.371557
```

```
(cv.gam(gamr))
```

```
## [1] 1.362884
```

- You can use these, though you should recognize that it's not quite right.

Correct CV

- Slight adjustment from code in lecture 6

```r
cv.mdl <- function(data, formulae, fit.function = lm, family=gaussian, nfolds = 5) {
  data <- na.omit(data)
  formulae <- sapply(formulae, as.formula)
  responses <- sapply(formulae, function(form) all.vars(form)[1])
  names(responses) <- as.character(formulae)
  n <- nrow(data)
  fold.labels <- sample(rep(1:nfolds, length.out = n))
  mses <- matrix(NA, nrow = nfolds, ncol = length(formulae))
  colnames <- as.character(formulae)
  for (fold in 1:nfolds) {
    test.rows <- which(fold.labels == fold)
    train <- data[-test.rows, ]
    test <- data[test.rows, ]
    for (form in 1:length(formulae)) {
      current.model <- fit.function(formula = formulae[[form]], data = train,
                                    family=family) #change here
      predictions <- predict(current.model, newdata = test, type='response') # change here
      test.responses <- test[, responses[form]]
      test.errors <- test.responses - predictions
      mses[fold, form] <- mean(test.errors^2)
    }
  }
  return(colMeans(mses))
}
```

```r
(cv.mdl(df, 'y~X1+X2', glm, binomial))
```

```
## [1] 0.2878001
```

```r
(cv.mdl(df, 'y~s(X1)+s(X2)', gam, binomial))
```

```
## [1] 0.3123546
```

```r
(cv.mdl(df, 'y~s(X1,X2)', gam, binomial))
```

```
## [1] 0.5241965
```