Binomial Regression

This is useful when we have dependent variables where the observations are binary (like pass/fail).

The model is still $Y = \beta_0 + \beta_1 X1$ but we make two changes. One is that the model relates to the observations through a transformation called the link function and the other is to use a different distribution than the Gaussian (aka normal). To start, we will use the logit link and the binomial distribution.

The logit is short for log odds.

One way to think about the need for a link function is that the quantity we are modeling is a proportion so it is bounded between the closed interval [0, 1]. In order to achieve this bound we transform the model through the link function. That is, the model we started above for Y (with slope and intercept) is not bounded by 0 or 1, it can range through the entire real line. To bound the model we apply the inverse logit link, which is

$$\frac{\exp(Y)}{1+\exp(Y)}$$

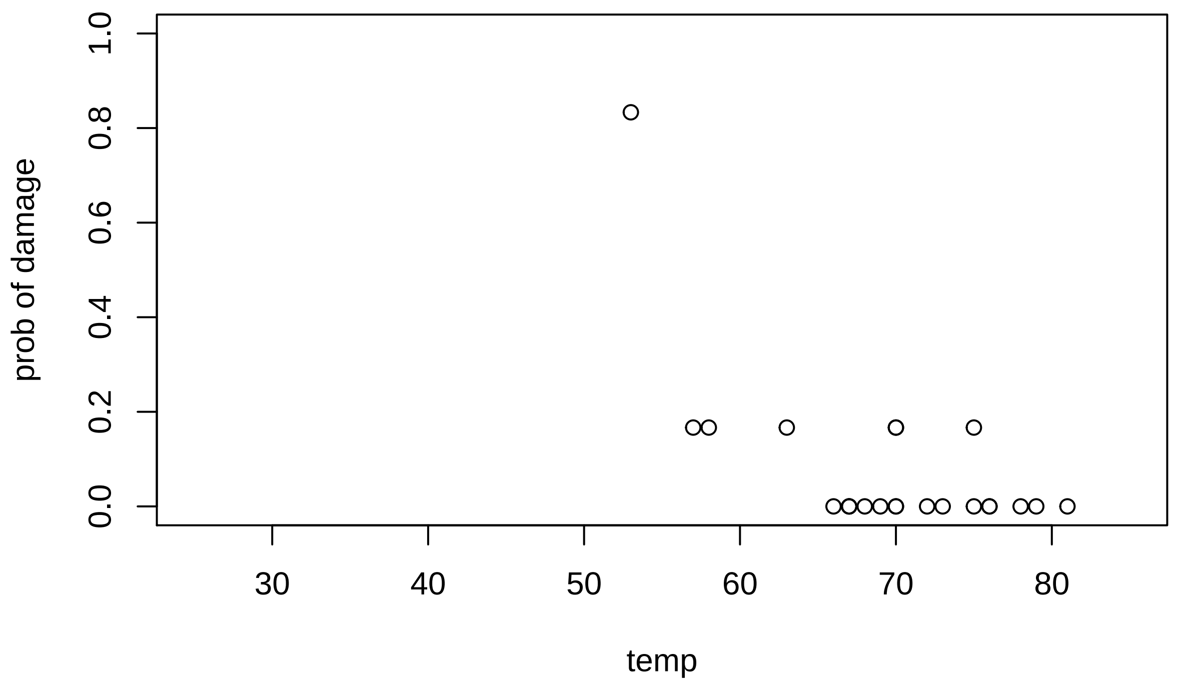
.

Let's do an example. This one comes from a textbook by Julian Faraway and I am using his example data set about the Space Shuttle explosion.

Use the ? interface to get help such as ?orings to get help on the data set orings.

If you get an error message that says Error in library(faraway) : there is no package called 'faraway', then you will need to install the package faraway.

```r
library(faraway)
data(orings)
attach(orings)
plot(damage/6 ~ temp, data=orings, xlim=c(25,85), ylim=c(0,1), xlab="temp", ylab="prob of damage")
```



NOTE: Some of you couldn't download the faraway package. I put the dataset in the repository so you can run the following chunk by changing eval=TRUE and doing knitr. This chunk repeats the commands of the previous chunk but loading the data from the repository rather than the faraway library. I'm leaving this next chunk as eval=FALSE for now.

```
orings <- read.table("../generalized-linear-model/orings.dat",header=T)
attach(orings)
plot(damage/6 ~ temp, data=orings, xlim=c(25,85), ylim=c(0,1), xlab="temp", ylab="prob of damage")
```

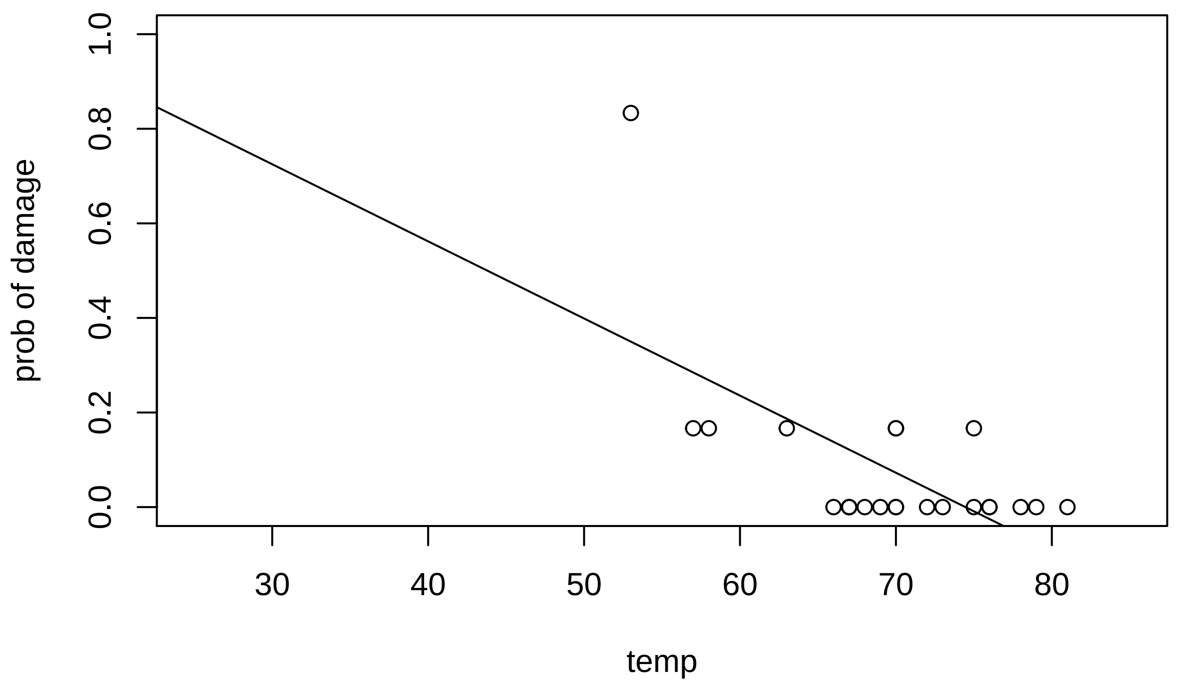
Wrong Way: Regular Regression

Let's see what happens when we do a traditional linear model on these data. This is incorrect as you will see.

```
lmod <- lm(damage/6 ~ temp, orings)
summary(lmod)
```

```
##
## Call:
## lm(formula = damage/6 ~ temp, data = orings)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.1379 -0.1035 -0.0237  0.0660  0.4835
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.21429    0.29993    4.05  0.00058 ***
## temp        -0.01631    0.00429   -3.80  0.00104 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.142 on 21 degrees of freedom
## Multiple R-squared:  0.408,  Adjusted R-squared:  0.379
## F-statistic: 14.5 on 1 and 21 DF,  p-value: 0.00104
```

```
plot(damage/6 ~ temp, orings, xlim=c(25,85), ylim=c(0,1), xlab="temp", ylab="prob of damage")
abline(lmod)
```



```r
predict(lmod)
```

```
##         1         2         3         4         5         6         7
##  0.349881  0.284643  0.268333  0.186786  0.137857  0.121548  0.121548
##         8         9        10        11        12        13        14
##  0.121548  0.105238  0.088929  0.072619  0.072619  0.072619  0.072619
##        15        16        17        18        19        20        21
##  0.040000  0.023690 -0.008929 -0.008929 -0.025238 -0.025238 -0.057857
##        22        23
## -0.074167 -0.106786
```

The model is strange because it yields preditions of negative proportions. It didn't happen in this data set but it is possible for a linear model with Guassian error to predict proportions greater than 1.

Better Approach: Logistic Regression

Now let's run a logistic regression. This uses the R command glm and the binomial family .

```r
datamatrix <- cbind(damage, 6-damage)
logitmod <- glm( datamatrix ~ temp, family=binomial(link="logit"), orings)
summary(logitmod)
```

```
## 
## Call:
## glm(formula = datamatrix ~ temp, family = binomial(link = "logit"), 
##     data = orings)
## 
## Deviance Residuals: 
##    Min      1Q  Median      3Q     Max  
## -0.953  -0.735  -0.439  -0.208   1.956  
```

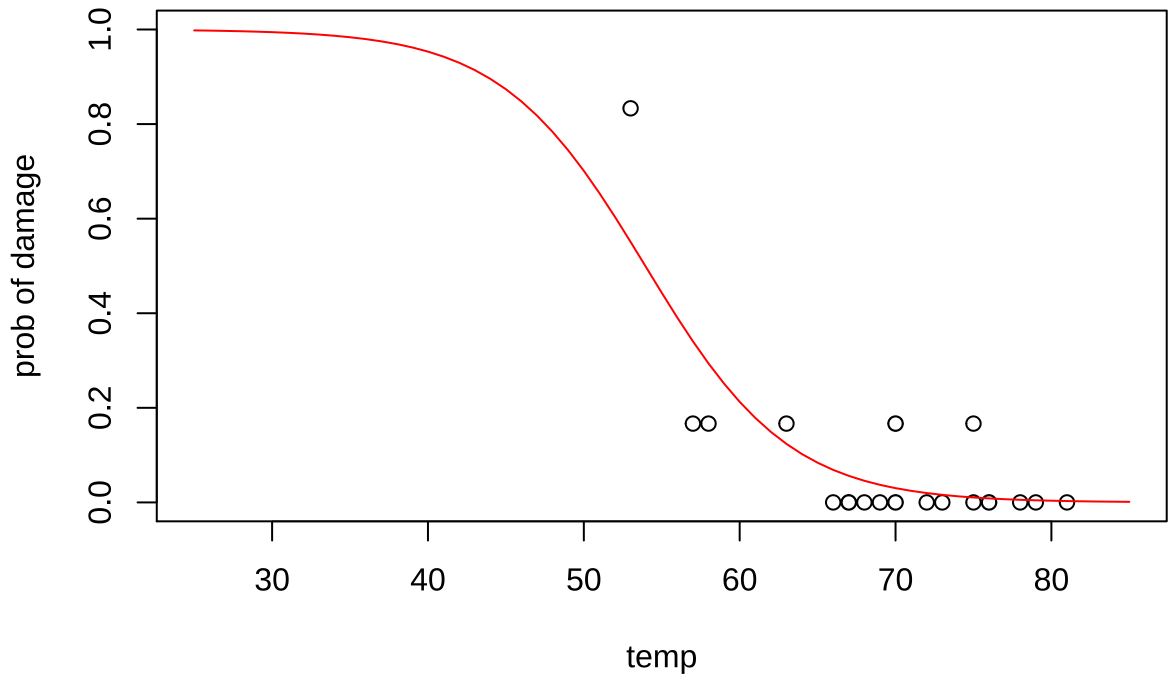
```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  11.6630     3.2963    3.54    4e-04 ***
## temp         -0.2162     0.0532   -4.07  4.8e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 38.898  on 22  degrees of freedom
## Residual deviance: 16.912  on 21  degrees of freedom
## AIC: 33.67
##
## Number of Fisher Scoring iterations: 6
```

```
predict(logitmod)
```

```
##       1       2       3       4       5       6       7       8       9
##  0.2026 -0.6623 -0.8786 -1.9597 -2.6084 -2.8247 -2.8247 -2.8247 -3.0409
##      10      11      12      13      14      15      16      17      18
## -3.2571 -3.4734 -3.4734 -3.4734 -3.4734 -3.9058 -4.1221 -4.5545 -4.5545
##      19      20      21      22      23
## -4.7708 -4.7708 -5.2032 -5.4195 -5.8519
```

Let's add the predicted curve to the previous plot. The red curve is the model we are fitting. It is a curve because of the inverse logit transformation.

```
plot(damage/6 ~ temp, data=orings, xlim=c(25,85), ylim=c(0,1), xlab="temp", ylab="prob of damage")
x.pred <- seq(25,85,1)
lines(x.pred,ilogit(11.663 - .2162*x.pred),col="red")
```



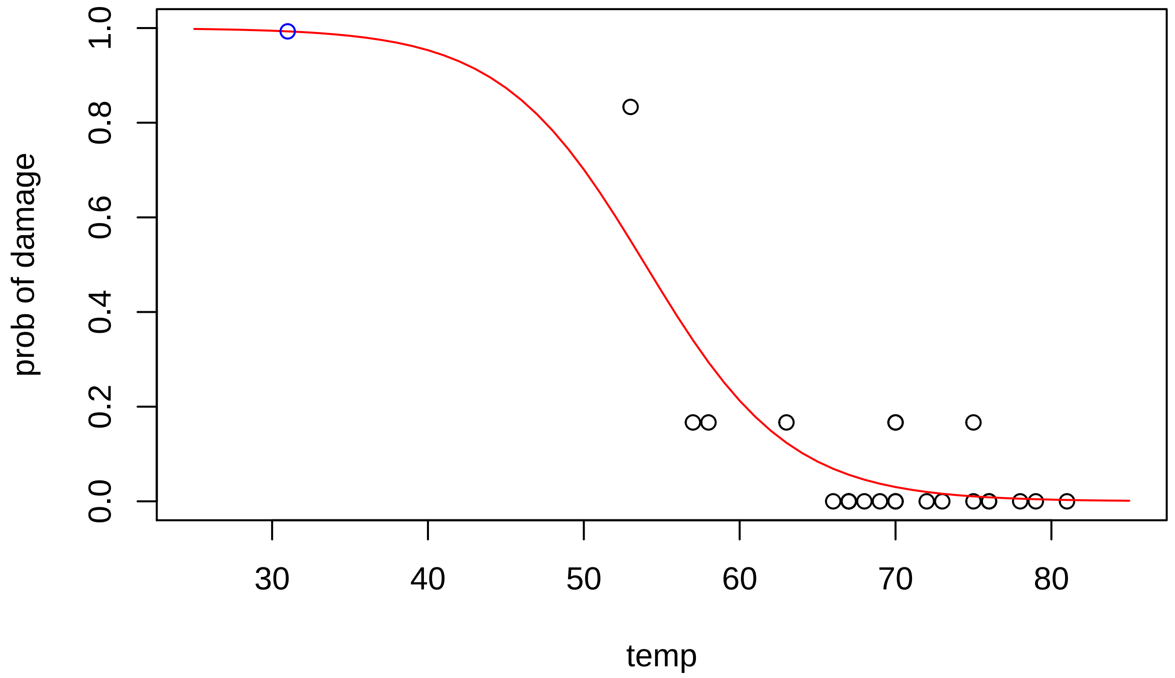
A new feature of Rmarkdown: a chunk can be given a name. I named the previous chunk *plotmodel*.

We can make new predictions as with traditional regression equation.

```
plot(damage/6 ~ temp, data=orings, xlim=c(25,85), ylim=c(0,1), xlab="temp", ylab="prob of damage")
x.pred <- seq(25,85,1)
lines(x.pred,ilogit(11.663 - .2162*x.pred),col="red")
#predict prob at 31 degrees
#plot point on curve
ilogit(11.6630 - .2162*31)
```

```
## [1] 0.993
```

```
#add a blue point to the plot to indicate the new prediction at temperature = 31
points(31, ilogit(11.6630 - .2162*31), col="blue")
```



In the code chunk above I show how to reuse code. It is not good form to cut and paste syntax to redo the plot because if I decide to change something in the plot I have to make changes at multiple places in the code. By recalling a chunk I can reuse code and if I decide to make changes I only have to make a change in one place.

In the previous chunk, I manually computed the predicted score by multiplying 31 by the slope and adding the intercept. If you want, R can do the computation for you. You can use the predict() function and specify the new value of the temperature variable. The new data needs to be in a data.frame with the same variable names as originally used to define the model.

```
predict(logitmod, newdata=data.frame(temp=c(31)))
```

```
##    1
## 4.96
```

```r
#check that equal to hand computation
11.6630 - .2162*31
```

```
## [1] 4.961
```

So that checks out up to roundoff error because I only had four decimal places for the slope and intercept.

Residual deviance

The residual deviance is the sum of squared residuals.

```r
temp.out <- residuals(logitmod)
sum(temp.out^2)
```

```
## [1] 16.91
```

The residual deviance for the Null model is just a simple model with only the intercept.

```r
logitmod.null <- glm( datamatrix ~ 1, family=binomial(link="logit"), orings)
summary(logitmod.null)
```

```
##
## Call:
## glm(formula = datamatrix ~ 1, family = binomial(link = "logit"),
##     data = orings)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -0.998  -0.998  -0.998   0.695   4.478
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -2.446      0.314   -7.78  7.1e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 38.898  on 22  degrees of freedom
## Residual deviance: 38.898  on 22  degrees of freedom
## AIC: 53.66
##
## Number of Fisher Scoring iterations: 4
```

```r
temp.out <- residuals(logitmod.null)
sum(temp.out^2)
```

```
## [1] 38.9
```

Measures of Fit

There are several ways to assess fit of models, unfortunately there is no single best approach to use in all settings.

AIC

A common measures is the AIC, based on information theory. The smaller the AIC, the better the fit. AIC can only be used to compare models to each other, it is relative measure not an absolute measure. The model with the lowest AIC is the best fitting model.

AIC combines the log likelihood of the model and assesses a penalty for the number of parameters. The more parameters you have the model the greater the penalty.

AIC is printed in the output of summary() when you do a glm. You can also extract it from the output of the summary command with the AIC() command.

```
AIC(logitmod)
```

```
## [1] 33.67
```

The computation of AIC emerges from the log likelihood of the maximum likelihood, which is beyond the scope of this course.

$$\text{AIC} = -2 * \text{loglikelihood} + 2 * k$$

where k is the number of parameters.

```
#check this in r, k=2

-2*c(logLik(logitmod)) + 2*2
```

```
## [1] 33.67
```

BIC

A variant of AIC is the BIC, derived from a different framework. Similar to AIC but also includes sample size into the computation. As with AIC, you select the model with the lowest BIC. In R you can extract the BIC from R with the function BIC().

```
BIC(logitmod)
```

```
## [1] 35.95
```

Quick Maximum Likelihood (ML) tutorial

This section uses R to communicate a conceptual point using numerical methods. R is not just for analysis.

The idea of ML is to use the model to set up a likelihood framework that includes both unknown parameters and information about the observed data. One optimizes that likelihood, i.e., solves for the unknown parameters that make the observed data most likely.

There are two ways to do ML. One is numerical and the is analytic.

Numerical

The numeric method sets up the likelihood to optimize. Consider 10 tosses of a fair coin; 7 heads and 3 tails. The likelihood for this is $p^7 * (1-p)^3$. R has an optimization function that minimizes, so we can maximize the likelihood by taking the negative of the likelhood (minimizing a negative is equivalent to maximizating).

I first define the function to optimize, then send it to the optimization function.

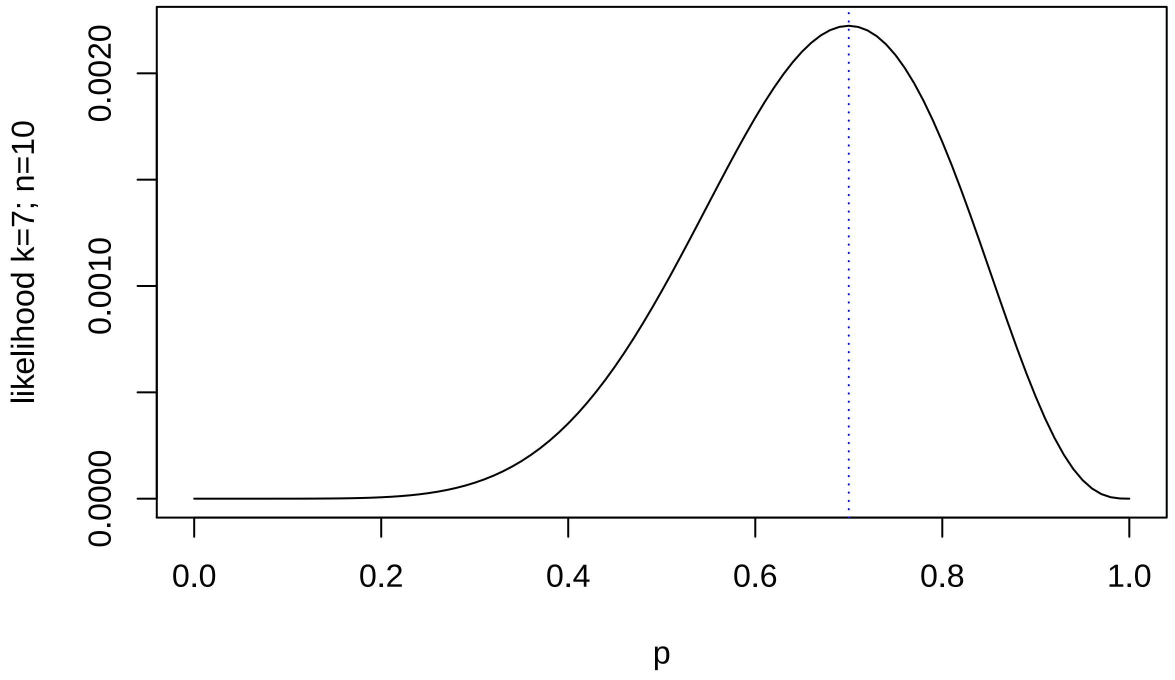
```r
binom <- function(p) {-1*p^7*(1-p)^3}
optimize(binom,c(0,1))
```

```
## $minimum
## [1] 0.7
##
## $objective
## [1] -0.002224
```

The output is .6999, which is very close to the theoretical value of .7.

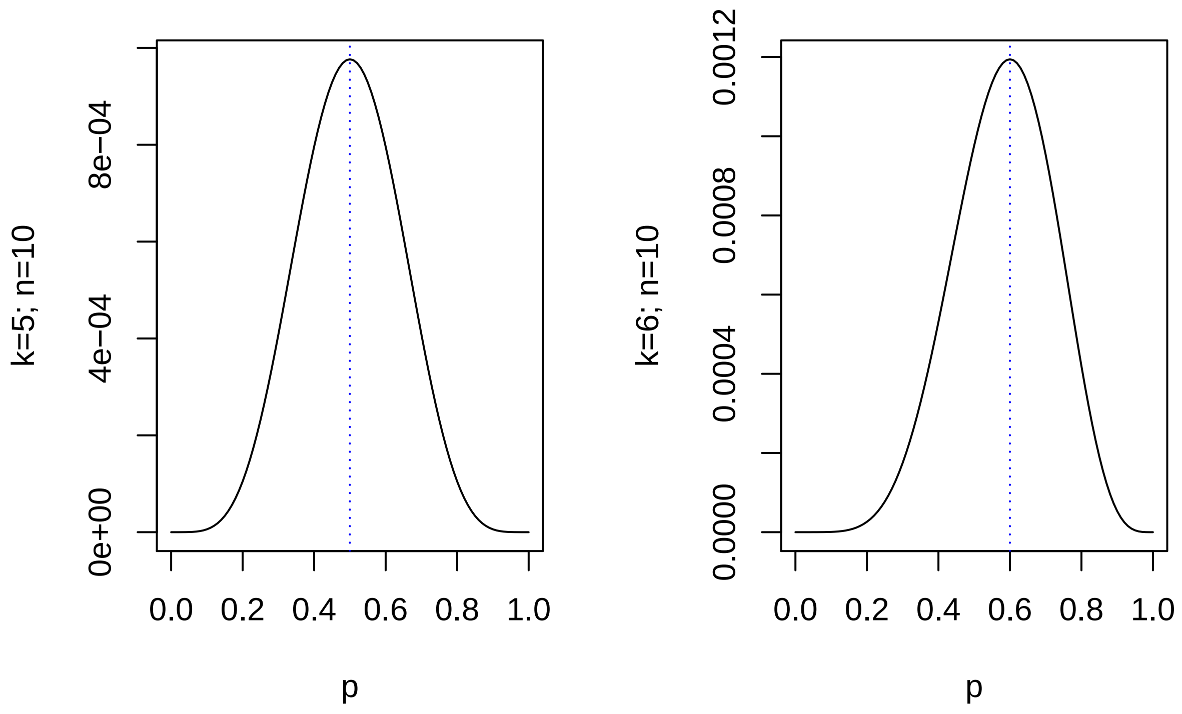
We can plot the likelihood as a function of p to see where the maximum value is located.

```r
#return to likelihood, i.e., not negative likelihood
binom2 <- function(p,k,n) {p^k*(1-p)^(n-k)}
n <- 10
k <- 7
p <- seq(0,1,.01)
plot(p,binom2(p,k,n),type="l",ylab=paste("likelihood k=",k,"; n=",n,sep=""))
abline(v=k/n,lty=3,col="blue")
```



```r
#illustrate for loop, do two plots with k=5,6
#show how to write labels so they take arguments
#from the for loop
```

```
k <- c(5,6)
n <- 10
#setup page with four plots
par(mfcol=c(1,2))
for (i in 1:length(k)) {
  plot(p,binom2(p,k[i],n),type="l",ylab=paste("k=",k[i],"; n=",n,sep=""))
abline(v=k[i]/n,lty=3,col="blue")
}
```



```
#return plot to one per page
par(mfcol=c(1,1))
```

Analytic

The analytic method involves using calculus so I won't go into detail here. Define the likelihood in terms of k successes and (n-k) failures so p^k * (1-p)^(n-k). One takes the first derivative of the likelihood with respect to the unknown parameter p, sets the derivate to 0 and solves for p. The solution will be that p = k/n. Out comes the traditional definition of the proportion.

One can also find variance and standard error of parameters using these methods, either numerical or analtyic.

Sometimes the analytic solution is too difficult or cannot be expressed in closed form so the only option is to solve the problem numerically.

Inverse logit

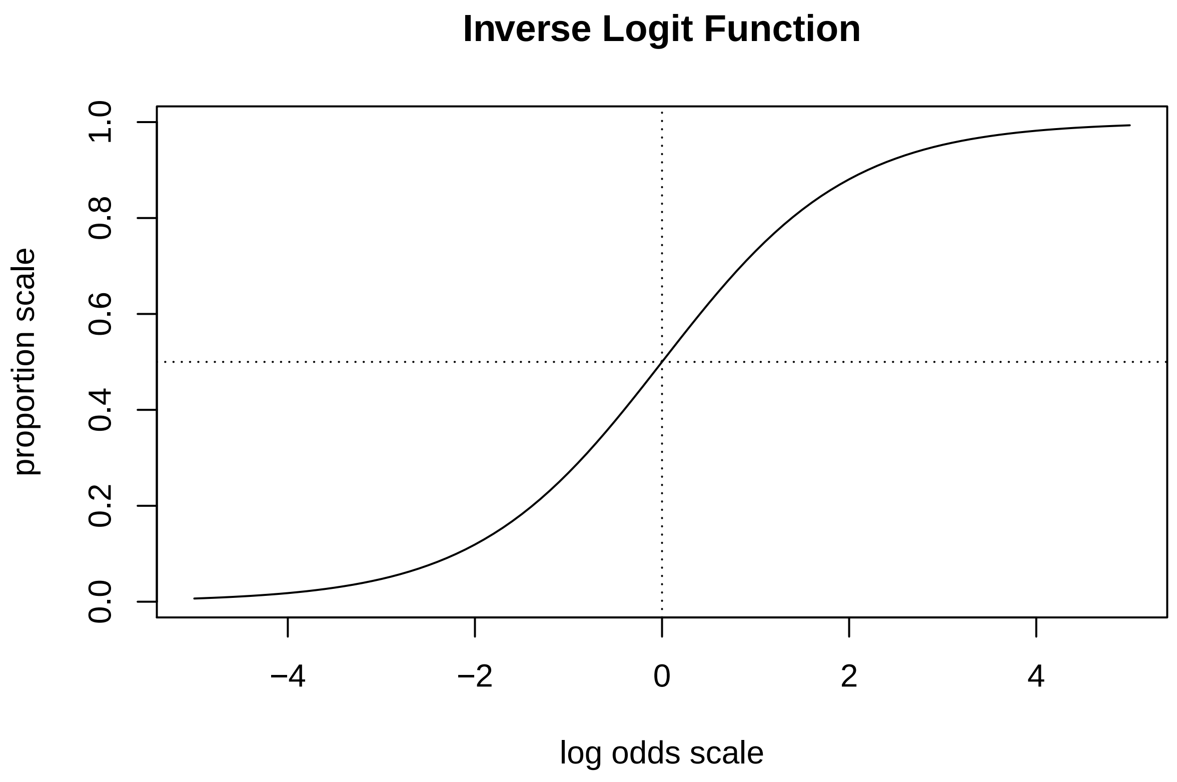
We should take a look at the ilogit function. Within R many functions can be examined by just typing the function name. Some times functions are hidden from view inside a package so you may need to use the triple colon, as in faraway:::ilogit.

Here is the ilogit function inside the package faraway. It has a few lines because it checks for missing data; the key is the last line.

```r
#faraway's ilogit
ilogit <- function (x)
{
    if (any(omit <- is.na(x))) {
        lv <- x
        lv[omit] <- NA
        if (any(!omit))
            lv[!omit] <- Recall(x[!omit])
        return(lv)
    }
    exp(x)/(1 + exp(x))
}
```

Plot the inverse logit function

```r
curve(ilogit, -5, 5, main="Inverse Logit Function",
      xlab="log odds scale", ylab="proportion scale")
# add horizontal line at .5 and vertical at 0
abline(h=.5,lty=3)
abline(v=0,lty=3)
```



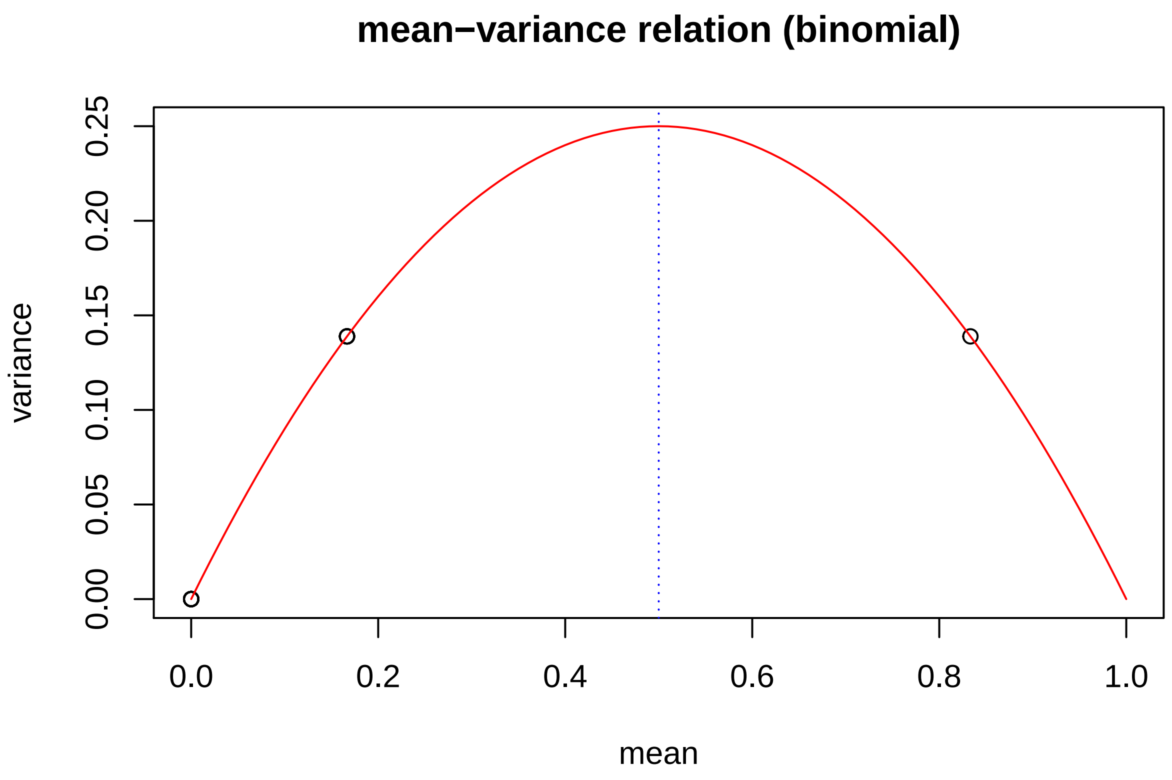
Diagnosing the binomial

Some distributions have a relationship between the mean and the variance. The mean of the binomial is p and the variance of p is $\frac{p*(1-p)}{n}$. So if you know p you know both the the mean and variance. The normal distribution does not have a relationship between the mean and the variance. Other distributions have different relationships that can be used to assess distributions.

Let's plot the mean and variance of the oring data.

There were only three proportions that were observed, 0, 1/6 and 5/6. The plot below shows those three points and superimposes the theoretical curve in red of the relation between the mean and variance.

```
plot(damage/6, (damage/6)* (1-(damage/6)),xlim=c(0,1), ylim=c(0,.25), xlab="mean", ylab="variance",
     main="mean-variance relation (binomial)")
x.seq <- seq(0,1,.01)
lines(x.seq,x.seq*(1-x.seq), col="red")
abline(v=.5,lty=3,col="blue")
```



The maximal variance in the binomial is at .5, denoted by the blue dotted line.

This is a well-chosen example because it is a pure binomial process. Other cases may not be so straightforward such as cases where the are other sources of noise that contaminate the binomial distribution.

Long-Form Data File

In the oring example the data were entered in terms of number of successes/failures out of 6. Take a look at the oring data again to see the structure of the data.

It is possible to have data in a different format where each row represents whether a single oring is a success or failure.

I'll rearrange the data to show this structure. Here I am doing it through lots of coding so you can see what I'm doing. There are simpler ways of doing it but they aren't so transparent.

```
#same results whether data are entered on long form or as binomial counts
y <- c(rep(1,5),rep(0,1),rep(1,1),rep(0,5), rep(1,1),rep(0,5), rep(1,1),rep(0,5), rep(0,36), rep(1,1),r
y
```

```
##   [1] 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
##  [71] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
## [106] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
x <- rep(orings$temp,each=6)
x
```

```
##   [1] 53 53 53 53 53 53 57 57 57 57 57 57 58 58 58 58 58 58 63 63 63 63 63
##  [24] 63 66 66 66 66 66 66 67 67 67 67 67 67 67 67 67 67 67 67 67 67 67 67
##  [47] 67 67 68 68 68 68 68 68 69 69 69 69 69 69 70 70 70 70 70 70 70 70 70
##  [70] 70 70 70 70 70 70 70 70 70 70 70 70 70 70 70 72 72 72 72 72 72 73 73
##  [93] 73 73 73 73 75 75 75 75 75 75 75 75 75 75 75 75 76 76 76 76 76 76 76
## [116] 76 76 76 76 76 78 78 78 78 78 78 79 79 79 79 79 79 81 81 81 81 81 81
```

```
longdata <- data.frame(x,y)

logitmod.long <- glm(y~x,family=binomial(link="logit"))
summary(logitmod.long)
```

```
##
## Call:
## glm(formula = y ~ x, family = binomial(link = "logit"))
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.265  -0.340  -0.247  -0.130   3.022
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  11.6630     3.2962    3.54    4e-04 ***
## x            -0.2162     0.0532   -4.07  4.8e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 76.745  on 137  degrees of freedom
## Residual deviance: 54.759  on 136  degrees of freedom
## AIC: 58.76
##
## Number of Fisher Scoring iterations: 6
```

This produces the same deviance as the original logit model we ran with the data in the form of counts out of 6.

```r
anova(logitmod.long)
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: y
##
## Terms added sequentially (first to last)
##
##
##      Df Deviance Resid. Df Resid. Dev
## NULL                   137       76.7
## x     1       22       136       54.8
```

More complicated regression models

We can have any complicated model on the regression. For example, we can add a quadratic term to the regression.

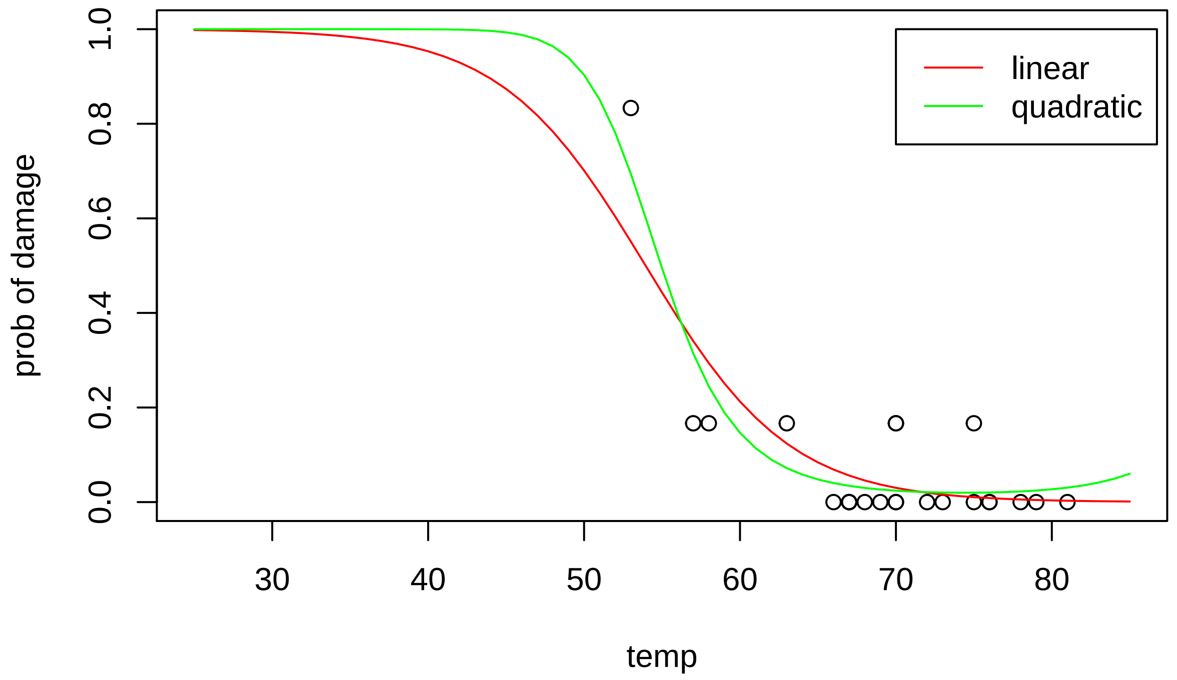
I'll reuse the plotmodel chunk so I can superimpose the new predicted curve for the quadratic fit on the previous plot with the linear fit (i.e., only a single X as a predictor).

It is possible to include transformations of variables, such as squaring, by using the I() inside a formula. This means "as is" so the formula is interpreting the "^" as a transformation.

```r
glmodsq <- glm(y ~ x + I(x^2), family=binomial)

#reuse previous plot
plot(damage/6 ~ temp, data=orings, xlim=c(25,85), ylim=c(0,1), xlab="temp", ylab="prob of damage")
x.pred <- seq(25,85,1)
lines(x.pred,ilogit(11.663 - .2162*x.pred),col="red")
predictorvar <- seq(25,85,1)
lines(predictorvar,ilogit(53.09 - 1.5316*predictorvar + .01029*predictorvar^2), col="green")

#add legend to plot
legend(70,1,legend=c("linear","quadratic"),lty=c(1,1),
       col=c("red","green"))
```



Compare Fit of Linear and Quadratic Models

```r
anova(logitmod.long, glmodsq, test="LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x
## Model 2: y ~ x + I(x^2)
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       136       54.8
## 2       135       52.4  1     2.39     0.12
```

Best Approach: generalized linear mixed model

We can improve the generalized linear model we ran above by taking into account that observations in this example come in clusters or group of six. The study examined cases with six orings at a time, so in a sense these are repeated measures but we didn't take that into account. The generalized linear *mixed* model (GLMM) allows us to model such clustering; it is generalized linear model that includes random effects.

The packages lme4 and nlme both can do generalized linear mixed models.

In terms of the structural model, the GLMM is written as

$$Y = \beta_0 + \beta_1 X1 + \beta_2 X2 + \pi + \epsilon$$

where π is normally distributed N(0,v(π)) and ϵ is normally distributed N(0, v(ϵ))

This model differs from a GLM because it includes π, which is another source of variance.

In the oring example you can think of it as a random intercept that gives each oring cluster of 6 a unique intercept. Just rearrange the terms

$$Y = (\beta_0 + \pi) + \beta_1 X1 + \beta_2 X2 + \epsilon$$

The term inside the parentheses is the total intercept: β_0 applies to all orings and the term π applies uniquely to a cluster of six. I omit subscripts to avoid getting things cluttered.

```
library(lme4)
```

```
## Warning: package 'lme4' was built under R version 3.1.1

## Loading required package: Matrix
## Loading required package: Rcpp
```

```
#create a  clustering variable for the 23 orings
n <- 23
clustering <- factor(rep(1:n,each=6))

longdata$clustering <- clustering

output.glmer <- glmer(y ~ x + (1|clustering), family=binomial, data=longdata)
summary(output.glmer)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: y ~ x + (1 | clustering)
##    Data: longdata
##
##      AIC      BIC   logLik deviance df.resid
##     60.8     69.5    -27.4     54.8      135
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -1.107 -0.244 -0.176 -0.092  9.750
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  clustering (Intercept) 4.41e-15 6.64e-08
## Number of obs: 138, groups:  clustering, 23
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  11.6630     3.2969    3.54    4e-04 ***
## x            -0.2162     0.0532   -4.07  4.8e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##   (Intr)
## x -0.994
```

In this example the random effect component was very small, essentially negligable. We will talk more about why this is the case when we cover random effects in more detail and see examples where the random effect is not negligable.

Other Links: The probit example

The probit is another link function that can be used with logistic regression. All you do is change "logit" to "probit" in the call to glm().

```
datamatrix <- cbind(damage, 6-damage)
probitmod <- glm( datamatrix ~ temp, family=binomial(link="probit"), orings)
summary(probitmod)
```

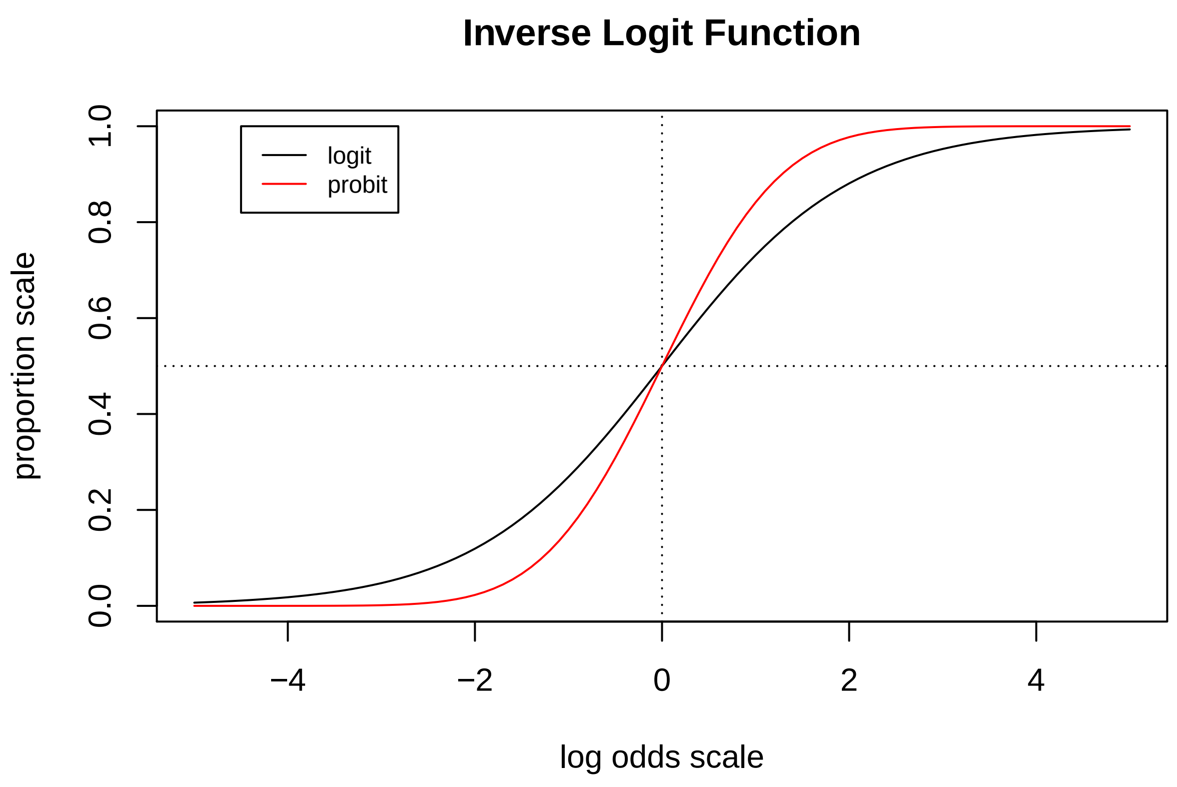
```
##
## Call:
## glm(formula = datamatrix ~ temp, family = binomial(link = "probit"),
##     data = orings)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.013  -0.776  -0.447  -0.158   1.998
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   5.5915     1.7105    3.27   0.0011 **
## temp         -0.1058     0.0266   -3.98  6.8e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 38.898  on 22  degrees of freedom
## Residual deviance: 18.131  on 21  degrees of freedom
## AIC: 34.89
##
## Number of Fisher Scoring iterations: 6
```

```
predict(probitmod)
```

```
##        1        2        3        4        5        6        7        8
## -0.01615 -0.43937 -0.54517 -1.07419 -1.39160 -1.49741 -1.49741 -1.49741
##        9       10       11       12       13       14       15       16
## -1.60321 -1.70901 -1.81482 -1.81482 -1.81482 -1.81482 -2.02642 -2.13223
##       17       18       19       20       21       22       23
## -2.34384 -2.34384 -2.44964 -2.44964 -2.66125 -2.76705 -2.97866
```

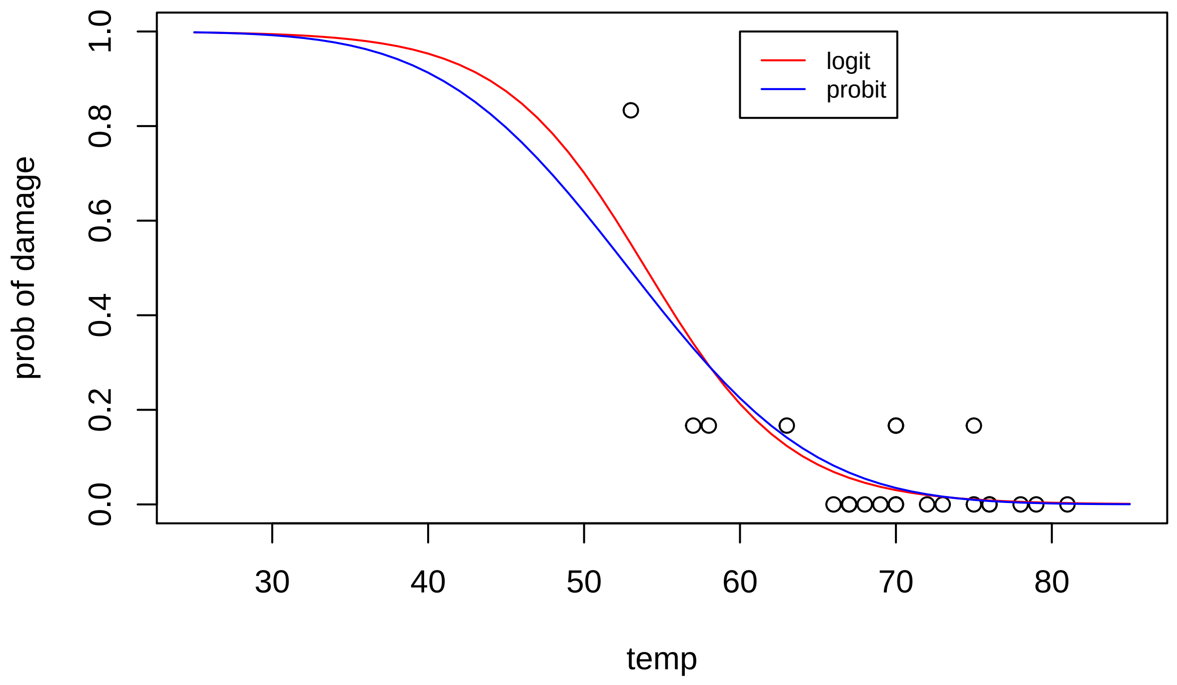
The inverse of the probit is the inverse cumulative normal distribution. It looks similar to the inverse logit. Here I plot both theoretical curves.

```
curve(ilogit, -5, 5, main="Inverse Logit Function",
      xlab="log odds scale", ylab="proportion scale")
# add horizontal line at .5 and vertical at 0
abline(h=.5,lty=3)
abline(v=0,lty=3)
curve(pnorm,-5,5,add=T,col="red")
legend(-4.5,1, legend=c("logit","probit"),lty=c(1,1),
       col=c("black","red"),cex=.75)
```



Here is the predicted curve for the probit superimposed on the previous graph we saw for the logit (note reuse of the <> chunk so I don't have to retype that plot).

```
plot(damage/6 ~ temp, data=orings, xlim=c(25,85), ylim=c(0,1), xlab="temp", ylab="prob of damage")
x.pred <- seq(25,85,1)
lines(x.pred,ilogit(11.663 - .2162*x.pred),col="red")
lines(x.pred,pnorm(5.5915 - .1058*x.pred),col="blue")
legend(60,1,legend=c("logit","probit"),lty=c(1,1),col=c("red","blue"),cex=.75)
```



The two curves are very similar. They differ mostly in the region where there is little data (roughly between temp 40 and 50). Where there is more data they agree more closely.

Difference between logit and probit links

Both accomplish the same thing of converting the real numbers (positive and negative) onto the closed interval [0,1]. In many cases the findings are almost indistinguishable, as we saw in the previous example.

However, they do imply different processes and vary in computational difficulty for more complex models. This can be illustrated in the context of choice models. People are given choices between various objects (products, snacks, drinks) and their choices (select one) are recorded. So these data can be modeled much like the oring data, as a series of yes/no observations.

The logit approach implies a restrictive independence of the number of options constraint. If the choice proportion for one item is greater than another, say $p_i > p_j$ for two products i and j, then that inequality remains regardless of any other products. If you like coke over pepsi, you will always select coke over pepsi regardless of what other drinks are available. So it doesn't permit switches where if tea is available, now you prefer pepsi to coke. Technically, the condition is more complicated because it is the ratio of p_i and p_j that is preserved regardless of the other options. Sometimes it doesn't make sense to impose that condition in the model, so that elimates the logit from consideration as an analytic approach.

The probit has a different feature: it allows correlations across the choice options unlike the logit. It may be the case that if you select one option you are likely to also select another (e.g., if you buy cheese you may be likely to also buy bread). Those kinds of interdependencies between the choice options cannot be modeled with the logit (recall, the logit implies a type of independence of options) but can be modeled (sometimes with heavy computational work) with the probit.

Unfortunately, there aren't easy approaches to diagnose which fits data better. The AIC and BIC can be used because they do not require models to be nested.

```
AIC(logitmod)
```

```
## [1] 33.67
```

```
AIC(probitmod)
```

```
## [1] 34.89
```

Bigger Picture

There is a broad family of stastical models under the generalized linear model framework.

The "linear" refers to an additive model such as $\beta_0 + \beta_1 X_1 + \beta_2 X_2$. The generalized refers to a general class of distributions that include the normal, the binomial, the Poisson, the negative binomial, gamma and some others.

The link relates the scale of the model to the scale of the observations. For instance, we saw above that the scale of the model for binomial data is log odds so we use the inverse logit to convert log odds to the observed scale of proportions.

Distribution	Links
normal	identity

Distribution	Links
binomial	logit, probit, log-log
Poisson	log

Extra material (not included in html)