Logistic regression (with R)

Christopher Manning

4 November 2007

1 Theory

We can transform the output of a linear regression to be suitable for probabilities by using a logit link function on the lhs as follows:

$$\operatorname{logit} p = \log o = \log \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{1}$$

The odds can vary on a scale of $(0, \infty)$, so the log odds can vary on the scale of $(-\infty, \infty)$ – precisely what we get from the rhs of the linear model. For a real-valued explanatory variable x_i, the intuition here is that a unit additive change in the value of the variable should change the odds by a constant multiplicative amount.

Exponentiating, this is equivalent to:[1]

$$e^{\operatorname{logit} p} = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_k x_k} \tag{2}$$

$$o = \frac{p}{1-p} = e^{\beta_0} e^{\beta_1 x_1} e^{\beta_2 x_2} \cdots e^{\beta_k x_k} \tag{3}$$

The inverse of the logit function is the logistic function. If $\operatorname{logit}(\pi) = z$, then

$$\pi = \frac{e^z}{1+e^z}$$

The logistic function will map any value of the right hand side (z) to a proportion value between 0 and 1, as shown in figure 1.

Note a common case with categorical data: If our explanatory variables x_i are all binary, then for the ones that are false (0), we get $e^0 = 1$ and the term disappears. Similarly, if $x_i = 1$, $e^{\beta_i x_i} = e^{\beta_i}$. So we are left with terms for only the x_i that are true (1). For instance, if $x_3, x_4, x_7 = 1$ only, we have:

$$\operatorname{logit} p = \beta_0 + \beta_3 + \beta_4 + \beta_7 \tag{4}$$

$$o = e^{\beta_0} e^{\beta_3} e^{\beta_4} e^{\beta_7} \tag{5}$$

The intuition here is that if I know that a certain fact is true of a data point, then that will produce a constant change in the odds of the outcome ("If he's European, that doubles the odds that he smokes").

Let $L = L(D; B)$ be the likelihood of the data D given the model, where $B = \{\beta_0, \ldots, \beta_k\}$ are the parameters of the model. The parameters are estimated by the principle of maximum likelihood. Technical point: there is no error term in a logistic regression, unlike in linear regressions.

[1] Note that we can convert freely between a probability p and odds o for an event versus its complement:

$$o = \frac{p}{1-p} \qquad\qquad p = \frac{o}{o+1}$$

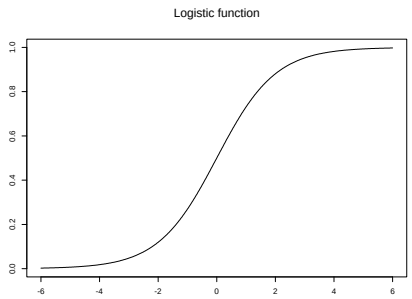


Figure 1: The logistic function

2 Basic R logistic regression models

We will illustrate with the Cedegren dataset on the website.

```
cedegren <- read.table("cedegren.txt", header=T)
```

You need to create a two-column matrix of success/failure counts for your response variable. You *cannot* just use percentages. (You can give percentages but then weight them by a count of success + failures.)

```
attach(cedegren)
ced.del <- cbind(sDel, sNoDel)
```

Make the logistic regression model. The shorter second form is equivalent to the first, but *don't* omit specifying the family.

```
ced.logr <- glm(ced.del ~ cat + follows + factor(class), family=binomial("logit"))
ced.logr <- glm(ced.del ~ cat + follows + factor(class), family=binomial)
```

The output in more and less detail:

```
> ced.logr

Call:  glm(formula = ced.del ~ cat + follows + factor(class), family = binomial("logit"))

Coefficients:
   (Intercept)            catd            catm            catn            catv        followsP
       -1.3183         -0.1693          0.1786          0.6667         -0.7675          0.9525
      followsV  factor(class)2  factor(class)3  factor(class)4
        0.5341          1.2704          1.0480          1.3742

Degrees of Freedom: 51 Total (i.e. Null);  42 Residual
Null Deviance:     958.7
Residual Deviance: 198.6  AIC: 446.1

> summary(ced.logr)

Call:
glm(formula = ced.del ~ cat + follows + factor(class), family = binomial("logit"))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
```

```
-3.24384  -1.34325   0.04954   1.01488   6.40094

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -1.31827    0.12221 -10.787  < 2e-16
catd           -0.16931    0.10032  -1.688 0.091459
catm            0.17858    0.08952   1.995 0.046053
catn            0.66672    0.09651   6.908 4.91e-12
catv           -0.76754    0.21844  -3.514 0.000442
followsP        0.95255    0.07400  12.872  < 2e-16
followsV        0.53408    0.05660   9.436  < 2e-16
factor(class)2  1.27045    0.10320  12.310  < 2e-16
factor(class)3  1.04805    0.10355  10.122  < 2e-16
factor(class)4  1.37425    0.10155  13.532  < 2e-16

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 958.66  on 51  degrees of freedom
Residual deviance: 198.63  on 42  degrees of freedom
AIC: 446.10

Number of Fisher Scoring iterations: 4
```

Residual deviance is the difference in $G^2 = -2 \log L$ between a maximal model that has a separate parameter for each cell in the model and the built model. Changes in the deviance (the difference in the quantity $-2 \log L$) for two models which can be nested in a reduction will be approximately χ^2-distributed with dof equal to the change in the number of estimated parameters. Thus the difference in deviances can be tested against the χ^2 distribution for significance. The same concerns about this approximation being valid only for reasonably sized expected counts (as with contingency tables and multinomials in Suppes (1970)) still apply here, but we (and most people) ignore this caution and use the statistic as a rough indicator when exploring to find good models.

We're usually mainly interested in the relative goodness of models, but nevertheless, the high residual deviance shows that the model cannot be accepted to have been likely to generate the data (`pchisq(198.63, 42)`≈ 1). However, it certainly fits the data better than the null model (which means that a fixed mean probability of deletion is used for all cells): `pchisq(958.66-198.63, 9)`≈ 1.

What can we see from the parameters of this model? catd and catm have different effects, but both are not very clearly significantly different from the effect of cata (the default value). All following environments seem distinctive. For class, all of class 2–4 seem to have somewhat similar effects, and we might model class as a two way distinction. It seems like we cannot profitably drop a whole factor, but we can test that with the `anova` function to give an analysis of deviance table, or the `drop1` function to try dropping each factor:

```
> anova(ced.logr, test="Chisq")
Analysis of Deviance Table

Model: binomial, link: logit

Response: ced.del

Terms added sequentially (first to last)

             Df Deviance Resid. Df Resid. Dev P(>|Chi|)
NULL                            51     958.66
cat           4   314.88        47     643.79 6.690e-67
```

```
follows        2   228.86        45     414.93 2.011e-50
factor(class)  3   216.30        42     198.63 1.266e-46
> drop1(ced.logr, test="Chisq")
Single term deletions

Model:
ced.del ~ cat + follows + factor(class)
              Df Deviance    AIC    LRT   Pr(Chi)
<none>             198.63 446.10
cat            4   368.76 608.23 170.13 < 2.2e-16
follows        2   424.53 668.00 225.91 < 2.2e-16
factor(class)  3   414.93 656.39 216.30 < 2.2e-16
```

The ANOVA test tries adding the factors only in the order given in the model formula (left to right). If things are close, you should try rearranging the model formula ordering, or using `drop1`, but given the huge drops in deviance, here, it seems clearly unnecessary.

Let's now try a couple of models by collapsing particular levels, based on our observations above.

```
> glm(ced.del ~ cat + follows + I(class == 1), family=binomial("logit"))

Call:  glm(formula = ced.del ~ cat + follows + I(class == 1), family = binomial("logit"))

Coefficients:
      (Intercept)              catd              catm              catn              catv
          -0.0757           -0.1614            0.1876            0.6710           -0.7508
        followsP          followsV  I(class == 1)TRUE
           0.9509            0.5195           -1.2452

Degrees of Freedom: 51 Total (i.e. Null);  44 Residual
Null Deviance:     958.7
Residual Deviance: 232.7  AIC: 476.2
> pchisq(232.72-198.63, 2)
[1] 1
```

The above model isn't as good. We could just collapse class 2 and 4. Formally that model can't be rejected as fitting the data as well as the higher parameter model at a 95% confidence threshold:

```
> glm(ced.del ~ cat + follows + I(class == 1) + I(class==3), family=binomial("logit"))

Call:  glm(formula = ced.del ~ cat + follows + I(class == 1) + I(class ==
      3), family = binomial("logit"))

Coefficients:
      (Intercept)              catd              catm              catn              catv
         0.009838         -0.170717          0.181871          0.666142         -0.789446
        followsP          followsV  I(class == 1)TRUE  I(class == 3)TRUE
         0.957487          0.529646         -1.328227         -0.280901

Degrees of Freedom: 51 Total (i.e. Null);  43 Residual
Null Deviance:     958.7
Residual Deviance: 202.1  AIC: 447.5
> pchisq(202.08-198.63, 1)
[1] 0.9367482
```

However, in terms of our model, where class is a scale, collapsing together classes 2 and 4 seems rather dubious, and should be put aside.

But it does seem reasonable to collapse together some of the word classes. a and d are certainly a natural class of noun modifiers, and it's perhaps not unreasonable to group those with m. Let's try those models. This one is worse:

```
> glm(ced.del ~ I(cat=="n") + I(cat=="v") + follows + factor(class), family=binomial("logit"))

Call:  glm(formula = ced.del ~ I(cat == "n") + I(cat == "v") + follows +
      factor(class), family = binomial("logit"))

Coefficients:
      (Intercept)  I(cat == "n")TRUE  I(cat == "v")TRUE           followsP           followsV
          -1.2699             0.5750            -0.8559             1.0133             0.5771
   factor(class)2     factor(class)3     factor(class)4
           1.2865             1.0733             1.4029

Degrees of Freedom: 51 Total (i.e. Null);  44 Residual
Null Deviance:     958.7
Residual Deviance: 229.1  AIC: 472.6
> pchisq(229.11-198.63, 2)
[1] 0.9999998
```

But this one cannot be rejected at a 95% confidence level:

```
> summary(glm(ced.del ~ I(cat=="n") + I(cat=="v") + I(cat=="m") +
      follows + factor(class), family=binomial))

Call:
glm(formula = ced.del ~ I(cat == "n") + I(cat == "v") + I(cat ==
    "m") + follows + factor(class), family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.26753  -1.22240   0.09571   1.05274   6.41257

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)       -1.43311    0.10177 -14.081  < 2e-16
I(cat == "n")TRUE  0.78333    0.06741  11.621  < 2e-16
I(cat == "v")TRUE -0.64879    0.20680  -3.137  0.00171
I(cat == "m")TRUE  0.29603    0.05634   5.254 1.49e-07
followsP           0.96188    0.07382  13.030  < 2e-16
followsV           0.53450    0.05659   9.445  < 2e-16
factor(class)2     1.26564    0.10314  12.271  < 2e-16
factor(class)3     1.04507    0.10352  10.096  < 2e-16
factor(class)4     1.37014    0.10150  13.498  < 2e-16

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 958.66  on 51  degrees of freedom
Residual deviance: 201.47  on 43  degrees of freedom
AIC: 446.94
```

```
Number of Fisher Scoring iterations: 4
> pchisq(201.5-198.63, 1)
[1] 0.9097551
```

So, that seems like what we can do to sensibly *reduce* the model. But what about the fact that it doesn't actually fit the data excellently? We'll address that below.

But let's first again look at the model coefficients, and how they express odds.

```
> subset(cedegren, class==2 & cat=="m")
   sDel sNoDel cat follows class
14  377    349   m       C     2
19  233    120   m       V     2
24   98     27   m       P     2
> pDel.followsC = 377/(377+349)
> pDel.followsV = 233/(233+120)
> pDel.followsP = 98/(98+27)
> oDel.followsC = pDel.followsC / (1 - pDel.followsC)
> oDel.followsV = pDel.followsV / (1 - pDel.followsV)
> oDel.followsP = pDel.followsP / (1 - pDel.followsP)
> oDel.followsP / oDel.followsC
[1] 3.360055
> exp(0.96188)
[1] 2.616611
> oDel.followsV / oDel.followsC
[1] 1.797458
> exp(0.53450)
[1] 1.706595
```

For these cells (chosen as cells with lots of data points...), the model coefficients fairly accurately capture the change in odds of s-Deletion by moving from the baseline of a following consonant to a following pause or vowel. You can more generally see this effect at work by comparing the model predictions for these cells in terms of logits:

```
> subset(cbind(cedegren, pred.logit=predict(ced.logr)), class==2 & cat=="m")
   sDel sNoDel cat follows class pred.logit
14  377    349   m       C     2  0.1307542
19  233    120   m       V     2  0.6648370
24   98     27   m       P     2  1.0833024
> 0.6648370 - 0.1307542
[1] 0.5340828
> 1.0833024 - 0.1307542
[1] 0.9525482
```

You can also get model predictions in terms of probabilities by saying `predict(ced.logr, type="response")`. This can be used to calculate accuracy of predictions. For example:

```
> correct <- sum(cedegren$sDel * (predict(ced.logr, type="response") >= 0.5)) +
     sum(cedegren$sNoDel * (predict(ced.logr, type="response") < 0.5))
> tot <- sum(cedegren$sDel) + sum(cedegren$sNoDel)
> correct/tot
[1] 0.6166629
```

The low accuracy mainly reflects the high variation in the data: cat, class, and follows are only weak predictors. A saturated model only has accuracy of 0.6275. The null model baseline is $1 - (3755/(3755 +$

5091)) = 0.5755. Note the huge difference between predictive accuracy and model fit! Is this a problem? It depends on what we want to do with the model. (It is certainly the case that logistic regression cannot account for any hidden correlations that are not coded in the model.)

Let us look for problems by comparing observed and fitted values, placed onto a scale of counts (not proportions). You can get the full table by the first command below, or more readably by putting it into a data frame with what you had before, but I omit the results (a whole page):

```
data.frame(fit=fitted(ced.logr)*(sDel+sNoDel), sDel, tot=(sDel+sNoDel))
ced.fit <- cbind(cedegren, data.frame(fit=fitted(ced.logr)*(sDel+sNoDel), sDel, tot=(sDel+sNoDel),
     diff=(fitted(ced.logr)*(sDel+sNoDel)-sDel)))
```

The results can be made into a simple plot (figure 2:

```
> plot(ced.fit$sDel, ced.fit$fit)
> abline(a=0, b=1)
```

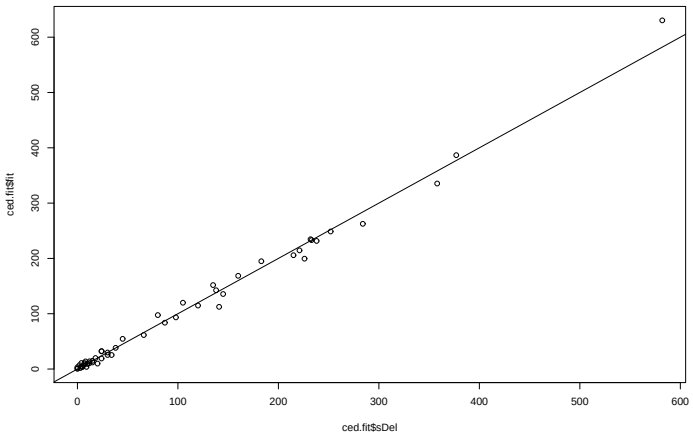


Figure 2: Model fit

This looks good, but it might instead be more revealing to look at a log scale. I use +1 since there are empty cells, and log(0) is undefined, and base 2 just for human interpretation ease. The result is in figure 3

```
plot(log(ced.fit$sDel + 1, 2), log(ced.fit$fit + 1, 2))
abline(a=0, b=1)
```

Of course, we would expect some of the low count cells to be badly estimated. But some are quite badly estimated. E.g., for cat=n, f=V, class=1, 10 deletions are predicted, but actually 20/21 tokens have deletion. In general, a number of cells for class 1 are poorly predicted, and we might worry that the model does okay on average only because very little class 1 data was collected.

Looking at the differences, 3 of the worst fit cells concern nouns of class 1 speakers:

```
> subset(ced.fit, cat=="n" & class==1)
   sDel sNoDel cat follows class       fit sDel.1 tot       diff
5    34     40   n       C     1 25.355376     34  74  -8.644624
10   20      1   n       V     1  9.884003     20  21 -10.115997
13    4     15   n       P     1 10.919041      4  19   6.919041
```

The assumptions of the logistic regression model are that each level of each factor (or each continuous explanatory variable) has an *independent* effect on the response variable. Explanatory variables do not have

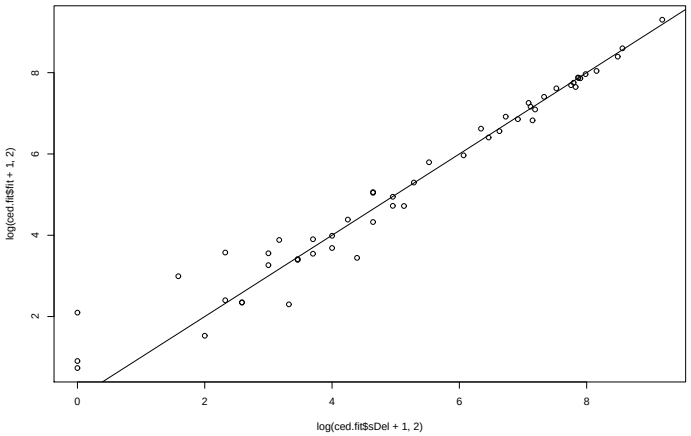


Figure 3: Model fit ($\log(x + 1, 2)$ scaled axes)

any kind of special joint effect ("local conjunctions") unless we explicitly put *interaction terms* into the model. We can put in an interaction term between just this category and class explicitly, and this model is significantly better:

```
> glm(ced.del ~ cat + follows + factor(class) + I(cat=="n" & class==1), family=binomial)

Call:  glm(formula = ced.del ~ cat + follows + factor(class) + I(cat ==
     "n" & class == 1), family = binomial)

Coefficients:
                  (Intercept)                           catd                           catm
                      -1.4573                        -0.1732                         0.1715
                         catn                           catv                       followsP
                       0.6243                        -0.7749                         0.9561
                     followsV                  factor(class)2                 factor(class)3
                       0.5388                         1.4222                         1.1999
               factor(class)4  I(cat == "n" & class == 1)TRUE
                       1.5256                         0.6147

Degrees of Freedom: 51 Total (i.e. Null);  41 Residual
Null Deviance:     958.7
Residual Deviance: 191.3  AIC: 440.8
> pchisq(198.63-191.34, 1)
>
> pchisq(deviance(ced.logr) - deviance(ced.logr2), df.residual(ced.logr) - df.residual(ced.logr2))
[1] 0.993066
```

The second form of the call to pchisq shows a cleverer way to get the deviance out of the models, rather than having to type it in again by hand. But it actually involves typing many more characters....

You can more generally put an interaction term between all levels of two factors by using the : operator:

```
> summary(glm(ced.del ~ cat + follows + factor(class) + cat:factor(class), family=binomial))

Call:
```

```
glm(formula = ced.del ~ cat + follows + factor(class) + cat:factor(class),
    family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-4.17648  -0.66890  -0.01394   0.88999   5.65947

Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)         -1.852e+00  3.200e-01  -5.786 7.20e-09
catd                 2.471e-03  4.018e-01   0.006 0.995093
catm                 7.623e-01  3.487e-01   2.186 0.028815
catn                 1.633e+00  3.719e-01   4.391 1.13e-05
catv                -1.603e+01  1.214e+03  -0.013 0.989464
followsP             9.573e-01  7.423e-02  12.896  < 2e-16
followsV             5.397e-01  5.693e-02   9.480  < 2e-16
factor(class)2       1.952e+00  3.568e-01   5.471 4.48e-08
factor(class)3       1.373e+00  3.565e-01   3.852 0.000117
factor(class)4       2.081e+00  3.506e-01   5.937 2.91e-09
catd:factor(class)2 -4.803e-01  4.427e-01  -1.085 0.278049
catm:factor(class)2 -7.520e-01  3.877e-01  -1.940 0.052417
catn:factor(class)2 -9.498e-01  4.143e-01  -2.292 0.021890
catv:factor(class)2  1.473e+01  1.214e+03   0.012 0.990323
catd:factor(class)3  9.870e-02  4.451e-01   0.222 0.824513
catm:factor(class)3 -2.376e-01  3.874e-01  -0.613 0.539550
catn:factor(class)3 -1.085e+00  4.129e-01  -2.627 0.008624
catv:factor(class)3  1.679e+01  1.214e+03   0.014 0.988968
catd:factor(class)4 -1.898e-01  4.366e-01  -0.435 0.663692
catm:factor(class)4 -8.533e-01  3.805e-01  -2.242 0.024942
catn:factor(class)4 -1.071e+00  4.074e-01  -2.627 0.008604
catv:factor(class)4  1.521e+01  1.214e+03   0.013 0.990006

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 958.66  on 51  degrees of freedom
Residual deviance: 135.68  on 30  degrees of freedom
AIC: 407.15

Number of Fisher Scoring iterations: 15

> pchisq(198.63-135.68, 42-30)
[1] 1
```

This model *does* fit significantly better than the model without the interaction term. However, looking at the coefficients, not only have a lot of parameters been added, but many of them look to be otiose. (Note similar estimates, little confidence that values aren't zero.) It looks like we could build a better model, and we will in a moment after another note on interaction terms.

Rather than writing `cat + factor(class) + cat:factor(class)`, you can more simply write `cat*factor(class)`. This works iff you want the written interaction term, and interaction terms and main effects for every subset of these variables. For instance, we can make the saturated model with:

```
summary(glm(ced.del ~ cat*follows*factor(class), family=binomial))
```

By construction, this model has 0 degrees of freedom, and a residual deviance of 0 (approximately – minor errors in numeric optimization occur). However, it isn't very interesting as a model.

It'd be nice to understand the linguistic situation better, but looking just at the model we built with the cat:class interaction term, it now looks like this is what is going on: cat n, and more marginally, cat m have special behavior (the other categories don't). The following environment and class are significant (even though classes 2 and 4 seem to behave similarly). The crucial interaction is then between class and cat n (where deletion of /s/ occurs highly significantly less often than would otherwise be expected for class 3 and cat n, and, although the counts are small, rather more often than expected with class 1 and cat n). So, here's a couple of last attempts at a model:

```
> summary(glm(ced.del ~ I(cat=="n") + I(cat=="m") + follows + factor(class)
        + I(cat=="n"&class==3)+I(cat=="n"&class==1), family=binomial))

Call:
glm(formula = ced.del ~ I(cat == "n") + I(cat == "m") + follows +
    factor(class) + I(cat == "n" & class == 3) + I(cat == "n" &
    class == 1), family = binomial)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-4.18107  -1.34799   0.07195   1.09391   5.98469

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                   -1.59891    0.11696 -13.671  < 2e-16 ***
I(cat == "n")TRUE              0.96556    0.07861  12.283  < 2e-16 ***
I(cat == "m")TRUE              0.32632    0.05533   5.898 3.68e-09 ***
followsP                       0.95515    0.07399  12.910  < 2e-16 ***
followsV                       0.51822    0.05640   9.188  < 2e-16 ***
factor(class)2                 1.36269    0.12019  11.338  < 2e-16 ***
factor(class)3                 1.29196    0.12188  10.600  < 2e-16 ***
factor(class)4                 1.48044    0.11872  12.470  < 2e-16 ***
I(cat == "n" & class == 3)TRUE -0.58351    0.11844  -4.927 8.37e-07 ***
I(cat == "n" & class == 1)TRUE  0.41904    0.23116   1.813   0.0699 .
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 958.66  on 51  degrees of freedom
Residual deviance: 180.56  on 42  degrees of freedom
AIC: 428.02

Number of Fisher Scoring iterations: 4
```

This model has the same number of parameters as the initial model we built, but improves the loglikelihood by about 18. That is, it makes the observed data more than 65 million times more likely.

But if you look back at the raw stats for nouns of class 1 speakers, there's still just this weird fact that they almost always delete /s/ before a vowel, and almost never delete it before a pause (why ever that may be). And our model still cannot capture that! But if we make an interaction between class 1, cat n and the levels of follows, then we could. We add two more parameters, but, behold, the residual deviance goes down a lot! The deviance is close to the cat:class interaction model, in a rather more interesting way.

```
> summary(ced.logr <- glm(ced.del ~ I(cat=="n") + I(cat=="m") + follows + factor(class)
       + I(cat=="n"&class==3)+I(cat=="n"&class==1) +I(cat=="n"&class==1):follows,
       family=binomial))

Call:
glm(formula = ced.del ~ I(cat == "n") + I(cat == "m") + follows +
    factor(class) + I(cat == "n" & class == 3) + I(cat == "n" &
    class == 1) + I(cat == "n" & class == 1):follows, family = binomial)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-3.634e+00  -1.280e+00  -1.054e-08   9.274e-01   6.079e+00

Coefficients:
                                       Estimate Std. Error z value Pr(>|z|)
(Intercept)                            -1.59612    0.11698 -13.644  < 2e-16 ***
I(cat == "n")TRUE                       0.95929    0.07864  12.198  < 2e-16 ***
I(cat == "m")TRUE                       0.32468    0.05534   5.867 4.45e-09 ***
followsP                                0.99832    0.07532  13.255  < 2e-16 ***
followsV                                0.50026    0.05665   8.831  < 2e-16 ***
factor(class)2                          1.36256    0.12021  11.334  < 2e-16 ***
factor(class)3                          1.29016    0.12191  10.583  < 2e-16 ***
factor(class)4                          1.47780    0.11874  12.445  < 2e-16 ***
I(cat == "n" & class == 3)TRUE         -0.58311    0.11854  -4.919 8.70e-07 ***
I(cat == "n" & class == 1)TRUE          0.47431    0.26866   1.765  0.07749 .
followsP:I(cat == "n" & class == 1)TRUE -2.15756    0.61380  -3.515  0.00044 ***
followsV:I(cat == "n" & class == 1)TRUE  2.65799    1.05244   2.526  0.01155 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 958.66  on 51  degrees of freedom
Residual deviance: 146.72  on 40  degrees of freedom
AIC: 398.19

Number of Fisher Scoring iterations: 4
```

3 Long format R logistic regression models (the Design package)

You need to have loaded the Design package for this part to work. Look at Baayen chapter 1 if you don't know how to do this!

Until now, we have used binary outcome data in a summary format (counts of sDel and sNoDel for each combination of levels of factors). An alternative is long format, where each observation is a line. Collected data often starts out in this form. You can then examine the data by constructing cross-tabulations. You can do them with any number of variables, but they get harder to read with more than two.

```
> ced.long <- read.table("cedegren-long.txt", header=T)
> ced.long[1:5,]
  sDel cat follows class
1    1   m       C     1
```

```
2    1   m       C     1
3    1   m       C     1
4    1   m       C     1
5    1   m       C     1
> ced.long$class <- factor(ced.long$class)
> attach(ced.long)
> xtabs(~ sDel + class)
    class
sDel    1    2    3    4
   0  414 1033 1065 1243
   1  165 1514 1320 2092
```

Do we need to build logistic regression models, or could we get everything that we need to get from just looking at crosstabs? For instance, it seems like we can already see here that /s/-deletion is strongly disfavored by class 1 speakers, but moderately preferred by other classes. Sometimes looking at crosstabs works, but the fundamental observation is this: the predictive effect of variables can be spurious, hidden or reversed when just looking at the marginal totals in crosstabs, because they do not take into account correlations of explanatory variables. The magical good thing that logistic regression does is work out the best way to attribute causal effect to explanatory variables. (Of course, it can only do this for variables coded in the data....)

Contrary to what Baayen suggests, you can load this into the basic glm function. Here's what you get:

```
> summary(glm(sDel ~ cat + follows + class, family=binomial))

Call:
glm(formula = sDel ~ cat + follows + class, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9219  -1.2013   0.7298   1.0796   1.8392

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.31827    0.12221 -10.787  < 2e-16
catd        -0.16931    0.10032  -1.688 0.091459
catm         0.17858    0.08952   1.995 0.046053
catn         0.66672    0.09651   6.908 4.91e-12
catv        -0.76754    0.21844  -3.514 0.000442
followsP     0.95255    0.07400  12.872  < 2e-16
followsV     0.53408    0.05660   9.436  < 2e-16
class2       1.27045    0.10320  12.310  < 2e-16
class3       1.04805    0.10355  10.122  < 2e-16
class4       1.37425    0.10155  13.532  < 2e-16

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 12061  on 8845  degrees of freedom
Residual deviance: 11301  on 8836  degrees of freedom
AIC: 11321

Number of Fisher Scoring iterations: 4
```

The deviances are huge when each observation is a cell. But note that the difference between the null and

residual deviance is the same as we started off with originally (760). And the estimated coefficients are just as we saw previously.

However, nevertheless, the Design package lets you do various spiffy stuff. So, let's try that one.

It turns out that some things don't work with lrm unless you use the magical incantation to create a data distribution object. I'm still trying to figure out what this does myself, but if you print it, part of it seems to summarize the factors....

```
> ced.ddist <- datadist(ced.long)
> options(datadist="ced.ddist")
```

But at any rate: (i) you need to have done it before you can call various other methods like summary or anova on an lrm object, and (ii) it's best to call it right at the beginning and set it with the options() function, because then various data statistics are stored there rather than being computed each time.

```
> ced.lrm <- lrm(sDel ~ cat + follows + factor(class))
> ced.lrm

Logistic Regression Model

lrm(formula = sDel ~ cat + follows + factor(class))

Frequencies of Responses
   0    1
3755 5091

       Obs  Max Deriv Model L.R.       d.f.          P          C        Dxy      Gamma
      8846      5e-09     760.04          9          0       0.66       0.32      0.336
     Tau-a         R2      Brier
     0.156      0.111      0.224

          Coef    S.E.    Wald Z P
Intercept -1.3183 0.12221 -10.79 0.0000
cat=d     -0.1693 0.10032  -1.69 0.0915
cat=m      0.1786 0.08952   1.99 0.0461
cat=n      0.6667 0.09651   6.91 0.0000
cat=v     -0.7675 0.21844  -3.51 0.0004
follows=P  0.9525 0.07400  12.87 0.0000
follows=V  0.5341 0.05660   9.44 0.0000
class=2    1.2704 0.10320  12.31 0.0000
class=3    1.0480 0.10355  10.12 0.0000
class=4    1.3742 0.10155  13.53 0.0000
```

Here "Model L.R." is again the difference between Null and Residual deviance. This is again associated with a difference in degrees of freedom and a p-value. Hooray! Our model is better than one with no factors!

Now I can get an anova. Note: for lrm, ANOVA does not present sequential tables adding factors, but considers each factor separately!

```
> anova(ced.lrm)
                Wald Statistics          Response: sDel

 Factor     Chi-Square d.f. P
 cat        164.93     4    <.0001
 follows    214.96     2    <.0001
```

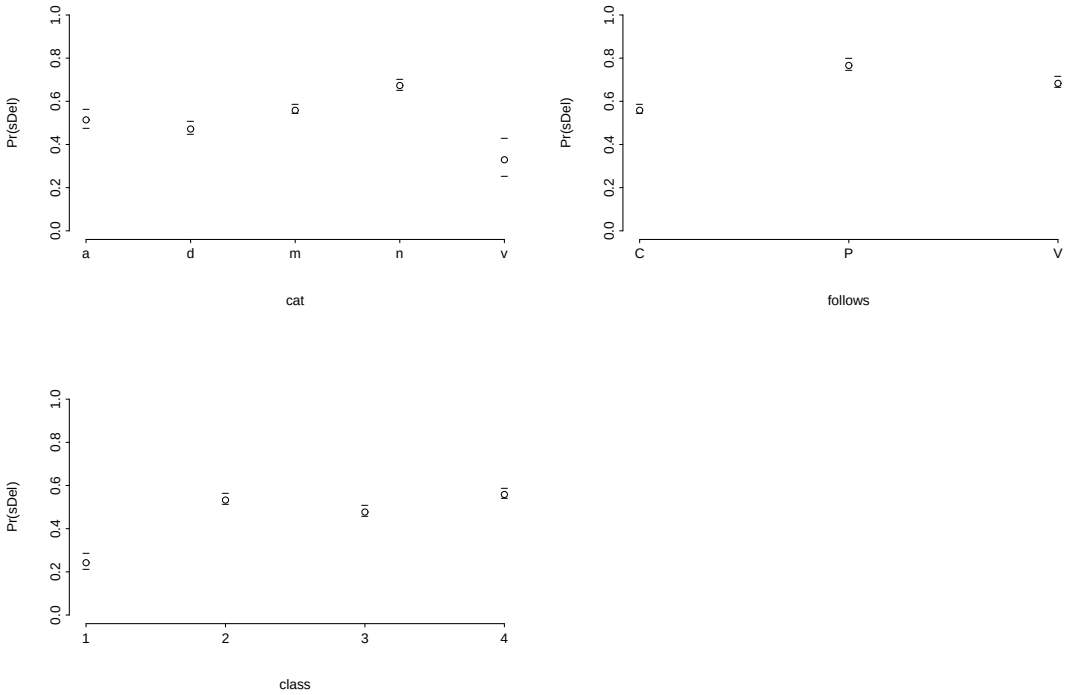
```
class      197.62     3    <.0001
TOTAL      656.61     9    <.0001
```

Among the nifty features of lrm is that you can do penalized (or regularized) estimation of coefficients to avoid overfitting (searched for with pentrace, or specified with the penalty parameter in lrm) – see Baayen pp. 224-227. This was the regularlization that appeared in the Hayes and Wilson paper.

lrm has a really nice option to plot the logit coefficients (turned back into probabilities) for each level of each factor. AFAIK, you can't do this with glm.

```
> par(mfrow = c(2,2))
> plot(ced.lrm, fun=plogis, ylab="Pr(sDel)", adj.subtitle=F, ylim=c(0,1))
```

It'd be even more fun with a real-valued explanatory variable.



One thing that we haven't addressed is that the class variable should by rights be an ordinal scale. There is an extension of logistic regression to ordinal explanatory variables, and it's discussed in Baayen, pp. 227ff. However, looking at this figure shows that the probability of /s/-deletion does not increase monotonically as you head along this putative ordinal scale. Something else seems to be at work. And so an ordinal logistic regression analysis is counterindicated here.

4 Working out data likelihood: practice with vectors

Most R calculations work directly on vectors, and that's often handy. Here we use that to directly work out the likelihood of the data according to a model. We will use our very initial model again for illustration.

We start with predicted probabilities of sDeletion for each cell, and turn them into log props:

```
> probsdel <- fitted(ced.logr)
```

```
> logprobdel <- log(probsdel)
> logprobnodel <- log(1 - probsdel)
```

Each of these components is a vector, as you'd see if you print them.

We then work out the total data loglikelihood in log space (a scalar) by summing the vectors of log expected counts. We would exponentiate it for data likelihood (but, here, this is too small a number for the computer to represent).

```
> loglike <- sum(cedegren$sDel * logprobdel) + sum(cedegren$sNoDel * logprobnodel)
> loglike
[1] -5650.287
> exp(loglike)
[1] 0
```

We can now also work out the loglikelihood of the null model, which just uses the overall rate of /s/-Deletion in the data (3755 / (3755 + 5091)).

```
> sDelProb <- 3755 / (3755 + 5091)
> loglikenull <- 3755 * log(sDelProb) + 5091 * log(1-sDelProb)
> loglikenull
[1] -6030.306
```

We then work out the G^2 quantity of -2 times the difference in log likelihoods:

```
> Gsq = -2 * (loglikenull - loglike)
> Gsq
[1] 760.0375
```

Almost miraculously, this does turn out to be exactly the same quantity that glm and lrm gave us above!!