Stat 5421 Lecture Notes: To Accompany Agresti Ch 9

Charles J. Geyer

November 09, 2020

Contents

Section 9.1 Two-Way Tables

Section 9.1.1 Only Main Effects

The *independence* or *homogeneity of proportions* model has mean-value parameters

$$\mu_{ij} = \alpha_i \beta_j$$

or canonical parameters

$$\theta_{ij} = \alpha_i + \beta_j \tag{1}$$

where the alphas and betas are different in the two equations (this causes no confusion, since if we look at parameters at all, these are usually canonical parameters).

Section 9.1.3 Interactions

The *saturated* model has completely arbitrary parameters. The mean value parameters μ_{ij} have no specified structure. The canonical parameters θ_{ij} have no specified structure. So this is the largest model. It has the most (arbitrarily variable) parameters.

Section 9.1.5 Hierarchical Models

The hierarchical model principle says that, if you have an interaction term of a certain order, then you have all lower-order interactions of the same variables. For the saturated model for a two-way table this says

$$\theta_{ij} = \alpha_i + \beta_j + \gamma_{ij} \tag{2}$$

(but this just says θ_{ij} can be anything. This is not an identifiable parameterization.

Identifiability

A statistical model (in general, not just in categorical data analysis) is *identifiable* if there is only one (vector) parameter value corresponding to a probability distribution. In exponential family models (Notes on Exponential Families, Section 6.2) every different mean vector corresponds to a different distribution, but different canonical parameter vectors can correspond to the same distribution.

If we are assuming *Poisson sampling* (Notes on Exponential Families, Part I, section 7) then the canonical parameters are identifiable if and only if the mean-value parameters are (because the link function (componentwise log) is invertible).

Clearly the specification for the two-way independence model (1) is not identifiable because one can add a constant to all of the alphas and subtract the same constant from all the betas and get the same result

$$(\alpha_i + c) + (\beta_j - c) = \alpha_i + \beta_j$$

Hence (2) is also not identifiable.

But we don't need to worry about identifiability when fitting models because the computer automatically takes care of it.

For example, consider the data in Table 3.8 in Agresti

```
counts <- c(17066, 14464, 788, 126, 37, 48, 38, 5, 1, 1)
drinks <- rep(c("0", "< 1", "1-2", "3-5", ">= 6"), times = 2)
malformation <- rep(c("Absent", "Present"), each = 5)
data.frame(drinks, malformation, counts)
```

```
##    drinks malformation counts
## 1       0       Absent  17066
## 2     < 1       Absent  14464
## 3     1-2       Absent    788
## 4     3-5       Absent    126
## 5    >= 6       Absent     37
## 6       0      Present     48
## 7     < 1      Present     38
## 8     1-2      Present      5
## 9     3-5      Present      1
## 10   >= 6      Present      1
```

We can fit this using R function `chisq.test`

```
foo <- xtabs(counts ~ malformation + drinks)
foo
```

```
##             drinks
## malformation   < 1  >= 6     0   1-2   3-5
##      Absent  14464    37 17066   788   126
##      Present    38     1    48     5     1
```

```
chisq.test(foo)
```

```
## Warning in chisq.test(foo): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  foo
## X-squared = 12.082, df = 4, p-value = 0.01675
```

But it says "approximation may be incorrect" so try

```
chisq.test(foo, simulate.p.value = TRUE)
```

```
##
##  Pearson's Chi-squared test with simulated p-value (based on 2000
##  replicates)
##
## data:  foo
## X-squared = 12.082, df = NA, p-value = 0.02999
```

But we can also use R function `glm` and its helper functions to do the job.

```
gout.indep <- glm(counts ~ malformation + drinks, family = poisson)
summary(gout.indep)
```

```
##
## Call:
## glm(formula = counts ~ malformation + drinks, family = poisson)
##
## Deviance Residuals:
##        1         2         3         4         5         6         7         8
##  0.00659   0.02830  -0.09735  -0.05669  -0.14540  -0.12356  -0.53649   1.56555
##        9        10
##  0.86842   1.63069
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)          9.579183   0.008309 1152.83   <2e-16 ***
## malformationPresent -5.855811   0.103844  -56.39   <2e-16 ***
## drinks>= 6          -5.944456   0.162434  -36.60   <2e-16 ***
## drinks0              0.165610   0.011287   14.67   <2e-16 ***
## drinks1-2           -2.906219   0.036469  -79.69   <2e-16 ***
## drinks3-5           -4.737855   0.089123  -53.16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
```

```
##     Null deviance: 95424.044  on 9  degrees of freedom
## Residual deviance:     6.202  on 4  degrees of freedom
## AIC: 80.511
##
## Number of Fisher Scoring iterations: 4
```

```
gout.sat <- glm(counts ~ malformation * drinks, family = poisson)
summary(gout.sat)
```

```
##
## Call:
## glm(formula = counts ~ malformation * drinks, family = poisson)
##
## Deviance Residuals:
##  [1]  0  0  0  0  0  0  0  0  0  0
##
## Coefficients:
##                                 Estimate Std. Error  z value Pr(>|z|)
## (Intercept)                     9.579418   0.008315 1152.082   <2e-16 ***
## malformationPresent            -5.941832   0.162434  -36.580   <2e-16 ***
## drinks>= 6                     -5.968500   0.164609  -36.259   <2e-16 ***
## drinks0                         0.165425   0.011302   14.637   <2e-16 ***
## drinks1-2                      -2.909920   0.036581  -79.547   <2e-16 ***
## drinks3-5                      -4.743136   0.089474  -53.011   <2e-16 ***
## malformationPresent:drinks>= 6  2.330914   1.026354    2.271   0.0231 *
## malformationPresent:drinks0     0.068189   0.217432    0.314   0.7538
## malformationPresent:drinks1-2   0.881772   0.477131    1.848   0.0646 .
## malformationPresent:drinks3-5   1.105550   1.017011    1.087   0.2770
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance:  9.5424e+04  on 9  degrees of freedom
## Residual deviance: -4.7518e-13  on 0  degrees of freedom
## AIC: 82.309
##
## Number of Fisher Scoring iterations: 3
```

```
anova(gout.indep, gout.sat, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: counts ~ malformation + drinks
## Model 2: counts ~ malformation * drinks
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         4      6.202
## 2         0      0.000  4    6.202   0.1846
```

```
anova(gout.indep, gout.sat, test = "Rao")
```

```
## Analysis of Deviance Table
##
## Model 1: counts ~ malformation + drinks
## Model 2: counts ~ malformation * drinks
##   Resid. Df Resid. Dev Df Deviance    Rao Pr(>Chi)
```

```
## 1         4      6.202
## 2         0      0.000  4    6.202 12.082  0.01675 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
p.value.lrt <- anova(gout.indep, gout.sat, test = "LRT")[2, "Pr(>Chi)"]
p.value.rao <- anova(gout.indep, gout.sat, test = "Rao")[2, "Pr(>Chi)"]
p.value.chisq <- suppressWarnings(chisq.test(foo)$p.value)
p.value.lrt
```

```
## [1] 0.1845623
```

```
p.value.rao
```

```
## [1] 0.01675137
```

```
p.value.chisq
```

```
## [1] 0.0167514
```

We notice a number of things.

- The P-value for the Rao test (0.01675) is exactly equal to that output for the Chi-Square test because the Pearson chi-square test is a special case of the Rao test.
- The likelihood ratio test disagrees quite strongly with the Rao test. Of course, these test are asymptotically equivalent, but here the sample size is not "large". The total number of subjects (3.2574×10^4) is large, but the expected number in some cells of the contingency table are quite small, a lot less than what the rule of thumb says is necessary for valid asymptotics (at least five in each cell), so maybe that accounts for the difference.

```
suppressWarnings(chisq.test(foo)$expected)
```

```
##             drinks
## malformation         < 1       >= 6           0        1-2         3-5
##      Absent  14460.59624 37.8915086 17065.13888 790.735955 126.6374102
##      Present    41.40376  0.1084914    48.86112   2.264045   0.3625898
```

- The computer (R function `glm`) knows how to deal with identifiability. It adds an intercept, so it is really using the formula

$$\theta_{ij} = \alpha + \beta_i + \gamma_j$$

for the independence model and

$$\theta_{ij} = \alpha + \beta_i + \gamma_j + \delta_{ij}$$

for the saturated model. Then it "drops" (sets equal to zero) one of the betas, one of the gammas, and all interaction terms corresponding to the dropped beta and the dropped gamma.

```
names(coef(gout.indep))
```

```
## [1] "(Intercept)"         "malformationPresent" "drinks>= 6"
## [4] "drinks0"             "drinks1-2"           "drinks3-5"
```

It keeps the "intercept" (α in our notation just above). It drops one of the coefficients for the factor `malformation` (and keeps the other one). It drops one of the coefficients for the factor `drinks` (and keeps the other four). And this gives it an identifiable model.

```
names(coef(gout.sat))
```

```
##  [1] "(Intercept)"                   "malformationPresent"
##  [3] "drinks>= 6"                    "drinks0"
##  [5] "drinks1-2"                     "drinks3-5"
```

```
##  [7] "malformationPresent:drinks>= 6" "malformationPresent:drinks0"
##  [9] "malformationPresent:drinks1-2"  "malformationPresent:drinks3-5"
```

It has all the parameters of the independence model plus a lot of "interaction" parameters (the ones whose names contain colons). But it drops all the "interaction" parameters that involve parameters that were "dropped" from the independence model (it contains no "interaction" terms containing `malformationAbsent` or `drinks< 1`) And this gives it an identifiable model.

Section 9.2 Three-Way Tables

Three way tables are just like two-way tables except the data have three subscripts for the three dimensions of the table. Our analogs of

Section 9.2.1 Only Main Effects

The *independence* or *homogeneity of proportions* model has mean-value parameters

$$\mu_{ijk} = \alpha_i \beta_j \gamma_k$$

or canonical parameters

$$\theta_{ijk} = \alpha_i + \beta_j + \gamma_k \tag{3}$$

where the alphas, betas, and gammas are different in the two equations (this causes no confusion, since if we look at parameters at all, these are usually canonical parameters).

The Saturated Model

As before, the *saturated* model has completely arbitrary parameters. The mean value parameters μ_{ijk} have no specified structure. The canonical parameters θ_{ijk} have no specified structure. So this is the largest model. It has the most (arbitrarily variable) parameters.

Example: High School Student Survey

This is Table 9.3 in Agresti and can be found in R package `CatDataAnalysis`.

```
# clean up R global environment
rm(list = ls())

library(CatDataAnalysis)
data(table_9.3)
names(table_9.3)
```

```
## [1] "a"     "c"     "m"     "count"
```

```
foo <- xtabs(count ~ a + c + m, data = table_9.3)
foo
```

```
## , , m = 1
##
##    c
## a     1   2
```

```
##   1 911  44
##   2   3   2
##
## , , m = 2
##
##    c
## a     1   2
##   1 538 456
##   2  43 279
```

Since `a`, `c`, and `m` are categorical, we should perhaps make them factors. But since they each have only two values, it does not matter whether we make them factors or not. If they had more than two values, then it would be essential to make them factors.

We can make these data look nicer by copying the names out of the book

```
dimnames(foo) <- list(alcohol = c("yes", "no"), cigarettes = c("yes", "no"), marijuana = c("yes", "no"))
aperm(foo, c(2, 3, 1))
```

```
## , , alcohol = yes
##
##           marijuana
## cigarettes yes  no
##        yes 911 538
##        no   44 456
##
## , , alcohol = no
##
##           marijuana
## cigarettes yes  no
##        yes   3  43
##        no    2 279
```

We needed R function `aperm`, which permutes the dimensions of an array, to present the data in the same order as in the book.

Of course, the order doesn't matter and neither do the `dimnames`. That is just to match the book.

Now R function `chisq.test` is useless, but R function `glm` has no problems. Our test of independence or homogeneity of proportions is the same, just with three "predictors" rather than two (the "predictors" are in scare quotes because they are just the `dimnames`)

```
gout.indep <- glm(count ~ a + c + m, data = table_9.3, family = poisson)
gout.sat <- glm(count ~ a * c * m, data = table_9.3, family = poisson)
anova(gout.indep, gout.sat, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: count ~ a + c + m
## Model 2: count ~ a * c * m
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1         4       1286
## 2         0          0  4     1286 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(gout.indep, gout.sat, test = "Rao")
```

```
## Analysis of Deviance Table
```

```
##
## Model 1: count ~ a + c + m
## Model 2: count ~ a * c * m
##   Resid. Df Resid. Dev Df Deviance    Rao  Pr(>Chi)
## 1         4       1286
## 2         0          0  4     1286 1411.4 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Clearly, the independence model does not fit the data. So statistics says these variables, alcohol use, cigarette use, and marijuana use have some association. No real surprise, but the data do confirm what just about everybody assumes.

Interactions

Our saturated model has the three-way interaction `a * c * m` but there can also be two-way interactions. The model with all two-way interactions would have canonical parameter

$$\theta_{ijk} = \alpha_{ij} + \beta_{ik} + \gamma_{jk}$$

(as before the terms with alphas, betas, and gammas need not match up with previous uses of these greek letters). And people who insist on the hierarchical principle would write

$$\theta_{ijk} = \alpha + \beta_i + \gamma_j + \delta_k + \varepsilon_{ij} + \zeta_{ik} + \eta_{jk}$$

but both formulas specify the same model and neither is identifiable.

So let's look at that model.

```
gout.all.two.way <- glm(count ~ (a + c + m)^2, data = table_9.3, family = poisson)
anova(gout.indep, gout.all.two.way, gout.sat, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: count ~ a + c + m
## Model 2: count ~ (a + c + m)^2
## Model 3: count ~ a * c * m
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         4    1286.02
## 2         1       0.37  3  1285.65   <2e-16 ***
## 3         0       0.00  1     0.37   0.5408
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
foompter <- anova(gout.indep, gout.all.two.way, gout.sat, test = "LRT")
foompter.p.values <- foompter[-1, "Pr(>Chi)"]
foompter.p.values
```

```
## [1] 1.915858e-278  5.408396e-01
```

This says the two-way interactions model (model 2) fits as well as the three-way interactions model (model 3, the saturated model) because $P = 0.5408$ is not statistically significant (not even close).

And it says the two-way interactions model (model 2) fits much better than the main effects only model (model 1, the independence model) because $P < 2 \times 10^{-16}$ is highly statistically significant (by anyone's standards).

We should not be overly impressed by very very small P-values because asymptotic approximation gives only small absolute errors rather than small relative errors. All this very very small P-value says is that an exact

calculation (if we could do it, which we cannot) would be something small, say $P < 0.001$. But there is no reason to believe the $P < 2 \times 10^{-16}$ that R prints is correct to within even several orders of magnitude.

Many Models

There are many hierarchical models here, because we need not include all of the main effects or all two-way interactions. The hierarchical principle says that if we have a two-way interaction, then we must have the corresponding main effects, but that leaves

$$
\begin{aligned}
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \varepsilon_{ij} + \zeta_{ik} + \eta_{jk} + \theta_{ijk} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \varepsilon_{ij} + \zeta_{ik} + \eta_{jk} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \varepsilon_{ij} + \zeta_{ik} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \varepsilon_{ij} + \eta_{jk} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \zeta_{ik} + \eta_{jk} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \varepsilon_{ij} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \zeta_{ik} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k + \eta_{jk} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \varepsilon_{ij} \\
\theta_{ijk} &= \alpha + \beta_i + \delta_k + \zeta_{ik} \\
\theta_{ijk} &= \alpha + \gamma_j + \delta_k + \eta_{jk} \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j + \delta_k \\
\theta_{ijk} &= \alpha + \beta_i + \gamma_j \\
\theta_{ijk} &= \alpha + \beta_i + \delta_k \\
\theta_{ijk} &= \alpha + \gamma_j + \delta_k \\
\theta_{ijk} &= \alpha + \beta_i \\
\theta_{ijk} &= \alpha + \gamma_j \\
\theta_{ijk} &= \alpha + \delta_k \\
\theta_{ijk} &= \alpha
\end{aligned}
$$

Of course, we already know many of these models don't fit the data. We saw that the main effects only model doesn't fit. So none of its submodels can fit either. But this is a complete listing of all hierarchical models (except for the saturated model).

How can we choose among all these models? There is a methodology for that, but this is getting a bit ahead of ourselves. There will be a handout for that. But for now, we just show it without much explanation.

```
library(glmbb)
out <- glmbb(count ~ a * c * m, data = table_9.3, family = poisson)
summary(out)
```

```
##
## Results of search for hierarchical models with lowest AIC.
## Search was for all models with AIC no larger than min(AIC) + 10
## These are shown below.
##
##   criterion  weight  formula
##   63.42      0.6927  count ~ a*c + a*m + c*m
##   65.04      0.3073  count ~ a*c*m
```

This says that none of the models that do not have all three two-way interactions fit the data according to the criterion: none have AIC less than the AIC for the best model ("best" according to AIC) plus 10. We don't know what AIC is yet, but we will learn in a later handout. For now, just consider it a criterion of goodness of fit.

If we want to see how bad some of those non-fitting models were, we can increase the cutoff.

```
library(glmbb)
out <- glmbb(count ~ a * c * m, data = table_9.3, family = poisson, cutoff = 90)
summary(out)
```

```
##
## Results of search for hierarchical models with lowest AIC.
## Search was for all models with AIC no larger than min(AIC) + 90
## These are shown below.
##
##   criterion  weight     formula
##    63.42     6.927e-01  count ~ a*c + a*m + c*m
##    65.04     3.073e-01  count ~ a*c*m
##   153.06     2.369e-20  count ~ a*c + c*m
```

That is a huge jump up to the next best fitting model, which does not (take my word for it for now) does not fit the data well at all.

Section 9.4 Four-Way and Beyond

And so on and so forth. Higher order tables are just more complicated, but present no new issues.

Example: Seat-Belt Use

These data are given in Table 9.8 in Agresti. The are apparently not in R package `CatDataAnalysis`.

```
# clean up R global environment
rm(list = ls())

count <- c(7287, 11587, 3246, 6134, 10381, 10969, 6123, 6693,
           996,   759,  973,  757,   812,   380, 1084,  513)
injury <- gl(2, 8, 16, labels = c("No", "Yes"))
gender <- gl(2, 4, 16, labels = c("Female", "Male"))
location <- gl(2, 2, 16, labels = c("Urban", "Rural"))
seat.belt <- gl(2, 1, 16, labels = c("No", "Yes"))
data.frame(gender, location, seat.belt, injury, count)
```

```
##    gender location seat.belt injury count
## 1  Female    Urban        No     No  7287
## 2  Female    Urban       Yes     No 11587
## 3  Female    Rural        No     No  3246
## 4  Female    Rural       Yes     No  6134
## 5    Male    Urban        No     No 10381
## 6    Male    Urban       Yes     No 10969
## 7    Male    Rural        No     No  6123
## 8    Male    Rural       Yes     No  6693
## 9  Female    Urban        No    Yes   996
## 10 Female    Urban       Yes    Yes   759
```

```
## 11 Female    Rural        No    Yes   973
## 12 Female    Rural       Yes    Yes   757
## 13   Male    Urban        No    Yes   812
## 14   Male    Urban       Yes    Yes   380
## 15   Male    Rural        No    Yes  1084
## 16   Male    Rural       Yes    Yes   513
```

```
xtabs(count ~ seat.belt + injury + location + gender)
```

```
## , , location = Urban, gender = Female
##
##          injury
## seat.belt    No   Yes
##       No   7287   996
##       Yes 11587   759
##
## , , location = Rural, gender = Female
##
##          injury
## seat.belt    No   Yes
##       No   3246   973
##       Yes  6134   757
##
## , , location = Urban, gender = Male
##
##          injury
## seat.belt    No   Yes
##       No  10381   812
##       Yes 10969   380
##
## , , location = Rural, gender = Male
##
##          injury
## seat.belt    No   Yes
##       No   6123  1084
##       Yes  6693   513
```

Looks OK.

```
out <- glmbb(count ~ seat.belt * injury * location * gender,
    family = "poisson")
summary(out)
```

```
##
## Results of search for hierarchical models with lowest AIC.
## Search was for all models with AIC no larger than min(AIC) + 10
## These are shown below.
##
##   criterion  weight
##   182.8      0.24105
##   183.1      0.21546
##   184.0      0.13742
##   184.8      0.09055
##   184.9      0.08446
##   185.0      0.08042
##   185.5      0.06462
```

```
##   185.8      0.05365
##   186.8      0.03237
##   formula
##   count ~ seat.belt*injury*location + seat.belt*location*gender + injury*location*gender
##   count ~ injury*gender + seat.belt*injury*location + seat.belt*location*gender
##   count ~ seat.belt*injury + seat.belt*location*gender + injury*location*gender
##   count ~ seat.belt*injury*location + seat.belt*injury*gender + seat.belt*location*gender + injury*l
##   count ~ seat.belt*injury + injury*location + injury*gender + seat.belt*location*gender
##   count ~ seat.belt*injury*location + seat.belt*injury*gender + seat.belt*location*gender
##   count ~ seat.belt*injury*location*gender
##   count ~ seat.belt*injury*gender + seat.belt*location*gender + injury*location*gender
##   count ~ injury*location + seat.belt*injury*gender + seat.belt*location*gender
```

Now there are several models that fit these data fairly well. We will leave it at that for now.

Parametric Bootstrap

Assuming Poisson Sampling

The method of R function `anova` for objects of class `"glm"` produced by R function `glm` has no optional argument `simulate.p.value`.

So here is how we could do that for our first example.

```
# clean up R global environment
rm(list = ls())
# re-establish stuff done in first section
counts <- c(17066, 14464, 788, 126, 37, 48, 38, 5, 1, 1)
drinks <- rep(c("0", "< 1", "1-2", "3-5", ">= 6"), times = 2)
malformation <- rep(c("Absent", "Present"), each = 5)
gout.indep <- glm(counts ~ malformation + drinks, family = poisson)
gout.sat <- glm(counts ~ malformation * drinks, family = poisson)
aout <- anova(gout.indep, gout.sat, test = "LRT")
```

This gets us back to where we were before we removed everything produced in our original analysis of these data. Now we need the estimated mean values (expected cell counts) and the P-value for the test

```
p.value.hat <- aout[2, "Pr(>Chi)"]
p.value.hat
```

```
## [1] 0.1845623
```

```
mu.hat <- predict(gout.indep, type = "response")
mu.hat
```

```
##            1            2            3            4            5            6
## 1.706514e+04 1.446060e+04 7.907360e+02 1.266374e+02 3.789151e+01 4.886112e+01
##            7            8            9           10
## 4.140376e+01 2.264045e+00 3.625898e-01 1.084914e-01
```

Now we can do the simulation

```
# set random number generator seed for reproducibility
set.seed(42)

nboot <- 999
p.value.star <- double(nboot)
```

```
for (iboot in 1:nboot) {
    # simulate new data from fitted model
    counts.star <- rpois(length(mu.hat), lambda = mu.hat)
    gout.indep.star <- glm(counts.star ~ malformation + drinks,
        family = poisson)
    gout.sat.star <- glm(counts.star ~ malformation * drinks,
        family = poisson)
    aout.star <- anova(gout.indep.star, gout.sat.star, test = "LRT")
    p.value.star[iboot] <- aout.star[2, "Pr(>Chi)"]
}
all.p.values <- c(p.value.star, p.value.hat)
mean(all.p.values <= p.value.hat)
```

```
## [1] 0.143
```

```
sd(all.p.values <= p.value.hat) / sqrt(nboot)
```

```
## [1] 0.01108136
```

The code above is a bit tricky, so here is the explanation.

- Each time through the loop we simulate new data based on our best estimate of the model under the null hypothesis. We are assuming Poisson sampling, so we use R function `rpois` to simulate the data. This function vectorizes, so it works with `mu.hat` being a vector.
- The next three statements do exactly the same as the original analysis except that we use the simulated data rather than the real data.
- Then we extract the P-value from the result of R function `anova` and store it for future use.
- After the loop terminates, we have `nboot` simulations from the distribution of the P-value under the null hypothesis. We also have one more (not simulated) P-value that has the same distribution under the null hypothesis as the simulated P-values. This is `p.value.hat`, the value for the real data.
- The estimated P-value is then the fraction of the time the bootstrap simulated P-values `p.value.star` are at least as extreme as the P-value for the actual data `p.value.hat`.
- The last statement calculates the Monte Carlo standard error, how far off our simulation is from what we would get if we did an infinite number of simulations.

The idea is that, when the asymptotics cannot be trusted, all we can say about a P-value is that is can be used as a test statistic. We take lower values as stronger evidence against the null hypothesis, but we don't know that it is correctly calibrated: we don't think that we would have $P \le \alpha$ with probability α for all α under the null hypothesis. So we need to simulate the distribution of this test statistic to compute a valid P-value. And that is what this does. Call the observed value of this test statistic $\widehat{P}$ and the simulated value of this test statistic P^*, then $\Pr(P^* \le \widehat{P})$ is the valid P-value.

The trick of including the observed P-value among the simulated P-values guarantees that we cannot get a spurious bootstrap P-value of zero by making the Monte Carlo sample size too small. The smallest bootstrap P-value we can get by this procedure is `1 / (nboot + 1)`.

This trick also comes with an argument that it gives an exact randomized test for any significance level that is a multiple of `1 / (nboot + 1)`. Under the null hypothesis, it has the discrete uniform distribution on the numbers `seq(1, nboot + 1) / (nboot + 1)`.

Assuming Multinomial Sampling

When we simulate, we are not using asymptotics. And we are using exact sampling distributions. Thus Poisson sampling and multinomial sampling are not equivalent (their equivalence is an asymptotic result).

So we can do the same simulation under multinomial sampling.

```
n <- sum(counts)
p.hat <- mu.hat / n
p.value.star <- double(nboot)
for (iboot in 1:nboot) {
    # simulate new data from fitted model
    counts.star <- rmultinom(1, n, prob = p.hat)
    gout.indep.star <- glm(counts.star ~ malformation + drinks,
        family = poisson)
    gout.sat.star <- glm(counts.star ~ malformation * drinks,
        family = poisson)
    aout.star <- anova(gout.indep.star, gout.sat.star, test = "LRT")
    p.value.star[iboot] <- aout.star[2, "Pr(>Chi)"]
}
all.p.values <- c(p.value.star, p.value.hat)
mean(all.p.values <= p.value.hat)
```

```
## [1] 0.139
```

```
sd(all.p.values <= p.value.hat) / sqrt(nboot)
```

```
## [1] 0.01095074
```

The only difference between this simulation and the one before is the line

```
    counts.star <- rmultinom(1, n, prob = p.hat)
```

and the lines defining `n` to be the multinomial sample size and `p.hat` to be the estimated multinomial parameter vector so we can use it in the statement just above.