Stat 5421 Lecture Notes: To Accompany Agresti Ch 9

Charles J. Geyer

August 30, 2023

Contents

1 Section 9.1 Two-Way Tables **1**
1.1 Section 9.1.1 Only Main Effects . . . 1
1.2 Section 9.1.3 Interactions . . . 2
1.3 Section 9.1.5 Hierarchical Models . . . 2
1.4 Identifiability . . . 2

2 Section 9.2 Three-Way Tables **6**
2.1 Section 9.2.1 Only Main Effects . . . 6
2.2 The Saturated Model . . . 6
2.3 Example: High School Student Survey . . . 6
2.4 Interactions . . . 8
2.5 Many Models . . . 8

3 Section 9.4 Four-Way and Beyond **10**
3.1 Example: Seat-Belt Use . . . 10

4 The Parametric Bootstrap **12**
4.1 Relevant and Irrelevant Simulation . . . 12
4.2 R Packages and Textbooks . . . 12
4.3 Kinds of Bootstrap . . . 13
4.4 The Bootstrap Analogy . . . 13
4.5 Return to Our Example . . . 19

Bibliography **25**

1 Section 9.1 Two-Way Tables

1.1 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).

1.2 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.

1.3 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.

1.4 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 on identifiability) 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 Agresti Chapter 1, section on sampling schemes and also notes on sampling schemes) 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
```

That's ugly. Can we force `xtabs` to keep the `drinks` variable in the right order?

```
drinks <- factor(drinks, levels = c("0", "< 1", "1-2", "3-5", ">= 6"))
foo <- xtabs(counts ~ malformation + drinks)
foo
```

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

Better, now back to statistics

```
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 = 0.03648
```

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)
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)          9.74479    0.00765 1273.86   <2e-16 ***
## malformationPresent -5.85581    0.10384  -56.39   <2e-16 ***
## drinks< 1           -0.16561    0.01129  -14.67   <2e-16 ***
## drinks1-2           -3.07183    0.03632  -84.57   <2e-16 ***
## drinks3-5           -4.90346    0.08906  -55.05   <2e-16 ***
## drinks>= 6          -6.11007    0.16240  -37.62   <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)
##
## Coefficients:
##                                 Estimate Std. Error  z value Pr(>|z|)
## (Intercept)                     9.744843   0.007655 1273.036   <2e-16 ***
## malformationPresent            -5.873642   0.144540  -40.637   <2e-16 ***
## drinks< 1                      -0.165425   0.011302  -14.637   <2e-16 ***
## drinks1-2                      -3.075345   0.036437  -84.402   <2e-16 ***
## drinks3-5                      -4.908562   0.089415  -54.896   <2e-16 ***
## drinks>= 6                     -6.133926   0.164577  -37.271   <2e-16 ***
## malformationPresent:drinks< 1  -0.068189   0.217432   -0.314   0.7538
## malformationPresent:drinks1-2   0.813582   0.471340    1.726   0.0843 .
## malformationPresent:drinks3-5   1.037361   1.014307    1.023   0.3064
## malformationPresent:drinks>= 6  2.262725   1.023674    2.210   0.0271 *
## ---
## 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: -1.5508e-12  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 tests are asymptotically equivalent, but here the sample size is not "large". The total number of subjects (32574) 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           0         < 1        1-2         3-5       >= 6
##      Absent  17065.13888 14460.59624 790.735955 126.6374102 37.8915086
##      Present    48.86112    41.40376   2.264045   0.3625898  0.1084914
```

- 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< 1"
## [4] "drinks1-2"           "drinks3-5"           "drinks>= 6"
```

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< 1"                      "drinks1-2"
##  [5] "drinks3-5"                      "drinks>= 6"
##  [7] "malformationPresent:drinks< 1"  "malformationPresent:drinks1-2"
```

```
##  [9] "malformationPresent:drinks3-5"  "malformationPresent:drinks>= 6"
```

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.

2 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

2.1 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).

2.2 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.

2.3 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.

2.4 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.

2.5 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.

3 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.

3.1 Example: Seat-Belt Use

These data are given in Table 9.8 in Agresti. They 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   formula
##   182.8      0.24105  count ~ seat.belt*injury*location + seat.belt*location*gender + injury*locatio
##   183.1      0.21546  count ~ injury*gender + seat.belt*injury*location + seat.belt*location*gender
##   184.0      0.13742  count ~ seat.belt*injury + seat.belt*location*gender + injury*location*gender
##   184.8      0.09055  count ~ seat.belt*injury*location + seat.belt*injury*gender + seat.belt*locati
##   184.9      0.08446  count ~ seat.belt*injury + injury*location + injury*gender + seat.belt*locatio
##   185.0      0.08042  count ~ seat.belt*injury*location + seat.belt*injury*gender + seat.belt*locati
##   185.5      0.06462  count ~ seat.belt*injury*location*gender
##   185.8      0.05365  count ~ seat.belt*injury*gender + seat.belt*location*gender + injury*location*g
##   186.8      0.03237  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.

4 The Parametric Bootstrap

4.1 Relevant and Irrelevant Simulation

4.1.1 Irrelevant

Most statisticians think a statistics paper isn't really a statistics paper or a statistics talk isn't really a statistics talk if it doesn't have simulations demonstrating that the methods proposed work great (at least on some toy problems).

IMHO, this is nonsense. Simulations of the kind most statisticians do *prove* nothing. The toy problems used are often very special and do not stress the methods at all. In fact, they may be (consciously or unconsciously) chosen to make the methods look good.

In scientific experiments, we know how to use randomization, blinding, and other techniques to avoid biasing the results. Analogous things are never AFAIK done with simulations.

When all of the toy problems simulated are *very different* from the statistical model you intend to use for your data, what could the simulation study possibly tell you that is relevant? Nothing.

Hence, for short, your humble author calls all of these millions of simulation studies statisticians have done *irrelevant simulation*.

4.1.2 Relevant

But there is a well-known methodology of *relevant simulation*, except that it isn't called that. It is called the *bootstrap*.

It idea is, for each statistical model and each data set to which it is applied, one should do a simulation study of this model on data of this form.

But there is a problem: the fundamental problem of statistics, that $\hat{\theta}$ is not θ. To be truly relevant we should simulate from the true unknown distribution of the data, but we don't know what that is. (If we did, we wouldn't need statistics.)

So as a second best choice we have to simulate from our best estimate of the true unknown distribution, the one corresponding to the parameter value $\hat{\theta}$ if that is the best estimator we know.

But we know that is the Wrong Thing. So we have to be sophisticated about this. We have to arrange what we do with our simulations to come as close to the Right Thing as possible.

And bootstrap theory and methods are extraordinarily sophisticated with many different methods of coming very close to the Right Thing.

4.2 R Packages and Textbooks

There are two well known R packages concerned with the bootstrap. They go with two well known textbooks.

- R package `boot` (Canty and Ripley, 2021) is an R recommended package that is installed by default in every installation of R. As the package description says, it goes with the textbook Davison and Hinkley (1997).
- The CRAN package `bootstrap` (Tibshirani and Leisch, 2019) goes with, as its package description says, the textbook Efron and Tibshirani (1993).

  The package description also says that "new projects should preferentially use the recommended package '`boot`' ". But I do not agree. The package maintainer is neither of Efron or Tibshirani, and I do not think they would agree. Whatever the politics of the R core team that make the `boot` package "recommended", they have nothing to do with the quality of the package or with the quality of the textbook they go with. If you like Efron and Tibshirani (1993), you should be using the R package `bootstrap` that goes with it.

These authors range from moderately famous (for a statistician) to very, very famous (for a statistician). Efron is the inventor of the term *bootstrap* in its statistical meaning.

4.3 Kinds of Bootstrap

Almost all of the bootstrap literature is about the *nonparametric bootstrap* that does not assume any parametric statistical model for the data. Most of the content of the textbooks cited above, and all of the R packages cited above are about this methodology.

But if one is using parametric models, like everything we do in this course, then what you are doing is inherently parametric, even if you use the nonparametric bootstrap. Thus there is no benefit to using the nonparametric bootstrap for anything in this course.

A small minority of the bootstrap literature is about the *parametric bootstrap* that does use the parametric statistical model for its simulations. So this is all we will discuss here.

For more on the nonparametric bootstrap, see my Stat 3701 lecture notes, some of which is repeated here.

4.4 The Bootstrap Analogy

4.4.1 The Name of the Game

The term "bootstrap" recalls the English idiom "pull oneself up by one's bootstraps".

The literal meaning of "bootstrap" in non-technical language is leather loops at the top of boots used to pull them on. So the literal meaning of "pull oneself up by one's bootstraps" is to reach down, grab your shoes, and lift yourself off the ground — a physical impossibility. But, idiomatically, it doesn't mean do the physically impossible; it means something like "succeed by one's own efforts", especially when this is difficult.

The technical meaning in statistics plays off this idiom. It means to get a good approximation to the sampling distribution of an estimator without using any theory. (At least not using any theory in the computation. A great deal of very technical theory may be used in justifying the bootstrap in certain situations.)

4.4.2 The Real World and the Bootstrap World

The discussion in this section is stolen from Efron and Tibshirani (1993), their Figure 8.1 and the surrounding text.

To understand the bootstrap you have to understand a simple analogy. Otherwise it is quite mysterious. I recall being mystified about it when I was a graduate student. I hope the students I teach are much less mystified because of this analogy. This appears to the untutored to be impossible or magical. But it isn't really. It is sound statistical methodology.

Let P_θ denote the true unknown probability distribution that we assume the data are a sample from,

The bootstrap makes an analogy between the real world and a mythical bootstrap world.

	real world	bootstrap world
true unknown distribution	P_θ	$P_{\hat{\theta}_n}$
true unknown parameter	θ	$\hat{\theta}_n$
data	Y has distribution P_θ	Y^* has distribution $P_{\hat{\theta}_n}$
estimator	$\hat{\theta}_n = t(Y)$	$\theta_n^* = t(Y^*)$
estimated standard error	$s(\hat{\theta}_n)$	$s(\theta_n^*)$

	real world	bootstrap world
approximate pivotal quantity	$(\hat{\theta}_n - \theta)/s(\hat{\theta}_n)$	$(\theta^*_n - \hat{\theta}_n)/s(\theta^*_n)$

The explanation.

- In the real world we have the true unknown distribution of the data P_θ. In the bootstrap world we have the "true" pretend unknown distribution of the data $P_{\hat{\theta}_n}$. Actually the distribution $P_{\hat{\theta}_n}$ is known, and that's a good thing, because it allows us to simulate data from it. But we pretend it is unknown when we are reasoning in the bootstrap world. It is the analog in the bootstrap world of the true unknown distribution P_θ in the real world.
- In the real world we have the true unknown parameter θ. It is what we are trying to estimate. In the bootstrap world we have the "true" pretend unknown parameter $\hat{\theta}_n$. Actually the parameter $\hat{\theta}_n$ is known, and that's a good thing, because it allows to see how close simulated estimators come to it. But we pretend it is unknown when we are reasoning in the bootstrap world. It is the analog in the bootstrap world of the true unknown parameter θ in the real world.
- In the real world we have data Y that is assumed to have distribution P_θ. In the bootstrap world we simulate data Y^* that are known to have distribution $P_{\hat{\theta}_n}$.

  In the real world we have only one data set Y. We have to imagine its randomness – that, if the data could be collected again, it would turn out different. In the bootstrap world we can simulate as many different Y^* as we like. We can actually see the distribution by making plots and compute things about the distribution by averaging over the simulations.

  The way we simulate depends on what parametric model we are assuming. Core R has functions to simulate Poisson data (`rpois`) and multinomial data (`rmultinom`). It has no methods for product multinomial, but this can easily be done using `rmultinom` and a loop.
- We have some estimator of θ, which must be a *statistic*, that is some function of the data that does not depend on the unknown parameter. In order to have the correct analogy in the bootstrap world, our estimate there must be the *same function* of the *bootstrap data*.
- Many procedures require some estimate of standard error of $\hat{\theta}_n$. Call that $\hat{s}_n$. It too must be a *statistic*, that is some function of the data that does not depend on the unknown parameter. In order to have the correct analogy in the bootstrap world, our estimate there must be the *same function* of the *bootstrap data*.
- Many procedures use so-called *pivotal quantities*, either exact or approximate.

  An exact pivotal quantity is a function of the data and the parameter of interest whose distribution *does not depend on any parameters*. The prototypical example is the t statistic

  $$\frac{\overline{X}_n - \mu}{s_n/\sqrt{n}}$$

  which has, when the data are assumed to be exactly normal, an exact t distribution on $n-1$ degrees of freedom (which does not depend on the unknown parameters μ and σ of the distribution of the data). Note that the pivotal quantity is *a function of* μ but the sampling distribution of the pivotal quantity *does not depend on* μ *or* σ: the t distribution with $n-1$ degrees of freedom does not does not have any unknown parameters.

  An asymptotic pivotal quantity is a function of the data and the parameter of interest whose asymptotic distribution *does not depend on any parameters*. The prototypical example is the z statistic

  $$\frac{\overline{X}_n - \mu}{s_n/\sqrt{n}}$$

(actually the same function of data and parameters as the t statistic discussed above), which has, when the data are assumed to have any distribution with finite variance, an asymptotic standard normal distribution (which does not depend on the unknown the distribution of the data). Note that the pivotal quantity is *a function of* μ but the sampling distribution of the pivotal quantity *does not depend on the unknown distribution of the data*: the *standard* normal distribution does not does not have any unknown parameters.

An approximate pivotal quantity is a function of the data and the parameter of interest whose sampling distribution does not depend on the unknown distribution of the data, at least not very much. Often such quantities are made by standardizing in a manner similar to those discussed above. Any time we have some purported standard errors of estimators, we can use them to make approximate pivotal quantities.

$$\frac{\hat{\theta}_n - \theta}{\hat{s}_n}$$

as in the bottom left cell of the table above.

The importance of pivotal quantities in (frequentist) statistics cannot be overemphasized. They are what allow valid exact or approximate inference. When we invert the pivotal quantity to make confidence intervals, for example,

$$\hat{\theta}_n \pm 1.96 \cdot \hat{s}_n$$

this is (exactly or approximately) valid *because the sampling distribution of the pivotal quantity does not depend on the true unknown distribution of the data, at least not much.* If it did depend strongly on the true distribution of the data, then our coverage could be way off, because our estimated sampling distribution of the pivotal quantity might be far from its correct sampling distribution.

More on pivotal quantities below. There are many pivotal quantities that one can use, not just those having the form of the bottom line of the table.

4.4.3 Caution: Use the Correct Analogies

In the bottom right cell of the table above there is a strong tendency for naive users to replace $\hat{\theta}_n$ with θ. But this is clearly incorrect. What plays the role of true unknown parameter value in the bootstrap world is $\hat{\theta}_n$ not θ. The distribution that bootstrap data are simulated from has true known parameter value $\hat{\theta}_n$.

There is a lesser tendency for naive users to replace $s(\theta_n^*)$ with $s(\hat{\theta}_n)$. But this is clearly incorrect. In the real world we know that $s(\hat{\theta}_n)$ is a random quantity. We are trying to account for this randomness in the same way a t distribution accounts for the randomness in the sample standard deviation. So we must use the quantity $s(\theta_n^*)$ in the bootstrap world that has similar randomness.

What one does in the bootstrap world must always perfectly mimic what one does in the real world with the sole exception that $\hat{\theta}$ is not θ.

4.4.4 Pivotal Quantities and Infinite Regress

Frequentist statistics has a problem of infinite regress.

- Suppose you are trying estimate the population mean μ.
- Naturally, you use the sample mean $\bar{x}_n$ as an estimator.
- If you want to know how good an estimator, the standard deviation of the estimator, which is $\sigma/\sqrt{n}$, tells you.
- But you don't know the population standard deviation σ either, so that's no help.
- So we estimate σ by the sample standard deviation $\hat{\sigma}_n$.
- But we still have a problem. We don't know how far $\hat{\sigma}_n$ is from σ.

- We can ask about the standard deviation of $\hat{\sigma}_n$, but theory can't tell us much about that. Too complicated. Theory can tell us the asymptotic variance of $\hat{\sigma}_n^2$

$$\sqrt{n}(\hat{\sigma}_n^2 - \sigma^2) \xrightarrow{\mathcal{D}} \text{Normal}(0, \mu_4 - \sigma^4)$$

  where μ_4 is the population fourth central moment.
- But you don't know μ_4 or σ either, so that's no help.
- So we estimate μ_4 by the sample fourth central moment

$$\hat{\mu}_{4,n} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x}_n)^4$$

- But we still have a problem. We don't know how far $\hat{\mu}_{4,n}$ is from μ_4.
- Theory can tell us

$$\sqrt{n}(\hat{\mu}_{4,n} - \mu_4) \xrightarrow{\mathcal{D}} \text{Normal}(0, \mu_8 - \mu_4^2 - 8\mu_3\mu_5 + 16\mu_3^2\mu_2)$$

- But now we have a bunch more unknown parameters to estimate.
- And this process goes on without end.

The method of pivotal quantities and asymptotic pivotal quantities is the idea that short circuits this infinite regress.

If we just standardize the quantity we are trying to get an asymptotic distribution of, then the infinite regress goes away

$$\frac{\bar{x}_n - \mu}{\hat{\sigma}_n / \sqrt{n}} \xrightarrow{\mathcal{D}} \text{Normal}(0, 1)$$

and the asymptotic distribution has no unknown parameters to estimate (that's what asymptotic pivotal quantity means).

But asymptotic pivotal quantities don't have to come from standardization. We know that the Wald, Wilks, and Rao test statistics are all asymptotically chi-square, and the chi-square distribution does not have any unknown parameters to estimate. So those are other asymptotically pivotal quantities.

4.4.5 How the Parametric Bootstrap Works

4.4.5.1 Hypothesis Tests

So the first thing we have to do is select an estimator of the parameter under the null hypothesis (because that is the distribution we are supposed to use when computing a hypothesis test). So let $\hat{\theta}_n$ be the MLE under the null hypothesis (or any other good estimate).

The next thing we have to do is select an asymptotically pivotal quantity, Say the likelihood ratio test statistic.

Now we simulate many datasets Y^* from the distribution of the data under $\hat{\theta}_n$. And we use these simulations to approximate the sampling distribution of the test statistic. We do not use the chi-square distribution! That is only good when n is nearly infinite. Instead we use the bootstrap distribution (simulation distribution). If the test statistic is $t(Y)$ and is asymptotically pivotal, then our bootstrap P-value is $\Pr\{t(Y^*) \geq t(Y)\}$, and this probability is approximated by averaging over our bootstrap simulations.

Actually we don't recommend the naive bootstrap P-value just described but rather a slightly more sophisticated bootstrap P-value that takes into account that, asymptotically at least, $t(Y^*)$ and $t(Y)$ have the same

distribution (which is the whole basis of the procedure). So if we have n_{boot} simulated test statistics $t(Y_1^*)$, $t(Y_2^*)$, ..., $t(Y_{n_{\text{boot}}}^*)$, then we also have one more $t(Y)$, and we should use

$$\frac{\#\{t(Y^*) \geq t(Y)\} + 1}{n_{\text{boot}} + 1}$$

for the bootstrap P-value, where # in the numerator means number of times the inequality holds. This corrects for n_{boot} being too small. The smallest a bootstrap P-value can be is $1/n_{\text{boot}}$, so you have to do a lot of bootstrap simulations to get a really small bootstrap P-value.

4.4.5.2 Confidence Intervals

There are many different schemes for nonparametric bootstrap confidence intervals. There are fewer for the parametric bootstrap, and discussions of the parametric bootstrap in textbooks are so sketchy that we do not get clear recommendations. Hence we offer the following, which is the analogue for the parametric bootstrap of what are called bootstrap t procedures for the nonparametric bootstrap.

The first thing we have to do is select an estimator of the parameter. Call that $\hat{\theta}_n$.

The next thing we have to do is select an estimator of the asymptotic standard deviation of the parameter. Like $\hat{\theta}_n$ this estimator depends on the data and hence is a random variable.

If we are doing Wald confidence intervals, and $\hat{\theta}_n$ is the MLE, then the asymptotic variance of the MLE is inverse Fisher information. If $\hat{\theta}_n$ is a vector parameter, then inverse Fisher information is a matrix. But then the diagonal elements of this matrix are the estimated variances of the individual components of $\hat{\theta}_n$. We saw this in the notes on likelihood computation. Since the estimated asymptotic standard deviation depends on the parameter, let us denote it $s(\theta)$. Then we have our estimate for the actual data $s(\hat{\theta}_n)$ and our estimate for bootstrap data is $s(\theta_i^*)$.

Now, as always in the parametric bootstrap, we simulate n_{boot} IID realizations of the data from the distribution with parameter $\hat{\theta}_n$. The parameter estimates for these simulated data sets are, as we said above θ_i^*, $i = 1, \ldots, n_{\text{boot}}$. Now we can form n_{boot} simulations of the asymptotically pivotal quantity

$$z_i^* = \frac{\theta_i^* - \hat{\theta}_n}{s(\theta_i^*)}$$

just like in the bottom row of our table about the real world and the bootstrap world.

Now comes the *tour de force*. We don't use the normal distribution to determine the critical value, in fact, we don't even have one critical value but rather two. We use the bootstrap distribution of the asymptotic pivotal quantity rather than its asymptotic distribution. We determine *from the bootstrap simulations* numbers c_1 and c_2 such that

$$c_1 \leq z_i^* \leq c_2 \text{ exactly } 1 - \alpha \text{ of the time}$$

to get coverage $1 - \alpha$. For example, to get an equal-tailed 95% confidence interval, we choose c_1 to be the 0.025 quantile of the bootstrap distribution of the z_i^* and choose c_2 to be the 0.975 quantile.

The point is that we use the bootstrap rather than the asymptotics to find critical values.

Then our bootstrap confidence interval is

$$(\hat{\theta}_n - c_2 s(\hat{\theta}_n), \hat{\theta}_n - c_1 s(\hat{\theta}_n)) \tag{4}$$

That is, we apply the critical values (which are exact in the bootstrap world) to the real world (where they are only the closest we can do to the Right Thing).

Note that our bootstrap confidence intervals do not have the form

$$\text{point estimate} \pm \text{critical value} \times \text{standard error}$$

which some intro stats books teach is the only kind of confidence interval that exists.

Of course we already have seen that likelihood confidence intervals and score (Rao) confidence intervals, don't have this form either.

But now we see that bootstrap Wald intervals also don't have this form.

If the bootstrap distribution of the z_i^* is skewed, then our bootstrap t confidence intervals are skewed to account for that.

But even if the bootstrap distribution of the z_i^* is not skewed, our bootstrap t confidence intervals account for the variability of estimating the asymptotic standard deviation. Even if "Student" (W. S. Gosset) had never invented the t distribution (nor anyone else), this bootstrap t procedure would invent it. When the population distribution is normal and we do a parametric bootstrap, the bootstrap distribution of the z_i^* is Student's t distribution on $n - 1$ degrees of freedom. So we do t tests without knowing any theory. The bootstrap does all the theory for us.

Moreover, when the true unknown population distribution is not normal, the bootstrap figures out the analogue of the "Student" t test for that population distribution, something theoretical statistics cannot do.

At least, it does all the theory relevant to the true unknown population distribution. As we have seen, the description of the bootstrap does involve some theory.

Actually we don't recommend the bootstrap confidence intervals just described, or rather we recommend them but only using a very specific method of computing bootstrap critical values. Suppose we choose n_{boot} so that the quantiles we want when multiplied by $n_{\text{boot}} + 1$ are integers. Then we take

$$\begin{aligned} c_1 &= z^*_{(n_{\text{boot}}+1)\alpha/2} \\ c_2 &= z^*_{(n_{\text{boot}}+1)(1-\alpha/2)} \end{aligned}$$

Like the procedure we actually recommend for bootstrap P-values, this procedure has +1 to correct for n_{boot} being small. But the argument is different.

The main virtue of this recommendation is that it avoids arguments about quantile estimation. R function `quantile` has 9 different procedures. None of them are as simple or as good as the procedure recommended here. Of course our procedure works only for specially chosen n_{boot} whereas R function `quantile` has to work for all sample sizes. So this procedure cannot replace R function `quantile` except in this particular application.

4.4.6 Another Infinite Regress

But we don't want to use the parametric bootstrap only when the sample size is infinite! So we still have an infinite regress, but a more tractable one.

- We decide to use the bootstrap because we are worried that the asymptotics might not "work" for the actual sample size of our actual data.
- So what if we are worried about whether the bootstrap works? Bootstrap the bootstrap! This is called a double bootstrap.
- So what if we are worried about whether the double bootstrap works? Bootstrap the double bootstrap! This is called a triple bootstrap.
- And so forth.

The double bootstrap can be useful (Geyer, 1991; Newton and Geyer, 1994) but is rather complicated.

It is easiest to explain for hypothesis tests. When we are worried about the bootstrap, we are worried that the bootstrap distribution of the test statistic is not exactly the same as the real world distribution of the test statistic. So our bootstrap P-values are not exactly correct (even if we have humongously large n_{boot}).

So our bootstrap P-values get demoted to being just another test statistic. We take them as evidence against the null hypothesis when they are near zero, but we no longer believe the bootstrap distributions for them. But what do we do when we doubt the distribution of the test statistic? Bootstrap it. So now we get double bootstrap P-values

$$\frac{\#\{t(Y^{**}) \ge t(Y^*)\} + 1}{n_{\text{boot}} + 1}$$

the same as our bootstrap P-values except one level up. For each bootstrap estimate θ^* we are doing a bootstrap inside a bootstrap to simulate many realizations Y^{**} of the data from the distribution having parameter value θ^* and using them to calculate this double bootstrap P-value. Geyer (1991) gives details.

Geyer (1991) makes another important point. The bootstrap not only provides better, more accurate hypothesis tests and confidence intervals, but also provides a diagnostic tool.

- If the bootstrap gives more or less the same answers as the asymptotics, then you didn't need the bootstrap.
- If the double bootstrap gives more or less the same answers as the single bootstrap, then you didn't need the double bootstrap.

Except you need the bootstrap or double bootstrap to reach these conclusions.

I don't know if anyone has actually done a triple bootstrap.

Newton and Geyer (1994) show how to speed up the calculation of a double bootstrap.

But we leave this subject here. We won't actually do any examples of the double bootstrap.

4.4.7 Parametric versus Nonparametric Bootstrap

The nonparametric bootstrap has trouble doing hypothesis tests for any data, it is mostly for confidence intervals. The nonparametric bootstrap has trouble doing any analysis of regression data. Both of these are explained in my Stat 3701 lecture notes.

The parametric bootstrap has neither of these problems. This gives another reason for always using the parametric bootstrap for parametric statistical models.

4.5 Return to Our Example

4.5.1 Bootstrap Hypothesis Tests

4.5.1.1 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.

Here we use the P-values rather than the test statistics as asymptotically pivotal quantities, but otherwise this is as described above.

4.5.1.2 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 exactly equivalent (their equivalence is an asymptotic result).

So we can do the same simulation under multinomial sampling.

```
n <- sum(counts)
pi.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 = pi.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 = pi.hat)
```

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

4.5.1.3 Assuming Product Multinomial Sampling

For these data the Poisson sampling model seems to be correct, but just for the sake of showing what to do if the sample sizes for `malformation == "Absent"` and `malformation == "Present"` were fixed in advance.

```
# Note mu.hat the same regardless of Poisson or product multinomial sampling
nAbsent <- sum(counts[malformation == "Absent"])
nPresent <- sum(counts[malformation == "Present"])
pi.hat.Absent <- mu.hat[malformation == "Absent"] / nAbsent
pi.hat.Present <- mu.hat[malformation == "Present"] / nPresent
p.value.star <- double(nboot)
for (iboot in 1:nboot) {
    # simulate new data from fitted model
    counts.star <- double(length(pi.hat))
    counts.star[malformation == "Absent"] <-
        rmultinom(1, nAbsent, prob = pi.hat.Absent)
    counts.star[malformation == "Present"] <-
        rmultinom(1, nPresent, prob = pi.hat.Present)
    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] 0.129
```

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

```
## [1] 0.01061056
```

4.5.2 Bootstrap Confidence Intervals

```
conf.level <- 0.95
```

We won't do all three sampling schemes, just Poisson. The changes for multinomial or product multinomial should be obvious from the above.

We will do Wald intervals for the cell probabilities in the bottom row of the table (for `malformation == "Present"`).

```
ilow <- (nboot + 1) * (1 - conf.level) / 2
ihig <- (nboot + 1) * (1 - (1 - conf.level) / 2)
c(ilow, ihig)
```

```
## [1]  25 975
```

```
imalf <- malformation == "Present"
imalf
```

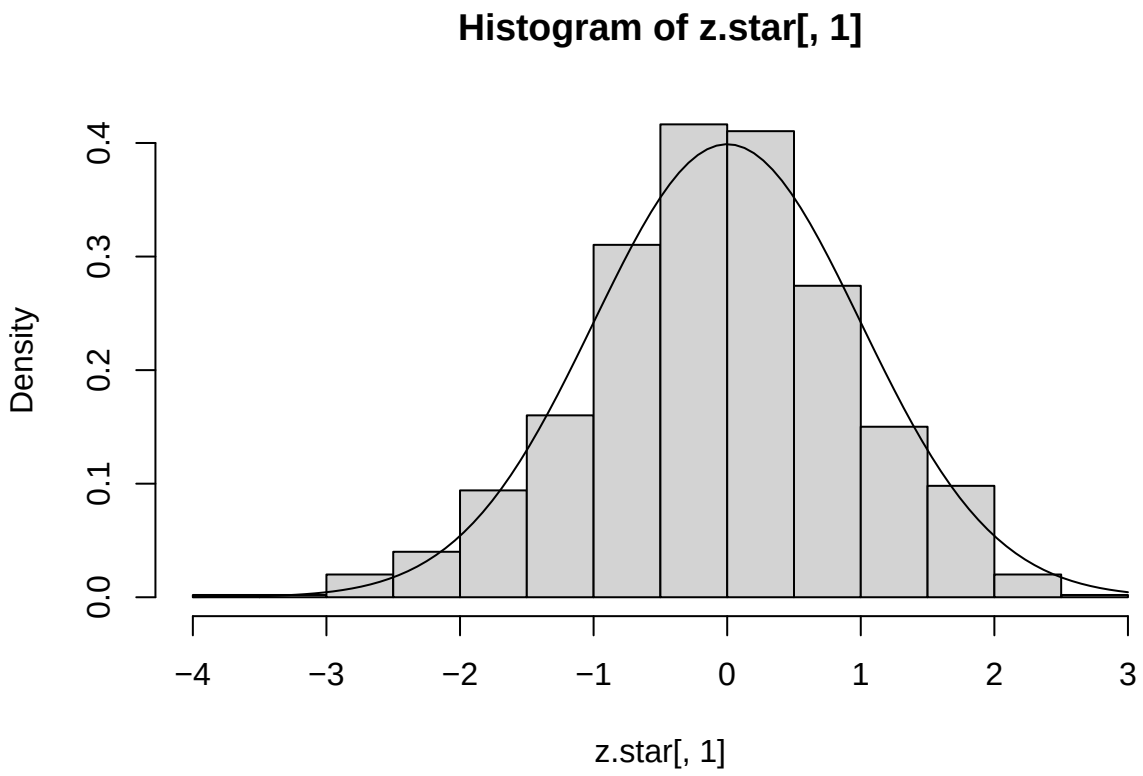
```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
# room for all simulations
z.star <- matrix(NaN, nrow = nboot, ncol = sum(imalf))

for (iboot in 1:nboot) {
    # simulate new data from fitted model
    counts.star <- rpois(length(mu.hat), lambda = mu.hat)
    # fit model to bootstrap data
    gout.star <- glm(counts.star ~ malformation + drinks, family = poisson)
    # get estimates and standard errors
    pout.star <- predict(gout.star, type = "response", se.fit = TRUE)
    z.star[iboot, ] <-
        (pout.star$fit[imalf] - mu.hat[imalf]) / pout.star$se.fit[imalf]
}
```

At this point each column of the matrix `z.star` is the bootstrap distribution of the asymptotic pivotal quantity for one of our estimates. Let's look at the first of these distributions.

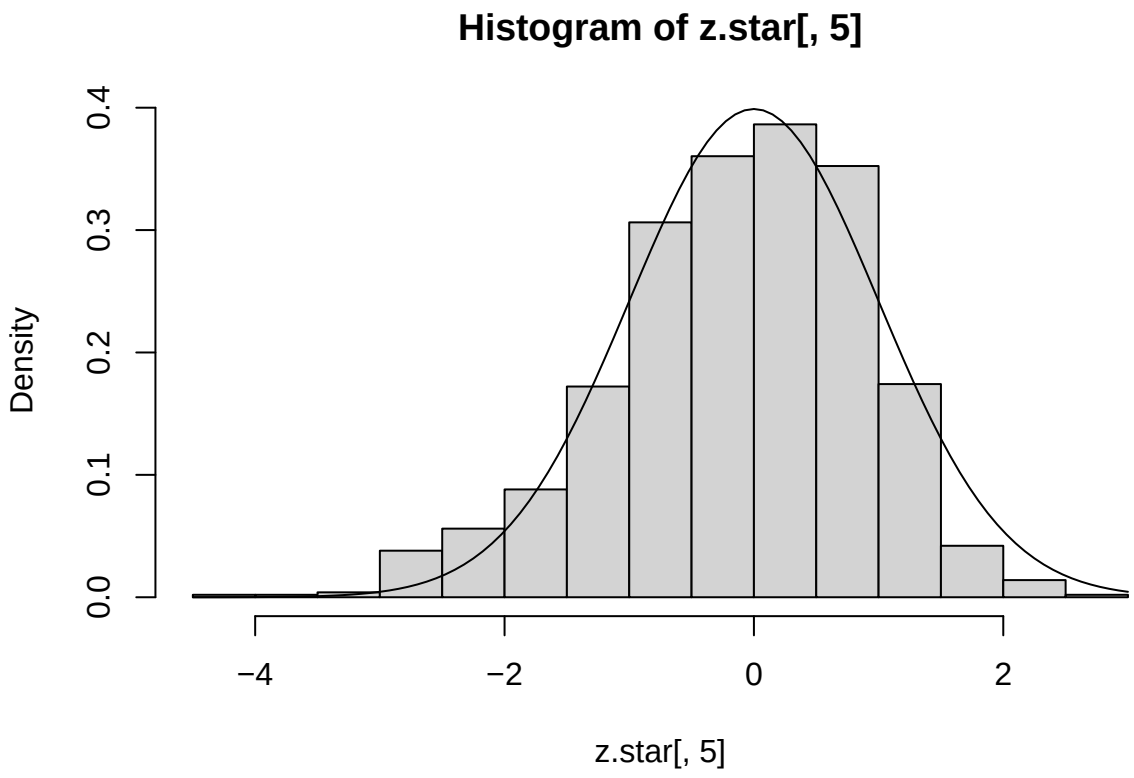
```
hist(z.star[ , 1], probability = TRUE)
curve(dnorm, add = TRUE)
```



Hard to tell whether this is different from standard normal or not.

Let's look at the last one.

```
hist(z.star[ , 5], probability = TRUE)
curve(dnorm, add = TRUE)
```



This is more obviously skewed.

Now extract the bootstrap critical values.

```
crit <- apply(z.star, 2, function(x)
    sort.int(x, partial = c(ilow, ihig))[c(ilow, ihig)])
colnames(crit) <- c("0", "< 1", "1-2", "3-5", ">= 6")
rownames(crit) <- c("lower", "upper")
crit
```

```
##                0       < 1       1-2       3-5      >= 6
## lower -2.053954 -2.048708 -2.286767 -2.257690 -2.406881
## upper  1.837185  1.839756  1.731685  1.756343  1.569598
```

The fact that none of these are ± 1.96 means the bootstrap is making some adjustment for skewness and perhaps for heavy tails, whether or not we can detect skewness in the histogram. So it seems the parametric bootstrap is working better than asymptotics (which would replace all of these critical values with ± 1.96).

So now we can make the confidence intervals

```
# fit model to real data
gout <- glm(counts ~ malformation + drinks, family = poisson)
# get estimates and standard errors
pout <- predict(gout, type = "response", se.fit = TRUE)
foo <- rbind(lower = pout$fit[imalf] - crit[2, ] * pout$se.fit[imalf],
             upper = pout$fit[imalf] - crit[1, ] * pout$se.fit[imalf])
colnames(foo) <- colnames(crit)
foo
```

```
##              0      < 1      1-2       3-5       >= 6
## lower 39.54073 33.49097 1.834867 0.2757468 0.07571918
## upper 59.28122 50.21525 2.830793 0.4742220 0.15874564
```

For comparison, here are the asymptotic intervals.

```
foo <- outer(c(-1, 1) * qnorm(0.975), pout$se.fit[imalf])
colnames(foo) <- c("0", "< 1", "1-2", "3-5", ">= 6")
rownames(foo) <- c("lower", "upper")
foo <- sweep(foo, 2, pout$fit[imalf], "+")
foo
```

```
##              0      < 1      1-2       3-5       >= 6
## lower 38.91785 32.97395 1.778291 0.2656787 0.06756858
## upper 58.80439 49.83356 2.749799 0.4595009 0.14941429
```

The bootstrap doesn't make a big difference. But it is better than the asymptotics. Not a lot different, but different.

Bibliography

Canty, A. and Ripley, B. D. (2021) *R Package* `boot`*: Bootstrap R Functions, Version 1.3-28.*

Davison, A. C. and Hinkley, D. V. (1997) *Bootstrap Methods and Their Application.* Cambridge: Cambridge University Press.

Efron, B. and Tibshirani, R. (1993) *An Introduction to the Bootstrap.* New York: Chapman & Hall.

Geyer, C. J. (1991) Constrained maximum likelihood exemplified by isotonic convex logistic regression. *Journal of the American Statistical Association*, **86**, 717–724.

Newton, M. A. and Geyer, C. J. (1994) Bootstrap recycling: A Monte Carlo algorithm for the nested bootstrap. *Journal of the American Statistical Association*, **89**, 905–912.

Tibshirani, R. and Leisch, F. (2019) *R Package* `bootstrap`*: Functions for the Book* An Introduction to the Bootstrap*, Version 2019.6.* Available at: https://CRAN.R-project.org/package=bootstrap.