Introduction to Simulations in R

Charles DiMaggio, PhD, MPH, PA-C

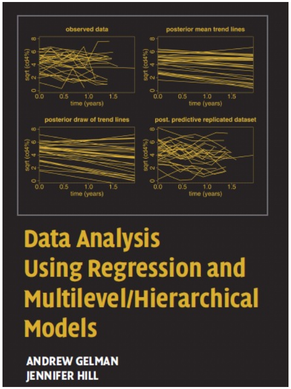
New York University Department of Surgery and Population Health
NYU-Bellevue Division of Trauma and Surgical Critical Care

June 10, 2015

- `http://www.columbia.edu/~cjd11/charles_dimaggio/DIRE/`
- Charles.DiMaggio@nyumc.org

Outline

This material has been shamelessly stolen.


observed data
posterior mean trend lines
posterior draw of trend lines
post. predictive replicated dataset
time (years)
sqrt (cd4%)
Data Analysis Using Regression and Multilevel/Hierarchical Models
ANDREW GELMAN
JENNIFER HILL

Buy and read this book!

Gellman and Hill, "Data Analysis Using Regression and Mulitlevel/Hierarchical Models", Cambridge University Press, 2007. (mostly chapters 7 and 8).

`http://www.stat.columbia.edu/~gelman/arm/`

Outline

sample()

simple random sample

```
sample(c("H","T"), size = 8, replace = TRUE)  # fair coin
sample(1:6, size = 2, replace = TRUE, prob=c(3,3,3,4,4,4)) #lo
```

- *replace=TRUE* to over ride the default sample without replacement
- *prob=* to sample elements with different probabilities, e.g. over sample based on some factor
- the *set.seed()* function allow you to make a reproducible set of random numbers.

probability distributions in R

- beta(shape1, shape2, ncp)
- binom(size, prob)
- chisq(df, ncp)
- exp(rate)
- gamma(shape, scale)
- logis(location, scale)
- norm(mean, sd)
- pois(lambda)
- t(df, ncp)
- unif(min, max)

convention for using probability functions in R

- *dxxx(x,)* returns the density or the value on the y-axis of a probability distribution for a discrete value of x
- *pxxx(q,)* returns the cumulative density function (CDF) or the area under the curve to the left of an x value on a probability distribution curve
- *qxxx(p,)* returns the quantile value, i.e. the standardized z value for x
- **rxxx(n,) returns a random simulation of size n**

```
qnorm(0.025)
qnorm(1-0.025)
```

sampling from probability distributions

```
rnorm(6) #  6 std nrml distribution values
rnorm(10, mean = 50, sd = 19) # set parameters
runif(n = 10, min = 0, max = 1) #uniform distribution
rpois(n = 10, lambda = 15) # Poisson distribution
 # toss coin 8 times using binomial distribution
 rbinom(n = 8, size = 1, p = 0.5)
 rbinom(8,1,.5) # args correct order
 # 18 trials, sample size 10, prob success =.2  rbinom(18, 10,
```

Exercise 1: Sampling and Simulations

Outline

bootstrapping a relative risk

- Use simulations to approximate results when no direct or closed solution
- E.g. Risk Ratios.
    - Use *rbinom()* to simulate (many times) rates of disease in exposed and unexposed populations
    - Divide results by the number of simulations and use the mean and 0.025 tails for the point estimate and confidence limits.

approach to bootstrapping a relative risk

1. simulate 5000 replicate bernoulli trials in sample size n_1 = exposed
2. divide those results by n_1 to get 5000 simulated risk estimates for the exposed group
3. repeat that process for the unexposed group n_2
4. divide 5000 simulated risks in exposed by 5000 simulated risks in unexposed to get 5000 simulate relative risks
5. calcualate mean and 0.25 tails from that population

Example: ASA and MI

RR calculation using *epitab()*

- Hennekens, 1987 study protective benefits of aspirin.
- 104 myocardial infarctions (fatal and non-fatal) among 11,037 people in the treatment group
- 189 MI's among 11,034 people in the placebo group
- Calculate RR and CI using with log-approximated approach (*epitools::epitab()*)

```
library(epitools)
asa.tab<- matrix(c(104,11037,189,11034),2,2)
epitab(asa.tab, method="riskratio")
```

simulate RR estimate

- use *rbinom()* to repeat 5,000 times an experiment where we count the number of outcomes (MI's) in two populations
    - probability of the outcome in a population defined by the results of the Hennekens study
- for each replicate, divide the number outcomes by number of people in each population to get 5,000 risk estimates for each group (treatment and placebo)
- calculate the RR for each simulation
- collect and describe results

```
set.seed(151)
tx <- rbinom(5000, 11037, 104/11037)
plac <- rbinom(5000, 11034, 189/11034)

r.tx<-tx/11037
r.plac<-plac/11034
rr.sim <- r.tx/r.plac

mean(rr.sim)
quantile(rr.sim, c(0.025, 0.975))
sd(rr.sim)
```

Try writing a function to calculate bootstrap estimates of relative risks. Test the function using the aspirin example.

Outline

some simple simulations

birth gender

- predict number of girls in 400 births, where prob of female birth is 48.8%

```
 n.girls<-rbinom(1,400, .488)
 n.girls
```

- to get distribution of the simulations, repeat the simulation many times

```
n.sims<-1000
n.girls<-rbinom(n.sims, 400, .488)
hist(n.girls)
```

- can do same thing with a loop
    - vectorized operation preferred in R, but loops useful in BUGS

```
n.sims<-1000
n.girls<-rep(NA, n.sims) # create vector to store simulations
for (i in 1:n.sims){
n.girls[i]<-rbinom(1,400,0.488)
}
hist(n.girls)
```

more complex simulations

account for twins

- 1/125 chance fraternal twins, each with 49.5% chance being girl
- 1/300 chance identical twins, 49.5% chance of both being girls.

```
birth.type <- sample (c("fraternal twin","identical twin","single birth"),
    size=400, replace=TRUE, prob=c(1/125, 1/300, 1 - 1/125 - 1/300))
girls <- rep (NA, 400)
for (i in 1:400){
  if (birth.type[i]=="single birth"){
 girls[i] <- rbinom (1, 1, .488)}
  else if (birth.type[i]=="identical twin"){
 girls[i] <- 2*rbinom (1, 1, .495)}
  else if (birth.type[i]=="fraternal twin"){
 girls[i] <- rbinom (1, 2, .495)}
}
n.girls <- sum (girls)
```

- vectorized version of the loop

```
girls <- ifelse (birth.type=="single birth", rbinom (400, 1, .488),
       ifelse ( birth.type=="identical twins", 2*rbinom (400, 1, .495),
        rbinom (400, 2, .495)))
```

using replicate()

repeat the simulation many times

```
girl.sim<-function(x){
birth.type <- sample (c("fraternal twin","identical twin","single birth"),
    size=x, replace=TRUE, prob=c(1/125, 1/300, 1 - 1/125 - 1/300))
girls <- ifelse (birth.type=="single birth", rbinom (400, 1, .488),
       ifelse ( birth.type=="identical twins", 2*rbinom (400, 1, .495),
        rbinom (400, 2, .495)))
return(sum(girls))
     }

girl.sim(400)

my.sims<-replicate(1000, girl.sim(400))
hist(my.sims)
```

Exercise 2: Using Simulation to Draw Statistical Inferences

confidence intervals

e.g. ratio of two proportions

- Survey 1000 people, 500 men and 500 women.
    - 75% men support death penalty, 65% women.
    - Ratio of men to women is $0.75/0.65 = 1.15$.
- Computing a standard error for this ratio can be challenging.
- Simulate the s.e. $\sqrt{\frac{p \cdot q}{n}}$

confidence intervals

doing the simulation

```
n.men <- 500
p.hat.men <- 0.75
se.men <- sqrt (p.hat.men*(1-p.hat.men)/n.men)
n.women <- 500
p.hat.women <- 0.65
se.women <- sqrt (p.hat.women*(1-p.hat.women)/n.women)

# Run 10,000 normal simulations for each group.
n.sims <- 10000
p.men <- rnorm (n.sims, p.hat.men, se.men)
p.women <- rnorm (n.sims, p.hat.women, se.women)

# ratio of the simulation
ratio <- p.men/p.women
# 95% CI of the ratio
int.95 <- quantile (ratio, c(.025,.975)); int.95
```

Outline

log earnings model

- basic idea:
    - run model
    - use resulting coefficients to set up simulations for predictive combinations you might not be able to get from simpler approaches
- e.g. height and gender as predictors of annual earnings

```
library(arm)
earnings<-read.csv()
earn.logmodel.3 <- lm (log.earn ~ height + male + height:male,
                        data=earnings)
display(earn.logmodel.3, digits=3)
```

using predict()
get point estimate

- predict() for log earnings 68" tall man

```
x.new <- data.frame (height=68, male=1)
pred.interval <- predict (earn.logmodel.3, x.new,
                          interval="prediction",level=.95)
pred.interval
exp(pred.interval)
```

simulate predictive uncertainty

simple case

- use point estimate from predict() and residual s.e.

```
pred<-exp(rnorm(1000,10.4, 0.88))
hist(log(pred)) # histogram on log scale
hist(pred) # histogram on original scale
mean(pred)
median(pred) # better measure
quantile(pred,c(0.25,0.75)) # 50% interval
quantile(pred, c(0.025, 0.975)) # 95% interval
```

- not necessary for simple case
    - simple combinations usually reasonably t-distributed (can just use predict())
    - but useful for more complicated combinations and comparisons, non-linear models e.g. logistic

simulate predictive uncertainty

more complicated case

- plug model values into simulation
- e.g. difference earnings between 68"-tall woman and 68"-tall man
    - GH function *sim()* in "arm", returns simulations for all the regression parameters in a model

```
pred.man<-exp(rnorm(1000, 8.4+.017*68-.079*1+.007*68*1, .88))
pred.woman<-exp(rnorm(1000, 8.4+.017*68-.079*0+.007*68*0, .88))
pred.diff <- pred.man - pred.woman
pred.ratio <- pred.man/pred.woman
median(pred.diff); mean(pred.diff)
hist(pred.diff); median(pred.ratio)
quantile(pred.ratio,c(.25,.75))
```

Outline

simulate difficult to calculate predictions

predicting congressional democratic victories

- simulation may be only approach to some predictions
- e.g. predict number congressional districts democrats will win based on previous election results
- predictors are democratic proportion from contested 343 (of 435 congressional districts) and whether candidate is incumbent
  - construct model to predict 1988 election from 1986 election
  - apply model to predict 1990 from 1988
  - can compare prediction to actual 1990 results
  - can model as continuous variable (proportion) or counts (glm)

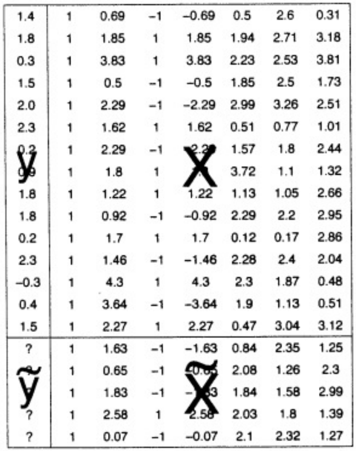
general approach

1. use *lm()* to fit linear model using observed data
2. create matrix of predictor values for unobserved data based on *lm()* results
3. run 1,000 simulations using the matrix
    - *arm::sim()* to simulate set regression coefficients and s.e.'s with uncertainty
    - multiply results of *sim()* by predictor matrix
4. collect results
    - e.g. sum across rows to get predicted proportion of races in which democrats got $> 50\%$ of vote

regression matrix

- recall, regression can be described in terms of a matrix, X, where each column is an indicator variable, and each row is a set of indicator variable values for that observation
- the *observed* outcome y_i is indexed as $X_i\beta = \beta_1 X_{i1} + ... + \beta_k X_{ik}$
- *unobserved* data $(\tilde{X}_i)$ can be used to predict unobserved outcomes $(\tilde{y}_i)$
  - if one group, X_{i1} (first column in the matrix) is equal to 1, a constant term for all individuals in the population
  - multi-level models, the first row of the model is allowed to vary to reflect group membership, so that *each level of the model has its own matrix of predictors*

regression matrix



y	X						
1.4	1	0.69	−1	−0.69	0.5	2.6	0.31
1.8	1	1.85	1	1.85	1.94	2.71	3.18
0.3	1	3.83	1	3.83	2.23	2.53	3.81
1.5	1	0.5	−1	−0.5	1.85	2.5	1.73
2.0	1	2.29	−1	−2.29	2.99	3.26	2.51
2.3	1	1.62	1	1.62	0.51	0.77	1.01
0.2	1	2.29	−1	−2.29	1.57	1.8	2.44
0.9	1	1.8	1	[illegible]	3.72	1.1	1.32
1.8	1	1.22	1	1.22	1.13	1.05	2.66
1.8	1	0.92	−1	−0.92	2.29	2.2	2.95
0.2	1	1.7	1	1.7	0.12	0.17	2.86
2.3	1	1.46	−1	−1.46	2.28	2.4	2.04
−0.3	1	4.3	1	4.3	2.3	1.87	0.48
0.4	1	3.64	−1	−3.64	1.9	1.13	0.51
1.5	1	2.27	1	2.27	0.47	3.04	3.12

$\tilde{y}$	$\tilde{X}$						
?	1	1.63	−1	−1.63	0.84	2.35	1.25
?	1	0.65	−1	−0.65	2.08	1.26	2.3
?	1	1.83	−1	−1.83	1.84	1.58	2.99
?	1	2.58	1	2.58	2.03	1.8	1.39
?	1	0.07	−1	−0.07	2.1	2.32	1.27

Figure: single-level regression matrix

1. fit model with lm()

```
load("~/vote86.RData")

fit.88 <- lm (vote.88 ~ vote.86 + incumbency.88)
    library(arm)
    display(fit.88)
               coef.est coef.se
(Intercept)    0.20     0.02
vote.86        0.58     0.04
incumbency.88  0.08     0.01
n = 343, k = 3
residual sd = 0.067, R-Squared = 0.88
```

2. create matrix

```
n.tilde <- length (vote.88)
X.tilde <- cbind (rep(1,n.tilde), vote.88, incumbency.90)

X.tilde
```

3. run simulations using matrix

```
n.sims <- 1000
sim.88 <- sim (fit.88, n.sims)
sim.88
y.tilde<- array(NA, c(n.sims, n.tilde))
for (s in 1:n.sims){
y.tilde[s,] <- rnorm (n.tilde, X.tilde %*% sim.88@coef[s,],
        sim.88@sigma[s])}
```

4. collect results

```
# sum rows
dems.tilde <- rowSums (y.tilde > .5, na.rm=T)
hist(dems.tilde)
summary(dems.tilde)
```

function based on code

```
Pred.88 <- function (X.pred, lm.fit){
n.pred <- dim(X.pred)[1]
sim.88 <- sim (lm.fit, 1)
y.pred <- rnorm (n.pred, X.pred %*% t(sim.88@coef),
sim.88@sigma)
return(y.pred)
}

my.predict<-Pred.88(X.tilde,fit.88)
hist(my.predict)
```

Exercise 3: Simulating GLM Predictions

Outline

fake-data simulation for model validation

1. simulate outcome data using a "statistical model ... set to fixed "true" values"
2. re-run the model many times using the simulated or fake outcome data as the dependent variable.
3. compare the model parameters from the fake data runs to the model parameters from the real data run

simple example model validation

$y = \alpha + \beta x + \epsilon$, where $\alpha = 1.4$, $\beta = 2.3$, and $\sigma = 0.9$

- set up data

```
library ("arm")
a <- 1.4
b <- 2.3
sigma <- 0.9
x <- 1:5
n <- length(x)
```

- simulate outcome data (normally-distributed error term that is key to the simulation...)

```
y <- a + b*x + rnorm (n, 0, sigma)
```

- fit model using the simulated outcome data

```
lm.1 <- lm (y ~ x)
```

check modeling results reasonably consistent with original parameters

```
display (lm.1)

lm(formula = y ~ x)
            coef.est coef.se
(Intercept) 1.70     0.97
x           2.38     0.29

b.hat <- coef (lm.1)[2]       # "b" is the 2nd coef in the model
b.se <- se.coef (lm.1)[2]     # "b" is the 2nd coef in the model

cover.68 <- abs (b - b.hat) < b.se     # this will be TRUE or FALSE
cover.95 <- abs (b - b.hat) < 2*b.se   # this will be TRUE or FALSE
cat (paste ("68% coverage: ", cover.68, "\n"))
68% coverage:  TRUE
cat (paste ("95% coverage: ", cover.95, "\n"))
95% coverage:  TRUE
```

loop multiple runs...

try t distribution since small n...

```
n.fake <- 1000
cover.68 <- rep (NA, n.fake)
cover.95 <- rep (NA, n.fake)
for (s in 1:n.fake){
  y <- a + b*x + rnorm (n, 0, sigma)
  lm.1 <- lm (y ~ x)
  b.hat <- coef (lm.1)[2]
  b.se <- se.coef (lm.1)[2]
  cover.68[s] <- abs (b - b.hat) < b.se
  cover.95[s] <- abs (b - b.hat) < 2*b.se
}
cat (paste ("68% coverage: ", mean(cover.68), "\n"))
cat (paste ("95% coverage: ", mean(cover.95), "\n"))

# t.68 <-  qt (.84, n-2)
# t.95 <-  qt (.975, n-2)
# t.95
#
```

Poisson example

roach infestation model "real" outcome data

- 160 Tx apartments compared to 104 control apartments
- number of roaches caught over a number of trap days
- control for baseline roach measurement and whether a senior living facility

```
roachdata <- read.csv ("~/roachdata.csv")

glm.1 <- glm (y ~ roach1 + treatment + senior, family=poisson,
data=roachdata, offset=log(exposure2))
display (glm.1)

            coef.est coef.se
(Intercept)  3.09     0.17
roach1       0.01     0.00
treatment   -0.52     0.20
senior      -0.38     0.27
```

create simulated or "fake" outcome data

```
n <- length(roachdata$y) # same length as the real data
X <- cbind (rep(1,n), roachdata$roach1, roachdata$treatment,
roachdata$senior) # matrix holds # predictors from data

 # parameters for sim
y.hat <- roachdata$exposure2 * exp (X %*% coef(glm.1))
# use Poisson for simulated outcome data
y.rep <- rpois (n, y.hat)
```

use number of zero counts in data as comparison statistic

```
print (mean (roachdata$y==0))
0.3587786
print (mean (y.rep==0))
0
```

- 36% zero counts in real data vs. no zero counts simulated data
- likely a problem, but need to look at many simulations to be sure

repeat simulation many times

compare simulated to real data

1. *sim()* to loop over multiple simulations
2. loop test statistic function for mean number of zeros in simulated data
3. check model using test statistic: not nearly same number of zeros (clearly a problem with model fit)

```
n.sims <- 1000
sim.1 <- sim (glm.1, n.sims)
y.rep <- array (NA, c(n.sims, n))
for (s in 1:n.sims){
  y.hat <- roachdata$exposure2 * exp (X %*% sim.1@coef[s,])
  y.rep[s,] <- rpois (n, y.hat)
}

Test <- function (y){
  mean (y==0)
}
test.rep <- rep (NA, n.sims)
for (s in 1:n.sims){
  test.rep[s] <- Test (y.rep[s,])
}

summary(test.rep)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000000 0.000000 0.000000 0.000687 0.000000 0.007634
print (mean (test.rep > Test(y)))
```

repeat with overdispersed model

better fit

```
glm.2 <- glm (y ˜ roach1 + treatment + senior, family=quasipoisson, data=roachdata,
  offset=log(exposure2))
display (glm.2)
            coef.est coef.se
(Intercept)  3.09     0.17
roach1       0.01     0.00
treatment   -0.52     0.20
senior      -0.38     0.27

n.sims <- 1000
sim.2 <- sim (glm.2, n.sims)
y.rep <- array (NA, c(n.sims, n))
overdisp <- summary(glm.2)$dispersion
for (s in 1:n.sims){
  y.hat <- roachdata$exposure2 * exp (X %*% sim.2@coef[s,])
  a <- y.hat/(overdisp-1)              # using R's parametrization for the
  y.rep[s,] <- rnegbin (n, y.hat, a)   # negative binomial distribution
}
test.rep <- rep (NA, n.sims)
for (s in 1:n.sims){
  test.rep[s] <- Test (y.rep[s,])
}
summary(test.rep)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1985  0.2901  0.3168  0.3198  0.3511  0.4466
print (mean (test.rep > Test(y)))
[1] 0.182
```