Chapter 10

Simulation

Simulations provide a powerful way to answer questions and explore properties of statistical estimators and procedures. In this chapter, we will explore how to simulate data in a variety of common settings, and apply some of the techniques introduced earlier.

10.1 Generating data

10.1.1 Generate categorical data

Simulation of data from continuous probability distributions is straightforward using the functions detailed in 3.1.1. Simulating from categorical distributions can be done manually or using some available functions.

```
> options(digits=3)
> options(width=72) # narrow output
> p = c(.1,.2,.3)
> x = runif(10000)
> mycat1 = numeric(10000)
> for (i in 0:length(p)) {
      mycat1 = mycat1 + (x >= sum(p[0:i]))
      }
> table(mycat1)
mycat1
   1    2    3    4
 955 1988 3028 4029
```

```
> mycat2 = cut(runif(10000), c(0, 0.1, 0.3, 0.6, 1))
> summary(mycat2)
  (0,0.1] (0.1,0.3] (0.3,0.6]   (0.6,1]
     1050      2033      3041      3876
> mycat3 = sample(1:4, 10000, rep=TRUE, prob=c(.1,.2,.3,.4))
> table(mycat3)
mycat3
   1    2    3    4
1023 2015 3009 3953
```

The `cut()` function (2.2.4) bins continuous data into categories with both endpoints defined by the arguments. Note that the `min()` and `max()` functions can be particularly useful here in the outer categories. The `sample()` function as shown treats the values $1, 2, 3, 4$ as a dataset, and samples from the dataset 10,000 times with the probability of selection defined in the `prob` vector.

10.1.2 Generate data from a logistic regression

Here we show how to simulate data from a logistic regression (7.1.1). Our process is to generate the linear predictor, then apply the inverse link, and finally draw from a distribution with this parameter. This approach is useful in that it can easily be applied to other generalized linear models (7.1). Here we make the intercept -1, the slope 0.5, and generate $5,000$ observations.

```
> intercept = -1
> beta = 0.5
> n = 5000
> xtest = rnorm(n, mean=1, sd=1)
> linpred = intercept + (xtest * beta)
> prob = exp(linpred)/(1 + exp(linpred))
> ytest = ifelse(runif(n) < prob, 1, 0)
```

While the results of `summary()` for a `glm` object is relatively concise, we can display just the estimated values of the coefficients from the logistic regression model using the `coef()` function (see 6.4.1).

```
> coef(glm(ytest ~ xtest, family=binomial))
(Intercept)       xtest
     -1.005       0.483
```

10.1.3 Generate data from a generalized linear mixed model

In this example, we generate data from a generalized linear mixed model (7.4.6) with a dichotomous outcome. We generate 1500 clusters, denoted by `id`. There is one predictor with a common value for all observations in a cluster (X_1). Each observation within the cluster has an order indicator (denoted by X_2) that has a linear effect (`beta_2`), and there is an additional predictor that varies among observations (X_3). The dichotomous outcome Y is generated from these predictors using a logistic link incorporating a normal distributed random intercept for each cluster.

The simulation approach is an extension of that shown in the previous section (see also 4.1.3).

```
> n = 1500; p = 3; sigbsq = 4
> beta = c(-2, 1.5, 0.5, -1)
> id = rep(1:n, each=p)         # 1 1 ... 1 2 2 ... 2 ... n
> x1 = as.numeric(id < (n+1)/2) # 1 1 ... 1 0 0 ... 0
> randint = rep(rnorm(n, 0, sqrt(sigbsq)), each=p)
> x2 = rep(1:p, n)              # 1 2 ... p 1 2 ... p ...
> x3 = runif(p*n)
> linpred = beta[1] + beta[2]*x1 + beta[3]*x2 + beta[4]*x3 + randint
```

```
> expit = exp(linpred)/(1 + exp(linpred))
> y = runif(p*n) < expit        # generate a logical as our outcome
```

We fit the model using the `glmer()` function from the `lme4` package.

```
> library(lme4)

Loading required package:  Matrix
Loading required package:  Rcpp

> glmmres = glmer(y ~ x1 + x2 + x3 + (1|id), family=binomial(link="logit"))
> summary(glmmres)
Generalized linear mixed model fit by maximum likelihood (Laplace
  Approximation) [glmerMod]
 Family: binomial  ( logit )
Formula: y ~ x1 + x2 + x3 + (1 | id)

     AIC      BIC   logLik deviance df.resid
    5251     5283    -2621     5241     4495

Scaled residuals:
   Min     1Q Median     3Q    Max
-2.019 -0.494 -0.286  0.569  2.846

Random effects:
 Groups Name        Variance Std.Dev.
 id     (Intercept) 3.09     1.76
Number of obs: 4500, groups:  id, 1500

Fixed effects:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.9668     0.1633  -12.04  < 2e-16 ***
x1            1.5557     0.1319   11.80  < 2e-16 ***
x2            0.4631     0.0501    9.25  < 2e-16 ***
x3           -1.0337     0.1550   -6.67  2.5e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
   (Intr) x1     x2
x1 -0.498
x2 -0.673  0.103
x3 -0.387 -0.073 -0.050
```

10.1.4 Generate correlated binary data

Another way to generate correlated dichotomous outcomes Y_1 and Y_2 is based on the probabilities corresponding to the 2×2 table. Given these cell probabilities, the variable probabilities can be expressed as a function of the marginal probabilities and the desired correlation, using the methods of Lipsitz and colleagues [103]. Here we generate a sample of 1000 values where: $P(Y_1 = 1) = .15, P(Y_2 = 1) = .25$, and $\text{Corr}(Y_1, Y_2) = 0.40$.

```
> p1 = .15; p2 = .25; corr = 0.4; n = 10000
> p1p2 = corr*sqrt(p1*(1-p1)*p2*(1-p2)) + p1*p2
> library(Hmisc)
> vals = rMultinom(matrix(c(1-p1-p2+p1p2, p1-p1p2, p2-p1p2, p1p2),
     nrow=1, ncol=4), n)
> y1 = rep(0, n); y2 = rep(0, n)    # put zeroes everywhere
> y1[vals==2 | vals==4] = 1         # and replace them with ones
> y2[vals==3 | vals==4] = 1         # where needed
> rm(vals, p1, p2, p1p2, corr, n)   # cleanup
```

The generated data is close to the desired values.

```
> cor(y1, y2)
[1] 0.429
> table(y1)
y1
   0    1
8515 1485
> table(y2)
y2
   0    1
7542 2458
```

10.1.5 Generate data from a Cox model

To simulate data from a Cox proportional hazards model (7.5.1), we need to model the hazard functions for both time to event and time to censoring. In this example, we use a constant baseline hazard, but this can be modified by specifying other `scale` parameters for the Weibull random variables.

```
> # generate data from Cox model
> n = 10000
> beta1 = 2; beta2 = -1
> lambdaT = .002 # baseline hazard
> lambdaC = .004  # hazard of censoring
> x1 = rnorm(n)   # standard normal
> x2 = rnorm(n)
> # true event time
> T = rweibull(n, shape=1, scale=lambdaT*exp(-beta1*x1-beta2*x2))
> C = rweibull(n, shape=1, scale=lambdaC)   #censoring time
> time = pmin(T,C)  #observed time is min of censored and true
> censored = (time==C)   # set to 1 if event is censored
> # fit Cox model
> library(survival)
> survobj = coxph(Surv(time, (1-censored))~ x1 + x2, method="breslow")
```

These parameters generate data where approximately 40% of the observations are censored. The `coxph()` function expects an observed event indicator. We tabulate the censoring indicator, then display the results as well as the associated 95% confidence intervals.