MyUNT EagleConnect Blackboard People & Departments Maps Calendars Giving to UNT

Benchmarks

A green light to greatness.

ABOUT BENCHMARK ONLINE SEARCH ARCHIVE SUBSCRIBE TO BENCHMARKS ONLINE

Home » issues » 2014-10 » rss-matters

RSS Matters

R_stats

Research and Statistical Support
University of North Texas

*BOO*tstrapping the Generalized Linear Model

Link to the last RSS article here:Factor Analysis with Binary items: A quick review with examples. -- Ed.

By Dr. Richard Herrington, Research and Statistical Support Consultant Team

Researchers do not need to be afraid - the availability of fast computers and public domain software libraries such as R and the R package *boot*, make forays into *bootstrap confidence interval estimation* reasonably straight forward. R package *boot* was designed to be general enough to allow the data analyst to simulate the empirical sampling distribution of most estimators (and then some), and to calculate corresponding confidence intervals for that estimator. There are a few tricks to learn when using package boot, but once those small hurdles have been navigated, the lessons learned can be applied more generally to other estimation settings.

R package boot is comprised of a set of functions that are well documented both with theory and examples in the book: *Bootstrap Methods and Their Application*, by A.C. Davison and D.V. Hinkley (1997). The purpose of this short note is to demonstrated how to approximate nonparametric confidence intervals, using resampling methods, for the *generalized linear model* (glm) using the R package *boot*.

We'll start off by simulating a data set from the following probability regression model:

```
samp.size<-5000


x1 <- rnorm(samp.size)

x2 <- rnorm(samp.size)

x3 <- rnorm(samp.size)
```

```
x4 <- rnorm(samp.size)


#   True Model

#   x0  x1   x2    x3   x4   x1*x2

z <- 1 + 2*x1 + 3*x2  + 4*x3 + 5*x4 + 10*x1*x2

pr <- 1/(1+exp(-z))

y <- rbinom(samp.size,1,pr)


> sim.data.df <- data.frame(y=y,x1=x1,x2=x2,x3=x3,x4=x4,
                ,x5=x1*x2)

> head(sim.data.df)

  y         x1         x2         x3         x4         x5

1 0  0.9632201 -1.0871521 -2.0283342  0.5727080 -1.0471668

2 0  2.8738768 -1.4818353  0.1265646  1.9195807 -4.2586121

3 1 -0.5552309  0.8576629  1.1878977 -0.7940654 -0.4762010

4 0 -0.7519217  0.7630796 -0.7534080 -0.6768429 -0.5737761

5 0  0.6789053 -1.6454898  0.5337027 -0.9163869 -1.1171318

6 0  1.4138792 -0.3052833  1.0388294 -0.9189572 -0.4316337

.

.

.
```

Using the R function *glm* we can estimate the model coefficients using a binomial probability model for the *y* outcome variable:

```
glm.fit<-glm(y~x1+x2+x3+x4+x1*x2,

         data=sim.data.df,

         family="binomial")

glm.fit


> glm.fit
```

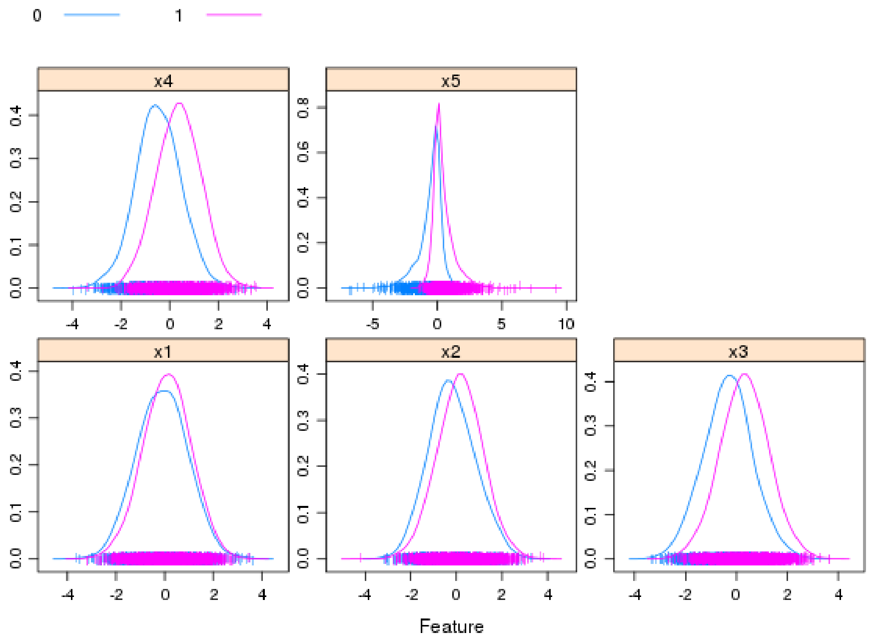
```
Call:  glm(formula = y ~ x1 + x2 + x3 + x4 + x1 * x2, family = "binomial",

    data = sim.data.df)



Coefficients:

(Intercept)           x1           x2           x3           x4        x1:x2

      1.009        1.973        3.101        4.081        5.113       10.144



Degrees of Freedom: 4999 Total (i.e. Null);  4994 Residual

     Null Deviance:           6910

 Residual Deviance: 1265  AIC: 1277
```

R function *glm* does a reasonably good job of recovering the population regression coefficients – although we did use a very large sample size in comparison to the number of variables in the model.

R package *caret* provides a useful helper function for displaying kernel density estimated histograms for the predictors as a function of the two level outcome variable *y*:

```
library(caret)

featurePlot(x = sim.data.df[,c(2:6)],

       y = as.factor(sim.data.df$y),

       plot = "density",

       scales = list(x = list(relation="free"),

                y = list(relation="free")),

       adjust = 1.5,

       pch = "|",

       layout = c(3, 3),

       auto.key = list(columns = 2))
```

The resulting plot is returned:



The chosen population coefficients separate the groups with a large difference between the groups (1/0) on the predictor variables. We can calculate the marginal probabilities of the estimated predictors to see how large the average probability change is, in moving from a 50% probability of being in group 1, to the estimated probability of being in group 1, given a unit change in the predictors:

```
library(arm)

glm.coefs<-coef(glm.fit)

invlogit(glm.coefs) - .50

(Intercept)          x1          x2          x3          x4       x1:x2

  0.2327767   0.3779851   0.4569387   0.4833883   0.4940197   0.4999607
```

We have chosen very large predictor *effect sizes* for the simulation. Essentially, predictors *x4* and *x5* maximally predict the probability of *y=1* membership: knowledge of predictors x4 and x5 move our predicted marginal probability of y=1 from .50 (absent the information from *x4* and *x5*) to .99 given the information provided by *x4* and *x5*.

Now on to the bootstrap confidence intervals: first we need to create a wrapper function that will pass the resampled data, and their corresponding indices, to the *glm* function:

```
glm.coefs<-function (dataset, index)
{
 sim.data.df<-dataset[index,]

 glm.fit <-try(glm(y~x1+x2+x3+x4, #+x1*x2,
                  data=sim.data.df,
                  family="binomial"), silent = TRUE)

 coefs<-try(coef(glm.fit), silent=TRUE)
 print(coefs)

 return(coefs)
}
```

The vector that contains the indices of the resampled data (*index*) will be passed to the *glm* function. Lastly, our wrapper function for glm - *glm.coefs* – will return the estimated coefficients back to the *boot* function for tabulation and post-processing. Additionally, we have used the *try* function so that if a resampled data set fails *glm* estimation, the *glm.coefs* and *boot* will not break out with error, but will instead continue with missing values for the coefficients. Lastly, we have put a print statement within the body of glm.coefs, so that we can monitor the estimated coefficients values as they are being estimated.

Our last bit of R script sends the data and glm.coefs function to boot for processing:

```
boot.fit<-boot(sim.data.df, glm.coefs, R=1000)
boot.fit

for(ii in 1:length(boot.fit$t0))
 {
 cat(rep("\n",5))
 print(names(boot.fit$t0[ii]))
 cat(rep("\n",2))
 print(boot.ci(boot.fit, conf = 0.95, type = c("norm","perc","basic"),index = ii))
 }
```

The for loop in this script isn't necessary, but is merely a short-cut for printing out the results of three different types of confidence intervals (CI) for for the six estimated parameters (intercept and x1-x6). Notice that we capture the true population parameter for each of the three CI types. This a simply a consequence of having used few predictors, an initial large sample size, and 1000 bootstrap samples in the bootstrap CI estimation.

```
> boot.fit

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = sim.data.df, statistic = glm.coefs, R = 1000)


Bootstrap Statistics :
     original      bias    std. error
t1*  1.008756 0.007386088  0.08582566
t2*  1.973487 0.011373649  0.12787464
t3*  3.101113 0.027926437  0.15442723
t4*  4.080900 0.027597606  0.17447659
t5*  5.113291 0.036752067  0.21991954
t6* 10.144203 0.074247504  0.42935352
> for(ii in 1:length(boot.fit$t0))
+  {
+  cat(rep("\n",5))
+  print(names(boot.fit$t0[ii]))
+  cat(rep("\n",2))
+  print(boot.ci(boot.fit, conf = 0.95, type = c("norm","perc","basic"), index = ii))
```

```
+   }
```

```
 [1] "(Intercept)"



BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.fit, conf = 0.95, type = c("norm", "perc",
    "basic"), index = ii)

Intervals :
Level      Normal              Basic              Percentile
95%   ( 0.833,  1.170 )   ( 0.824,  1.164 )   ( 0.854,  1.194 )
Calculations and Intervals on Original Scale

[1] "x1"



BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.fit, conf = 0.95, type = c("norm", "perc",
    "basic"), index = ii)

Intervals :
Level      Normal              Basic              Percentile
95%   ( 1.711,  2.213 )   ( 1.704,  2.191 )   ( 1.756,  2.243 )
```

```
Calculations and Intervals on Original Scale

 [1] "x2"



BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.fit, conf = 0.95, type = c("norm", "perc",
   "basic"), index = ii)

Intervals :
Level      Normal              Basic              Percentile
95%   ( 2.771,  3.376 )   ( 2.731,  3.369 )   ( 2.833,  3.471 )
Calculations and Intervals on Original Scale

 [1] "x3"



BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.fit, conf = 0.95, type = c("norm", "perc",
   "basic"), index = ii)

Intervals :
Level      Normal              Basic              Percentile
95%   ( 3.711,  4.395 )   ( 3.704,  4.369 )   ( 3.793,  4.457 )
Calculations and Intervals on Original Scale
```

```
[1] "x4"

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.fit, conf = 0.95, type = c("norm", "perc",
    "basic"), index = ii)

Intervals :
Level      Normal              Basic              Percentile
95%   ( 4.646,  5.508 )   ( 4.621,  5.498 )   ( 4.728,  5.606 )
Calculations and Intervals on Original Scale

[1] "x1:x2"

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot.fit, conf = 0.95, type = c("norm", "perc",
    "basic"), index = ii)
```

```
Intervals :

Level      Normal              Basic              Percentile

95%   ( 9.23, 10.91 )   ( 9.15, 10.84 )   ( 9.45, 11.13 )

Calculations and Intervals on Original Scale
```


Contact Us:

University Information Technology
1155 Union Circle #310709
Denton, TX 76203 USA
Voice: 940-565-4068
Fax: 940-565-4060

Visit Us:

Sage Hall, Room 338
http://it.unt.edu/benchmarks/

Email us:

Have questions on content or technical issues? Please contact us.
unt.uit@unt.edu