Simulation studies

Patrick Breheny

September 8

Introduction

- In statistics, we are often interested in properties of various estimation and model selection procedures: bias, variance, mean squared error, how often a certain variable is selected, etc.
- Suppose we have data $x_1, \ldots, x_p, y$ with distribution $f(\mathbf{x}, y)$; denote our estimate $g(\mathbf{x}, y)$ and our quantity of interest (*e.g.*, the variance of the estimator) as $h(g)$
- In principle, we could calculate the expected value of our quantity of interest by evaluating the integral

$$\int \int \cdots \int h(g(\mathbf{x}, y)) f(\mathbf{x}, y) dx_1 \cdots dx_p dy$$

Monte Carlo simulation

- Clearly, this integral is well beyond our means to evaluate except in special cases
- We can, however, approximate the integral via simulation
- To do this, we generate $\mathbf{x}$ and y from $f(\mathbf{x}, y)$, then calculate $g(\mathbf{x}, y)$ and $h(g)$
- We do this repeatedly, and obtain the random variables $\{h_1, h_2, \ldots, h_N\}$

Monte Carlo simulation

- Now, by the law of large numbers,

$$\lim_{N \to \infty} \frac{h_1 + \cdots + h_N}{N} = \mathrm{E}[h(g(\mathbf{x}, y))],$$

  our quantity of interest
- Although we cannot actually carry out this simulation an infinite number of times, we can obtain accurate approximations of our quantity of interest by running a large number of simulations (N)
- This technique of approximation is known as *Monte Carlo simulation* or *Monte Carlo integration*

General framework

The general framework for writing code to accomplish this is

```
for (i in 1:N)
  {
    Data = GenerateData()
    Estimate[i] = g(Data$X,Data$y)
  }
h(Estimate)
```

In general, and especially if the simulation is time-consuming, it is best to record and save all the estimates so that later on, you can calculate all the quantities of interest that occur to you

Extending the general framework

The general framework on the previous slide can be extended in many ways; suppose we wish to generate data according to several distributions and compare several different estimators:

```
for (i in 1:N)
  {
    for (j in 1:J)
      {
        Data = GenerateData(j)
        for (k in 1:K)
        {
          Estimate[i,j,k] = g(Data$X,Data$y,k)
        }
      }
  }
apply(Estimate,2:3,h)
```

Principles of experimental design

- Note that a simulation is an experiment, and the general principles of experimental design are often relevant
- For example, in the previous slide, I blocked on `Data` in order to eliminate variability due to data generation from my comparison of the estimators

Simulations in SAS?

- I will be focusing here on carrying out simulations in R
- SAS can also be used to carry out simulations, although the language is not designed for this purpose, and therefore requires the use of tools such as macros to set up variables and arrays that can be used across DATA and PROC statements
- This is not a severe impediment for simple simulations involving a small number of variables, but it becomes a burden when dealing with multivariable models
- Nevertheless, if you wish to see how one might set up a multivariable simulation of ridge regression vs. OLS in SAS, check the SAS code for this lecture

Overview

- To illustrate the principles of simulation design, we will implement a simulation study comparing ridge regression, lasso, and subset selection
- To better illustrate the structure of the simulation, and so that the code fits on a single slide, we will break down this job into several tasks:
  - Generating the data
  - Carrying out ridge regression
  - Carrying out the lasso
  - Carrying out subset selection
  - Comparing the three methods

Generating the data

```
genData <- function(n,p,beta)
  {
    X <- matrix(runif(n*p),nrow=n,ncol=p)
    y <- rnorm(n,X%*%beta)
    return(list(X=X,y=y))
  }
```

Possible changes/extensions: allowing the user to change σ^2, different distribution for y, different distribution for X, correlation among the columns of $\mathbf{X}$

Ridge regression

```
ridge <- function(Data,n.lam=301)
  {
    require(MASS)
    lam <- c(0,exp(seq(log(1e-2),log(1e7),len=n.lam)))
    fit <- lm.ridge(y~X,Data,lambda=lam)
    return(coef(fit)[which.min(fit$GCV),])
  }
```

Feel free to check that this range of λ values is reasonable by inserting a `plot` statement

Best subsets

```
bestsub <- function(Data)
  {
    require(leaps)
    fit <- regsubsets(y~X,Data)
    b <- numeric(ncol(Data$X)+1)
    names(b) <- fit$xnames
    bb <- coef(fit,which.min(summary(fit)[["bic"]]))
    b[names(bb)] <- bb
    return(b)
  }
```

Possible changes/extensions: allowing the user to specify selection via C_p instead of BIC

Lasso

```
lasso <- function(Data)
  {
    require(glmnet)
    cvfit <- cv.glmnet(Data$X,Data$y)
    return(as.numeric(coef(cvfit,s=cvfit$lambda.min)))
  }
```

Note that `as.numeric` is required here because `glmnet` stores its values in a special "sparse matrix" format

Putting it all together

```
N <- 100
beta <- c(2,-2,rep(0,8))
p <- length(beta)
results <- array(NA,dim=c(N,p,4),
  dimnames=list(1:N,1:p,c("Subset","Lasso","Ridge","OLS")))
for (i in 1:N)
  {
    Data <- genData(100,p,beta)
    results[i,,1] <- bestsub(Data)[-1]
    results[i,,2] <- lasso(Data)[-1]
    results[i,,3] <- ridge(Data)[-1]
    results[i,,4] <- coef(lm(y~X,Data))[-1]
    displayProgressBar(i,N)
  }
```

Looking at quantities of interest

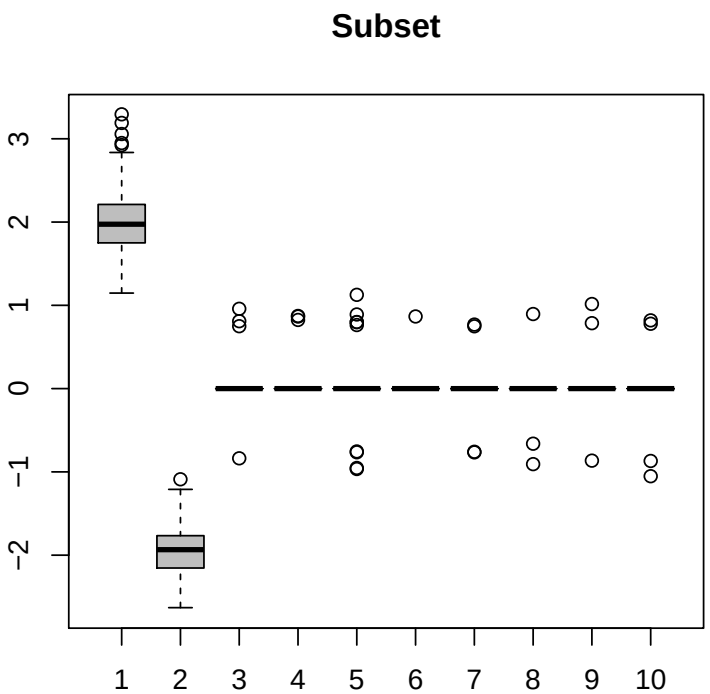
```
B <- apply(results,2:3,mean)-beta
V <- apply(results,2:3,var)
MSE <- B^2+V
apply(MSE,2,sum)
```

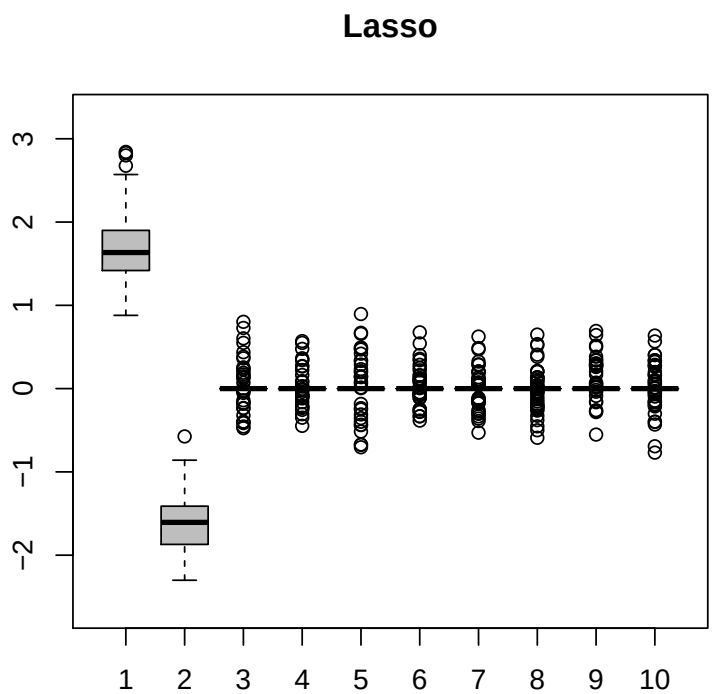
Obviously, there are many other ways to carry out this simulation, each with its own strengths and weaknesses, so feel free to modify everything

Looking at quantities of interest (cont'd)

One advantage to saving all the estimates, however, is that it allows you to look at other quantities of interest after the simulation is over, such as:

```
ylim <- range(results)
boxplot(results[,,1],col="gray",ylim=ylim,main="Subset")
boxplot(results[,,2],col="gray",ylim=ylim,main="Lasso")
```





How many simulations?

- As we said earlier, if N is large, our answer should be very close to the actual mean squared error (or whatever our quantity of interest may be)
- In the preceding code, I used $N = 100$; is that large enough?
- To answer this question, we can use standard confidence interval techniques to assess the variability of the mean
- So, for example, the standard error of $\hat{\beta}_1^{\text{Subset}}$ was 0.41; thus, with $N = 100$, our results are accurate to within about ± 0.08; with a sample size of $N = 1000$, we could cut that to ± 0.03
- Of course, it would also take 10 times as long to run; in reality, there is a tradeoff between accuracy and computational burden that depends on the problem