Computer class 4 solutions

Richard Wilkinson

Solution 1

In this question we will compare the performance of 3 models on the hills dataset.

```
library(MASS)
M1 <- lm(time ~ dist + climb, data=hills)
M2 <- lm(time ~ dist+climb + I(climb^2), data=hills)
M3 <- lm(time ~ dist*climb+I(dist^2)+I(climb^2), data=hills)
```

First we need to create the folds. Choose K=5 to begin with.

```
library(cvTools)
```

```
## Loading required package: lattice
```

```
## Loading required package: robustbase
```

```
folds <- cvFolds(n=dim(hills)[1], K = 5, R = 50)
```

To fit the model using cvTools we need to create a call function which runs the command we wish to repeat.

```
call_M1 <- call <- call('lm', formula=time ~ dist + climb)
call_M2 <- call <- call('lm', formula=time ~ dist + climb+ I(climb^2))
call_M3 <- call <- call('lm', formula=time ~ dist*climb+I(dist^2)+I(climb^2))
```

```
CV5fold_M1 <- cvTool(call_M1, data=hills,y=hills$time, folds=folds)
CV5fold_M2 <- cvTool(call_M2, data=hills,y=hills$time, folds=folds)
CV5fold_M3 <- cvTool(call_M3, data=hills,y=hills$time, folds=folds)
```

```
mean(CV5fold_M1)
```

```
## [1] 16.97273
```

```
mean(CV5fold_M2)
```

```
## [1] 13.5341
```

```
mean(CV5fold_M3)
```

```
## [1] 25.19269
```

So model 2 has the best predictive accuracy, and model 3 has the worst predictive accuracy.

Note that the predictive accuracy is also more variable for model 3.

```
sd(CV5fold_M1)
```

```
## [1] 0.9210194
```

```
sd(CV5fold_M2)
```

```
## [1] 1.095556
```

```
sd(CV5fold_M3)
```

```
## [1] 15.48685
```

Let's now do 10 fold CV.

```
folds <- cvFolds(n=dim(hills)[1], K = 10, R = 50)
CV5fold_M1 <- cvTool(call_M1, data=hills,y=hills$time, folds=folds)
CV5fold_M2 <- cvTool(call_M2, data=hills,y=hills$time, folds=folds)
CV5fold_M3 <- cvTool(call_M3, data=hills,y=hills$time, folds=folds)
mean(CV5fold_M1)
```

```
## [1] 16.8509
```

```
mean(CV5fold_M2)
```

```
## [1] 12.99181
```

```
mean(CV5fold_M3)
```

```
## [1] 16.98671
```

```
sd(CV5fold_M1)
```

```
## [1] 0.6422647
```

```
sd(CV5fold_M2)
```

```
## [1] 0.4279157
```

```
sd(CV5fold_M3)
```

```
## [1] 8.816136
```

And LOO-CV

```
folds <- cvFolds(n=dim(hills)[1], K = dim(hills)[1], R = 50)
CV5fold_M1 <- cvTool(call_M1, data=hills,y=hills$time, folds=folds)
CV5fold_M2 <- cvTool(call_M2, data=hills,y=hills$time, folds=folds)
CV5fold_M3 <- cvTool(call_M3, data=hills,y=hills$time, folds=folds)
mean(CV5fold_M1)
```

```
## [1] 16.74691
```

```
mean(CV5fold_M2)
```

```
## [1] 12.77997
```

```
mean(CV5fold_M3)
```

```
## [1] 13.89262
```

Notice that 5 and 10 fold CV agree that model 2 is the best and model 3 the worst (in terms of prediction). But LOO-CV gives model 3 a better prediction error than model 1, and nearly as good as model 2. This is a situation in which I would trust 5-fold CV rather than LOO-CV. It is possible that LOO-CV hasn't shaken up the data sufficiently, and so it may not be giving a true reflection of a model's predictive skill.

Solution 2

$$F(x) = \int_{-\infty}^{x} \frac{1}{\pi(1+x'^2)} dx' = \int_{-\frac{\pi}{2}}^{\tan^{-1}(x)} \frac{\sec^2(u)}{\pi(1+\tan^2(u)} du \tag{1}$$

$$= \int_{-\frac{\pi}{2}}^{\tan^{-1}(x)} \frac{1}{\pi} du \tag{2}$$

$$= \frac{1}{\pi}\tan^{-1}(x) + \frac{1}{2} \tag{3}$$

Thus, by rearranging $U = F(X)$ we get

$$X = \tan\left(\pi(U - \frac{1}{2})\right)$$

To implement this, we can do the following:

```
U <- runif(10^6)
X <- tan(pi*(U-0.5))
```

To check this as requested

```
xx <- c(-10,-5,0,5,10)
sapply(xx,  function(x) sum(X<=x)/length(X) )
```

```
## [1] 0.032067 0.062936 0.500349 0.937822 0.968606
```

```
pcauchy(xx)
```

```
## [1] 0.03172552 0.06283296 0.50000000 0.93716704 0.96827448
```

Solution 3

$$g(x) = \frac{1}{2}g_1(x) + \frac{1}{2}g_2(x)$$

where

$$g_1(x) = \begin{cases} e^{-x} & \text{if } x \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

and

$$g_2(x) = \begin{cases} e^{x} & \text{if } x \leq 0 \\ 0 & \text{otherwise.} \end{cases}$$

So we can sample from $g(x)$ by sampling $Y \sim \text{Exp}(1)$ and then setting

$$X = \begin{cases} Y & \text{with probability } \frac{1}{2} \\ -Y & \text{otherwise.} \end{cases}$$

Let's start by creating a function to sample from g.

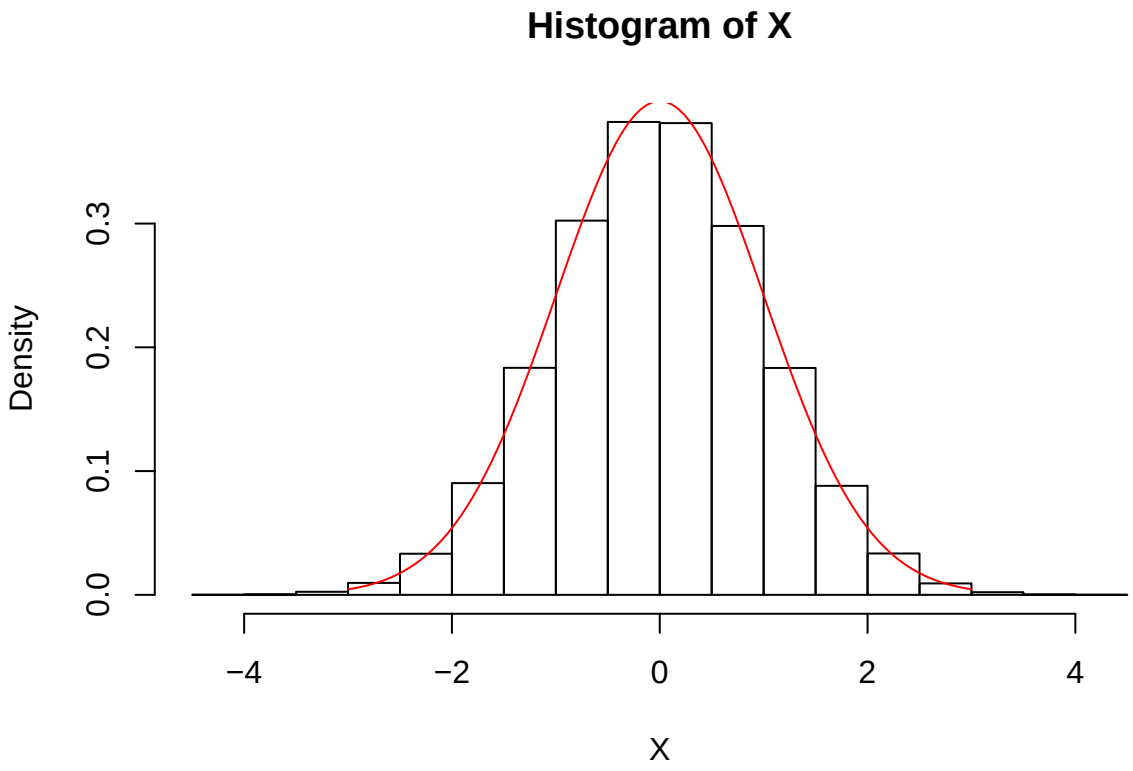
```
rg <- function(n){
  Y <- rexp(n,1)
  U <- sample(c(-1,1), n, replace=TRUE)
  return(Y*U)
}
```

We can write another function to calculate the acceptance probability:

```
acceptanceProb <- function(x){
  exp(abs(x) - x^2/2 -1/2)
}
```

We can then write a loop to do rejection sampling.

```
nacc<-0
X <- c()
while(nacc < 10^5){
  Y <- rg(1)
  if(runif(1)< acceptanceProb(Y)){
    nacc <- nacc+1
    X[nacc] <-Y
  }
}
hist(X, probability=TRUE)
curve(dnorm, -3,3, col=2, add=TRUE)
```



This shows how slow R is at doing loops. In this case it is quicker to do the vectorized calculation, even if this means we simulate more than the 10^5 random variables requested

```
Y<- rg(10^6)
p <- acceptanceProb(Y)
accept <- runif(10^6)<=p
X2 <- Y[accept]
length(X2)
```

```
## [1] 760448
```

The accepance probability is

$$\frac{1}{M} = \sqrt{\frac{\pi}{2e}}$$

which we can check with

```
sqrt(pi/(2*exp(1)))
```

```
## [1] 0.7601735
```

```
length(X2)/10^6
```

```
## [1] 0.760448
```