Resampling - Cross Validation and the Bootstrap

José Fortuny

Friday, April 10, 2015

This document describes the process used in duplicating the labs for chapter 5 of the ISLR book.

Preparing the Environment

We start with the code that sets us the environment and creates a training (and by default a test) dataset:

```
## Cross Validation

# Prepare the environment
library (ISLR)
set.seed(1)
train <- sample(392, 196)
```

Preparing the data and obtaining the estimates from the sample

Wr will use the Auto dataset to illustrate the concepts of Cross Validation. But first we prepare the data to eliminate NA values and use the "only" existing datase we have to calculate the residuals after fitting both a linear and a quadratic model.

```
# Use the Auto dataset to fit a linear model
# Fix the NA values in horsepower first
Auto[is.na(Auto$horsepower),c("horsepower")] <- mean(Auto$horsepower, na.rm=T)
lm.fit <- lm(mpg~horsepower, data=Auto, subset=train)
mean((Auto$mpg - predict(lm.fit, Auto))[-train]^2)
```

```
## [1] 26.14142
```

```
# And to fit a polynomial model
lm.fit2 <- lm(mpg[train]~poly(horsepower[train],2),data=Auto)
mean((Auto$mpg[train] - predict(lm.fit2,Auto))[-train]^2)
```

```
## Warning: 'newdata' had 392 rows but variables found have 196 rows
```

```
## [1] 15.18688
```

Cross Validation

For the sections that follow we will use the **boot** library to repeat actions automatically. We will also use the **glm** (Generalized Linear Model) function to fit the models, even for linear regression, as it provides the function **cv.glm** to run the Cross Validations (CV).

Leave One Cross Validation

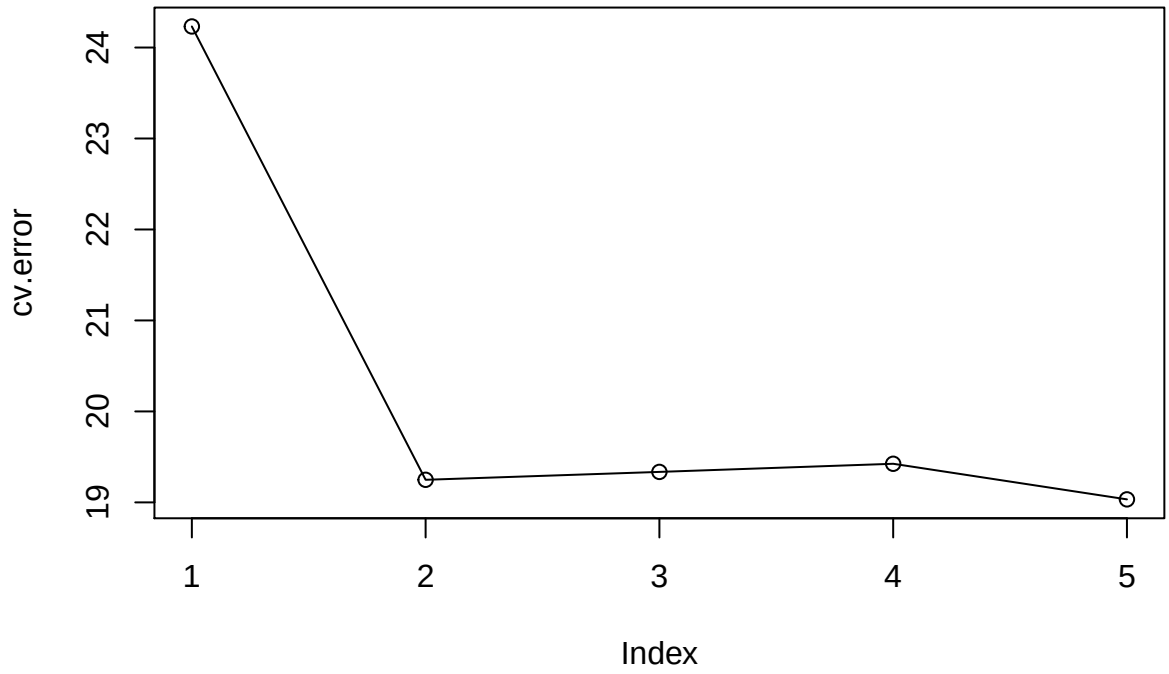
Since we're using a continuous variable as the label and a continuous variable as predictor we will use the error in the test dataset to calculate the standard error of the residuals for different order polynomial functions, as follows:

```
# Leave one out Cross Validation (LOOCV)
library(boot)
# glm.fit <- glm(mpg~horsepower, data=Auto)
# cv.err <- cv.glm(Auto, glm.fit)
# cv.err$delta

# fit through the fifth degree polynomial
cv.error <- rep(0,5)
for (i in 1:5) {
  glm.fit <- glm(mpg~poly(horsepower, i), data=Auto)
  cv.error[i] = cv.glm(Auto, glm.fit)$delta[1]
}
cv.error
```

```
## [1] 24.23151 19.24821 19.33498 19.42443 19.03321
```

```
plot(cv.error)
lines(cv.error)
```



Note that the for loop cycles through 5 cases and, in each case, we run a fit of a polynomial of degree equal

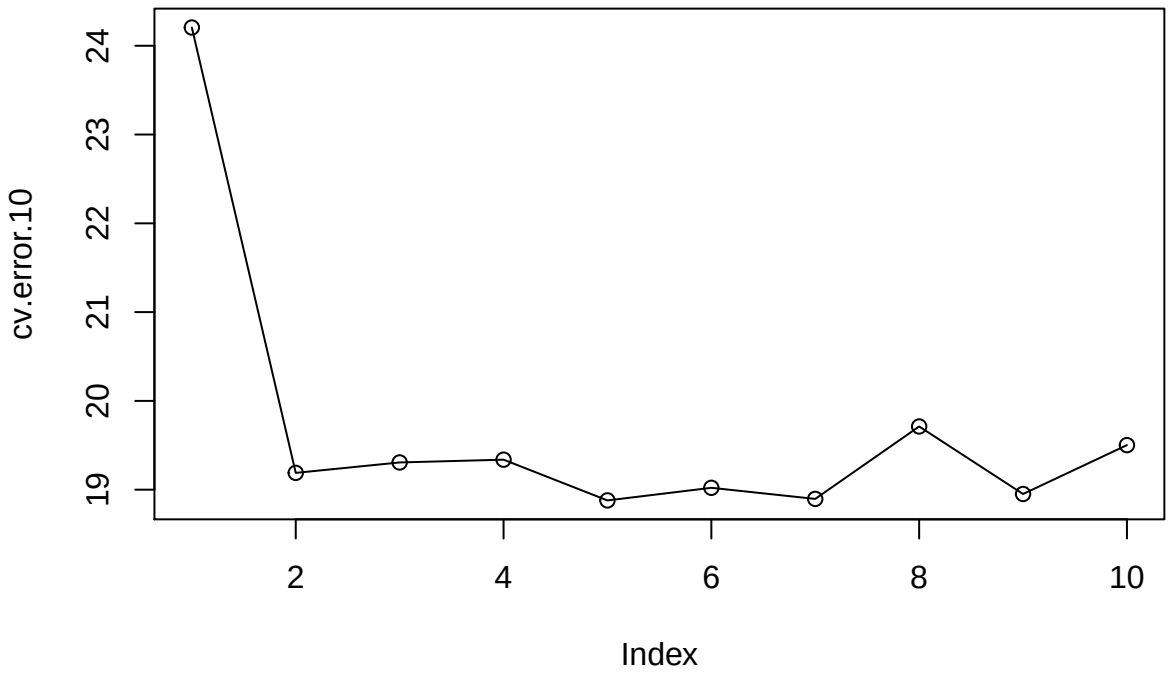
to the iterator. For that polynomial we run a CV process which, in this case, since there is no parameter K passed to the function cv.glm, results in a default K = 1 or LOOCV

K-Fold Cross Validation

```
# k-fold CV
set.seed(17)
cv.error.10 = rep(0,10)
for (i in 1:10) {
  glm.fit <- glm(mpg~poly(horsepower, i), data=Auto)
  cv.error.10[i] = cv.glm(Auto, glm.fit, K=10)$delta[1]
}
cv.error.10
```

```
##  [1] 24.20520 19.18924 19.30662 19.33799 18.87911 19.02103 18.89609
##  [8] 19.71201 18.95140 19.50196
```

```
plot(cv.error.10)
lines(cv.error.10)
```



The Bootstrap

In the Bootstrap we create samples, of the same size as the data available, with replacement and use these samples as training data. Then we can use the resulting residuals to more accurately extimate the variability of the model.

The bootstrap can be applied to models of any kind, as we can see in the example that starts us in which we're trying to estimate the value of α, the proportion of an investment to place in one of two possible investment options.

```
# This is for the example of the investment selection in section 5.2
# First we create the function that estimates the value of alpha
alpha.fn <- function (data, index) {
  X <- data$X[index]
  Y <- data$Y[index]
  return((var(Y)-cov(X,Y))/(var(X)+var(Y)-2*cov(X,Y)))
}
set.seed(1)
alpha.fn(Portfolio, sample(100, 100, replace=T))
```

```
## [1] 0.5963833
```

```
library(boot)      # The boot function in the same name library generates the replicates
boot(Portfolio, alpha.fn, R=1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Portfolio, statistic = alpha.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1* 0.5758321 -7.315422e-05  0.08861826
```

The **boot** function needs, at least, the *data* to which it should apply the *function* and the number of times to run the *replication*.

Here is another example of running the bootstrap to estimate the stability of the linear regression estimated coefficients.

```
# Now to estimate the accuracy of a linear regression model
boot.fn <- function (data, index) {
  return(coef(lm(mpg~horsepower, data=data, subset=index)))
}
set.seed(1)
boot.fn(Auto, sample(392,392,replace=TRUE))
```

```
## (Intercept)  horsepower
##  38.7387134  -0.1481952
```

```r
# now run the bootstrap
boot(Auto, boot.fn, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 39.9358610  0.0296667441 0.860440524
## t2* -0.1578447 -0.0003113047 0.007411218
```

```r
# compare against the theoretical values
summary(lm(mpg~horsepower,data=Auto))$coef
```

```
##               Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
## horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

And yet another example of running the bootstrap to estimate the variability of the coefficients of a quadratic fit

```r
# Let's now fit a quadratic model and use the bootstrap
boot.fn <- function (data, index) {
  coefficients(lm(mpg~horsepower+I(horsepower^2),data=data,subset=index))
}
set.seed(1)
boot(Auto, boot.fn, 1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##         original        bias     std. error
## t1* 56.900099702  6.098115e-03 2.0944855842
## t2* -0.466189630 -1.777108e-04 0.0334123802
## t3*  0.001230536  1.324315e-06 0.0001208339
```

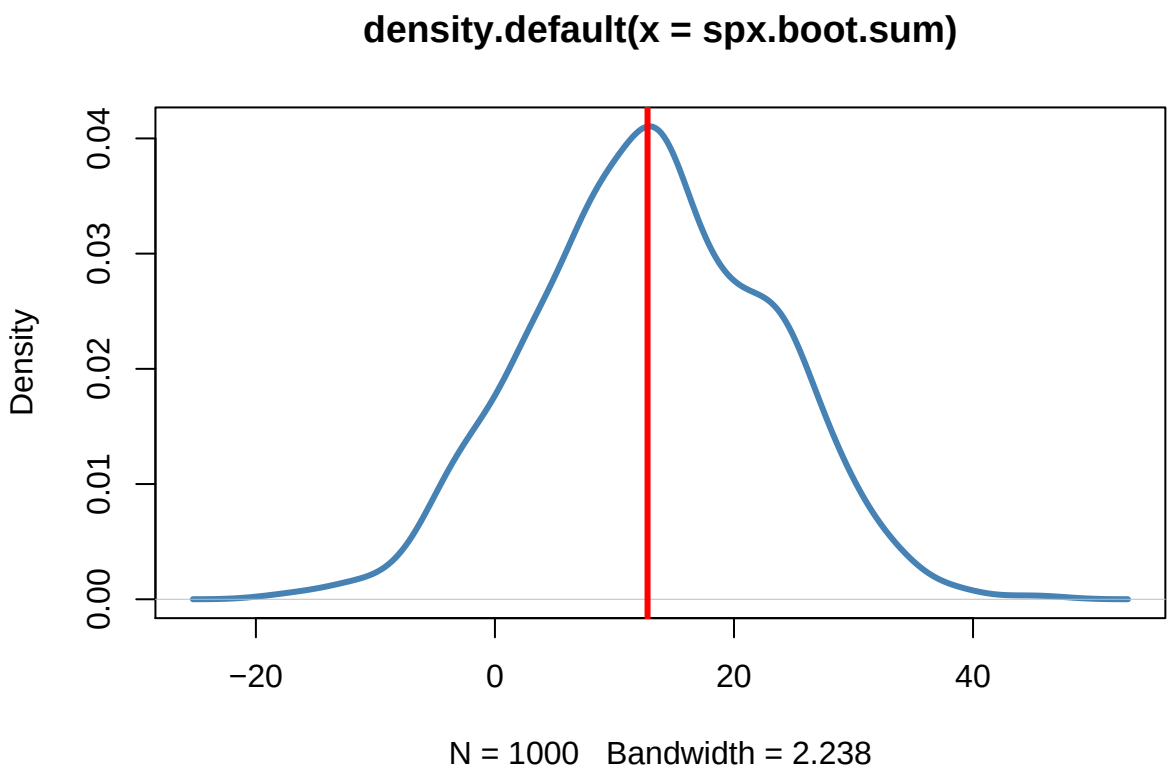
```r
# and compare against the lm calculated SE values
summary(lm(mpg~horsepower+I(horsepower^2),data=Auto))$coef
```

```
##                     Estimate   Std. Error   t value      Pr(>|t|)
## (Intercept)     56.900099702 1.8004268063  31.60367 1.740911e-109
## horsepower      -0.466189630 0.0311246171 -14.97816  2.289429e-40
## I(horsepower^2)  0.001230536 0.0001220759  10.08009  2.196340e-21
```

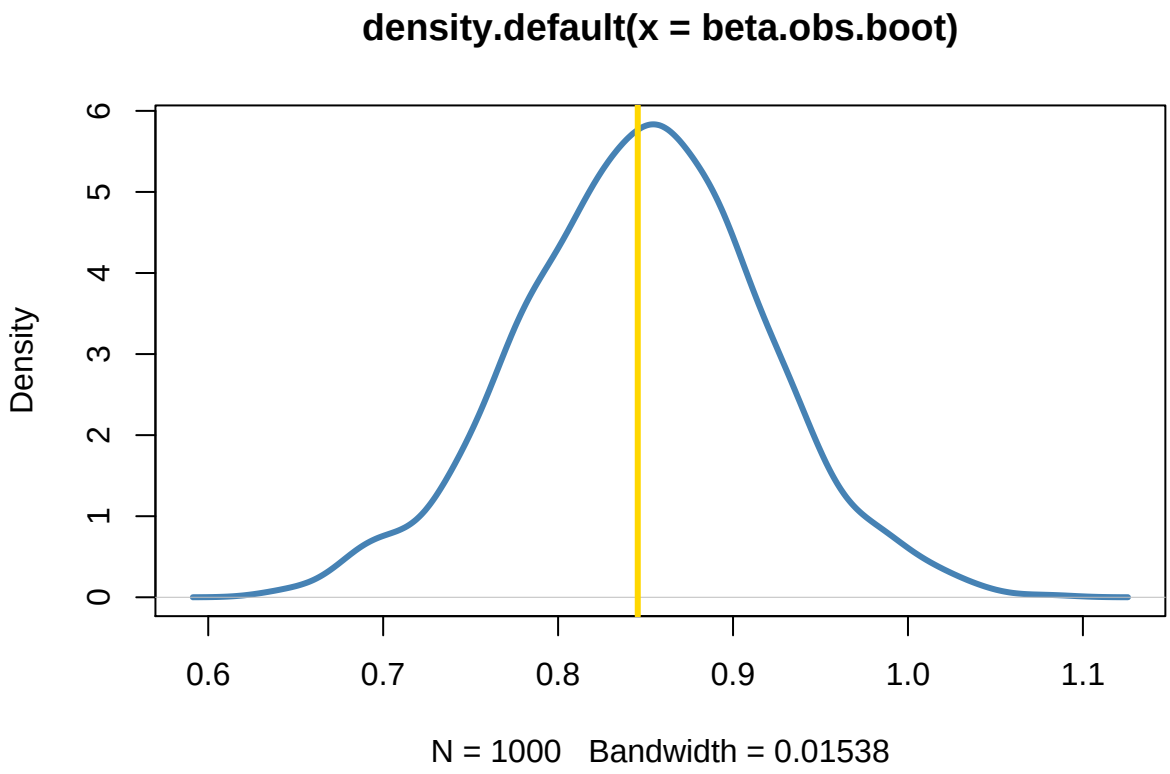
Bootstrap for the Burns databaset

The explanations for this example, which deals with estimating the IBM daily returns as a function of the S&P index, are documented within the code itself or you could find the full description here.

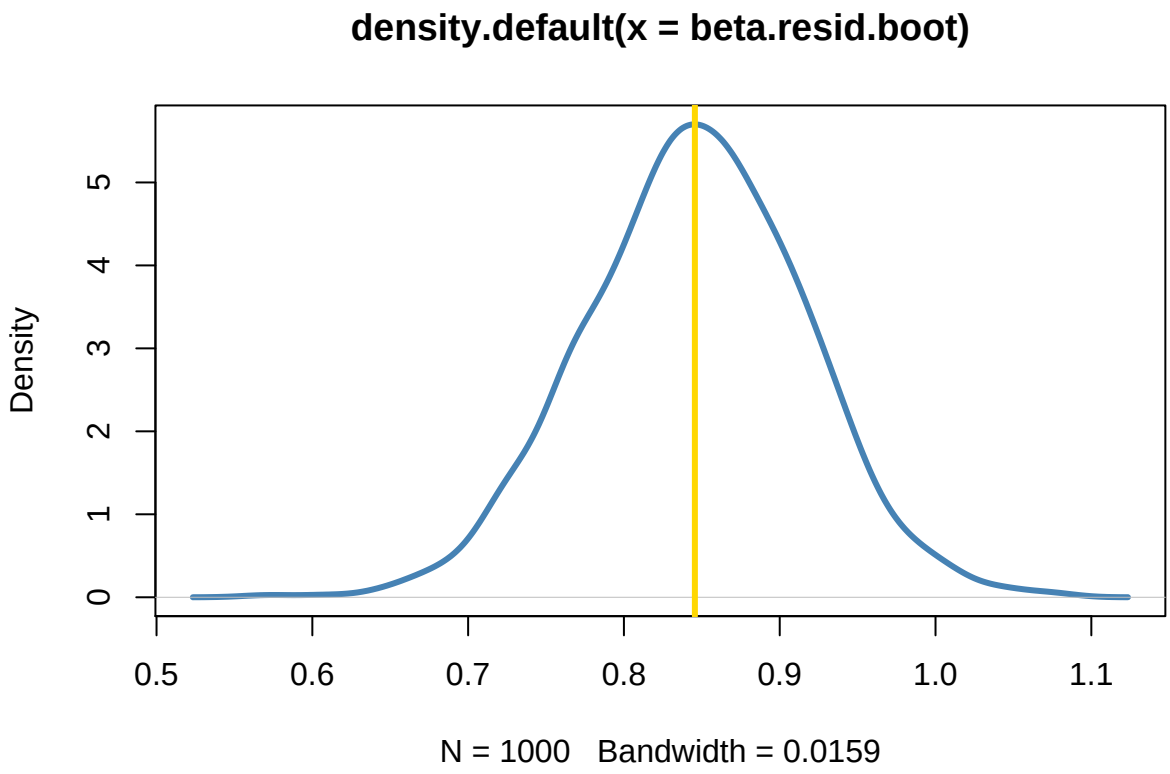
```
# Bootstrap on the Burns dataset
# (http://www.burns-stat.com/documents/tutorials/the-statistical-bootstrap-and-other-resampling-methods
spxibm <- as.matrix(read.table(
  "http://www.burns-stat.com/pages/Tutor/spx_ibm.txt",
  header=TRUE, sep='\t', row.names=1))
View(spxibm)
# extract the separate indices
spxret <- spxibm[, "spx"]
ibmret <- spxibm[, "ibm"]
# We want to calculate log returns (sum of the returns for the year)
# Here is how we put the bootstrap to use
spx.boot.sum <- numeric(1000)          # create a 1000 long numeric vector
for (i in 1:1000) {                    # set up the loop to create 1000 samples
  this.sample <- spxret[ sample(251, 251, replace = TRUE) ]
  # above: select 251 values at random with replacement
  spx.boot.sum[i] <- sum(this.sample) # calculate the sum of daily returns for this sample
}
# Since we now have 1000 samples for log returns, let's plot the density function
# (or a histogram) for the sample distribution
# and superimpose the actual value from the original file not sampled
plot(density(spx.boot.sum), lwd=3, col="steelblue")
abline(v = sum(spxret), lwd=3, col="red")
```




```
# If I were trying to estimate the ibm return from the spx return using a linear model
# we could bootstrap the value of the linear coefficient (or the intercept for that matter)
# as follows
beta.obs.boot <- numeric(1000)     # create a 1000 long numeric vector
for (i in 1:1000) {               # set up the loop to create 1000 samples
  this.sample <- sample(251, 251, replace = T)
  beta.obs.boot[i] <- coef(lm(ibmret[this.sample]~spxret[this.sample]))[2]
}
# plot the results
plot(density(beta.obs.boot), lwd=3, col="steelblue")
abline(v=coef(lm(ibmret ~ spxret))[2], lwd=3, col='gold')
```




```r
# Instead of bootstrappint the linear regression coefficient we could bootstrap the
# residuals, as follows
ibm.lm <- lm(ibmret ~ spxret)
ibm.fit <- fitted(ibm.lm)
ibm.resid <- resid(ibm.lm)
beta.resid.boot <- numeric(1000)
for (i in 1:1000) {
  this.sample <- sample(251, 251, replace=TRUE)
  beta.resid.boot[i] <- coef(lm(ibm.fit + ibm.resid[this.sample] ~ spxret))[2]
}
plot(density(beta.resid.boot), lwd=3, col="steelblue")
abline(v=coef(lm(ibmret ~ spxret))[2], lwd=3, col='gold')
```



Not too shabby!