5 Resampling

By: Udit (based on ISLR)

Setup

Bootstrapping courtesy of the **boot** package. Includes **cv.glm()** function used for cross-validation (including LOOCV).

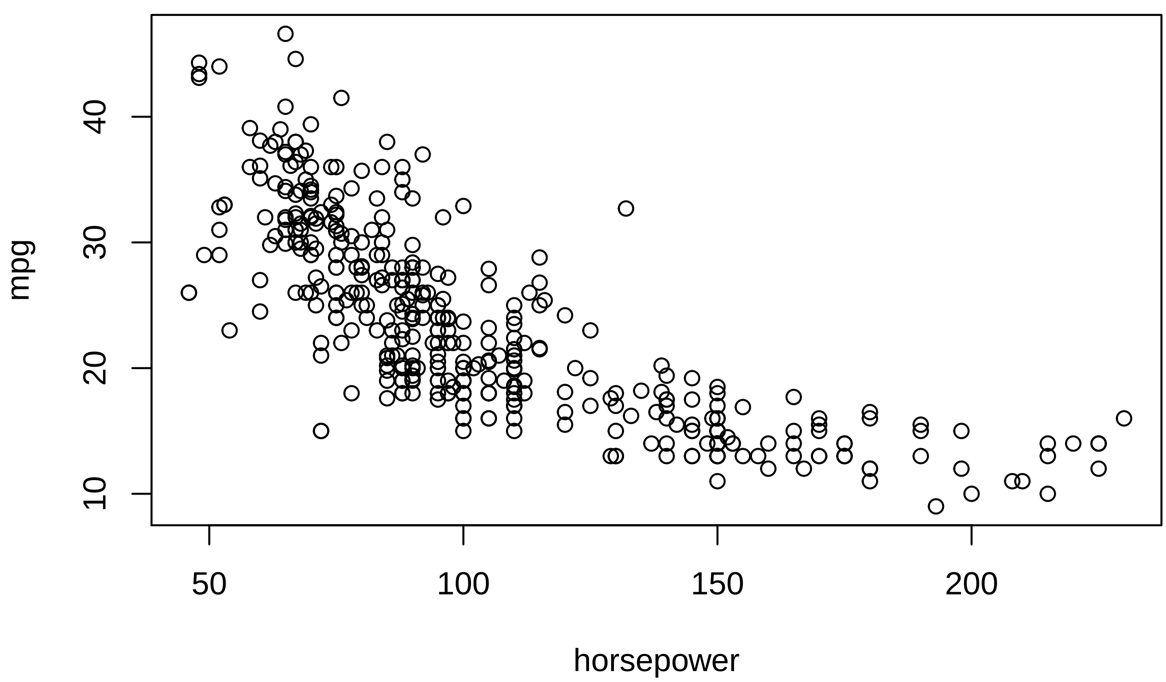
```
library(ISLR2)
library(boot)

attach(Auto)
names(Auto)
```

```
## [1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
## [6] "acceleration" "year"         "origin"       "name"
```

```
dim(Auto)
```

```
## [1] 392   9
```

```
plot(mpg~horsepower, data=Auto)
```



LOOCV

cv.glm() is general implementation, therefore does not use the formula approach available in Least-Square fit/ Simple Regression case.

cv.glm()$delta is a vector of length two. The first component is the raw cross-validation estimate of prediction error. The second component is the adjusted cross-validation estimate. The adjustment is designed to compensate

lm.influence() part of regression diagnostic to check for quality of fit.

```
glm.fit = glm(mpg~horsepower, data=Auto)
summary(glm.fit)  #GLM works, though LM provides better output summary
```

```
##
## Call:
## glm(formula = mpg ~ horsepower, data = Auto)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -13.5710   -3.2592   -0.3435    2.7630   16.9240
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.935861   0.717499   55.66   <2e-16 ***
## horsepower  -0.157845   0.006446  -24.49   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 24.06645)
##
##     Null deviance: 23819.0  on 391  degrees of freedom
## Residual deviance:  9385.9  on 390  degrees of freedom
## AIC: 2363.3
##
## Number of Fisher Scoring iterations: 2
```

```
#summary(lm(mpg~horsepower, data=Auto))

## LOOCV
cv.glm(Auto, glm.fit)$delta  #k (folds) = n (# of obs) by default
```

```
## [1] 24.23151 24.23114
```

```
## Brute-force implementation
run.LOOCV = function(data){
  n = nrow(data)
  error = 0
  for(i in 1:n){
    test = data[i,]
    train = data[-i,]
    glm.fit = glm(mpg~horsepower, data=train)
    glm.pred = predict(glm.fit, newdata=test)
    error = error + (glm.pred-test$mpg)^2
  }
  print(error/n)
}
run.LOOCV(Auto) # 24.23151
```

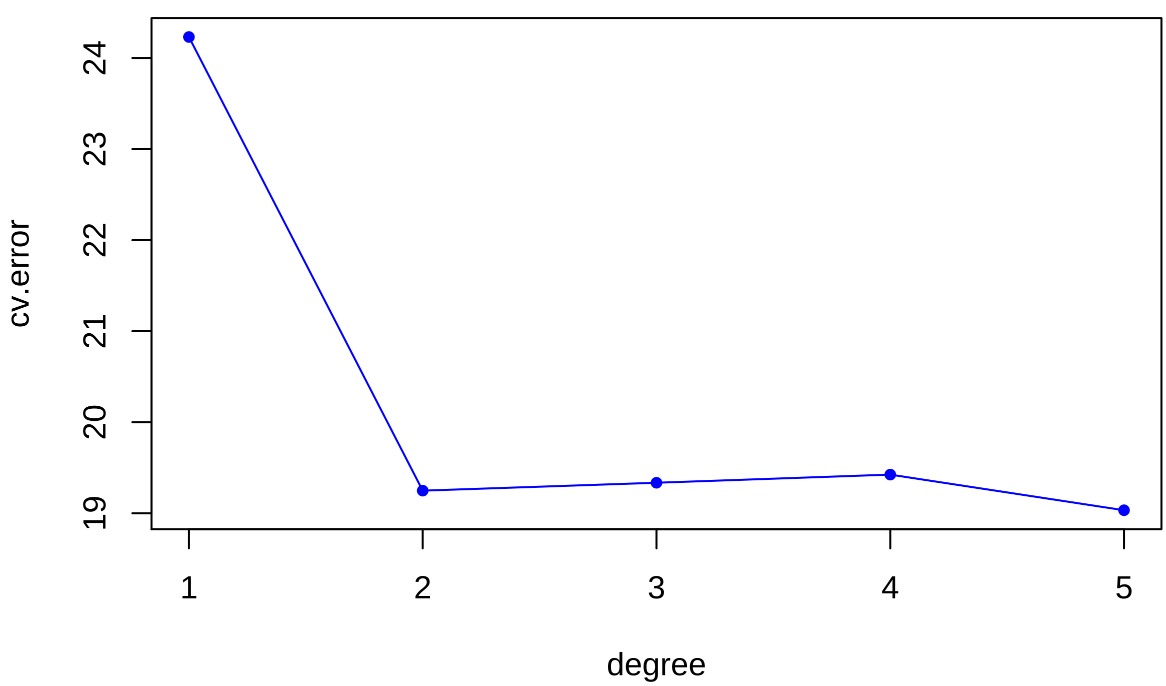
```
##         1
## 24.23151
```

```
## Leverage/'h' formula based implementation
loocv = function(fit){
  h = lm.influence(fit)$h
  mean((residuals(fit)/(1-h))^2)
}
loocv(glm.fit)  # 24.23151
```

```
## [1] 24.23151
```

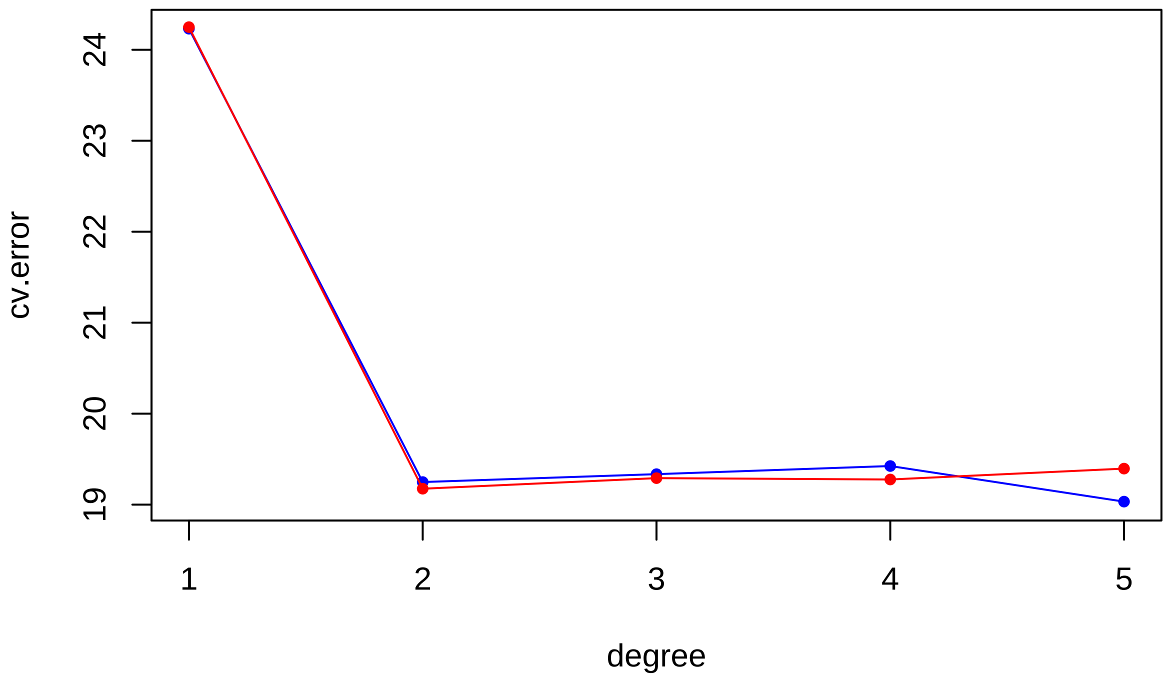
LOOCV - for model selection

```
degree = 1:5
cv.error = rep(0,5)

for(d in degree){
  glm.fit=glm(mpg~poly(horsepower,d), data=Auto)
  cv.error[d] = loocv(glm.fit)
}

plot(degree, cv.error, type="o", col="blue", pch=20)
```



Cross Validation: 10-Fold

```
cv.error10=rep(0,5)
for(d in degree){
  glm.fit=glm(mpg~poly(horsepower,d), data=Auto)
  cv.error10[d] = cv.glm(Auto, glm.fit, K=10)$delta[1]
}
plot(degree, cv.error, type="o", col="blue", pch=20)
lines(degree, cv.error10, type="o", col="red", pch=20)
```



Bootstrap

```
# Optimum allocation (alpha) b/w two securities X, Y to minimize variance
alpha= function(x,y){
  vx = var(x)
  vy = var(y)
  cxy = cov(x,y)
  (vy-cxy)/(vx+vy-2*cxy)
}

# Portfolio dataset in ISLR2
attach(Portfolio)
names(Portfolio)
```

```
## [1] "X" "Y"
```

```
dim(Portfolio)
```

```
## [1] 100   2
```

```r
# Optimum allocation at ~ 0.6
alpha(X,Y)
```

```
## [1] 0.5758321
```

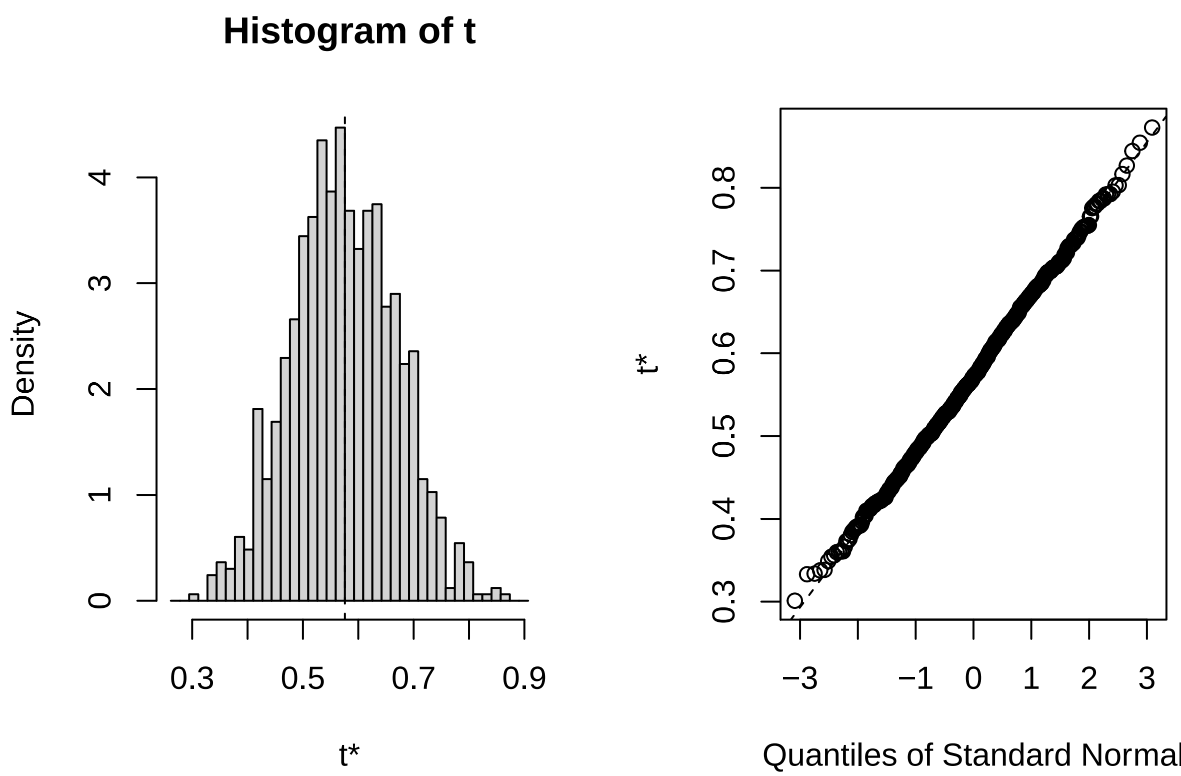
```r
# Estimate alpha for bootstrapped sample indicated by 'index'
alpha.fn = function(data, index){
  with(data[index,], alpha(X,Y))
}

# Bootstrapping
set.seed(1)
alpha.fn(Portfolio, sample(1:100, 100, replace=TRUE))
```

```
## [1] 0.7368375
```

```r
boot.out = boot(Portfolio, alpha.fn, R=1000)
boot.out
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Portfolio, statistic = alpha.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original       bias    std. error
## t1* 0.5758321 -0.001695873  0.09366347
```

```r
plot(boot.out)
```



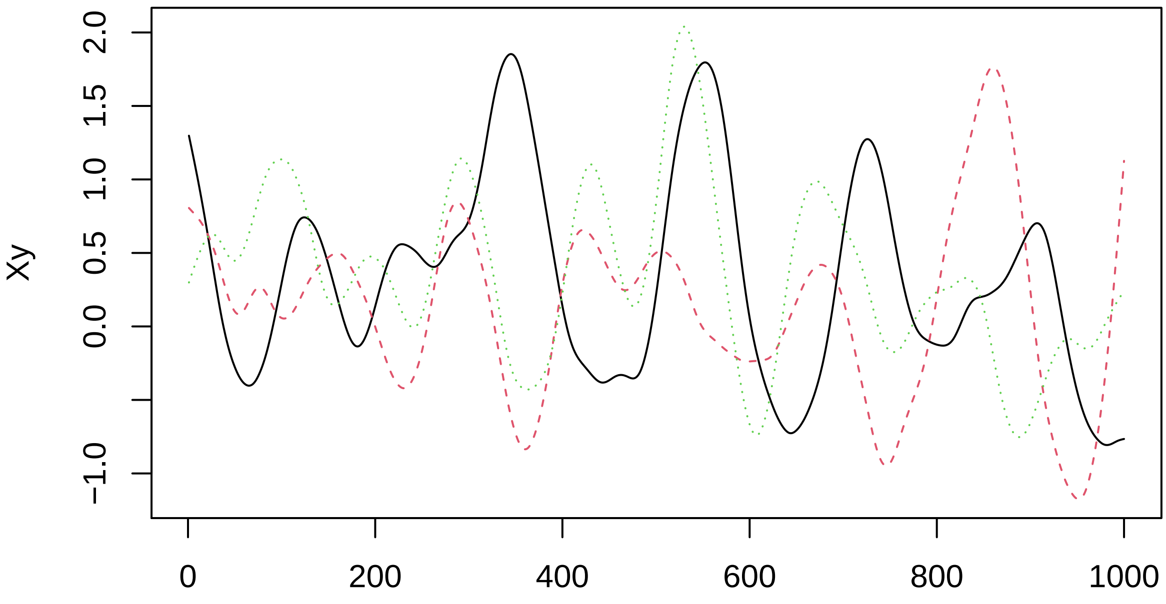
Block Bootstrapping

Note 1: There is very strong autocorrelation between consecutive rows of the data matrix. Roughly speaking, we have about 10-20 repeats of every data point, so the sample size is in effect much smaller than the number of rows (1000 in this case).

```
load("5.R.RData")
mod = lm(y~., data=Xy)
summary(mod)  # see Note 1
```

```
##
## Call:
## lm(formula = y ~ ., data = Xy)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.44171 -0.25468 -0.01736  0.33081  1.45860
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.26583    0.01988  13.372  < 2e-16 ***
## X1           0.14533    0.02593   5.604 2.71e-08 ***
## X2           0.31337    0.02923  10.722  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5451 on 997 degrees of freedom
## Multiple R-squared:  0.1171, Adjusted R-squared:  0.1154
## F-statistic: 66.14 on 2 and 997 DF,  p-value: < 2.2e-16
```

```r
matplot(Xy, type="l")
```




```r
# Bootstrapping to estimate SE(X1) - similar as above
se.fn = function(data, index){
  mod = lm(y~., data=data[index,])
  coef(mod)
  #coef(summary(mod))[, "Std. Error"][[2]]
}
boot.out = boot(Xy, se.fn, R=1000)
boot.out
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Xy, statistic = se.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1* 0.2658349 1.286777e-05  0.01436685
## t2* 0.1453263 1.197267e-03  0.02839712
## t3* 0.3133670 1.558673e-03  0.03515003
```

```r
# Block Sampling - using 10 contiguous blocks of 100 obs each
se.fn1 = function(data, index){
  newXY = Xy[data[index]+rep(0:99, each=10),]
  mod = lm(y~., data=newXY)
```

```r
  coef(mod)
}
s = seq(1,1000,100)
boot.out = boot(s, se.fn1, R=1000)
boot.out  #0.2 ... ~10x when we take auto-correlation into account
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = s, statistic = se.fn1, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.2658349 0.003003285  0.08808016
## t2* 0.1453263 0.005589519  0.19027500
## t3* 0.3133670 0.091483471  0.32885745
```