Lab 5 - Cross-Validation and the Bootstrap

An Introduction to Statistical Learning

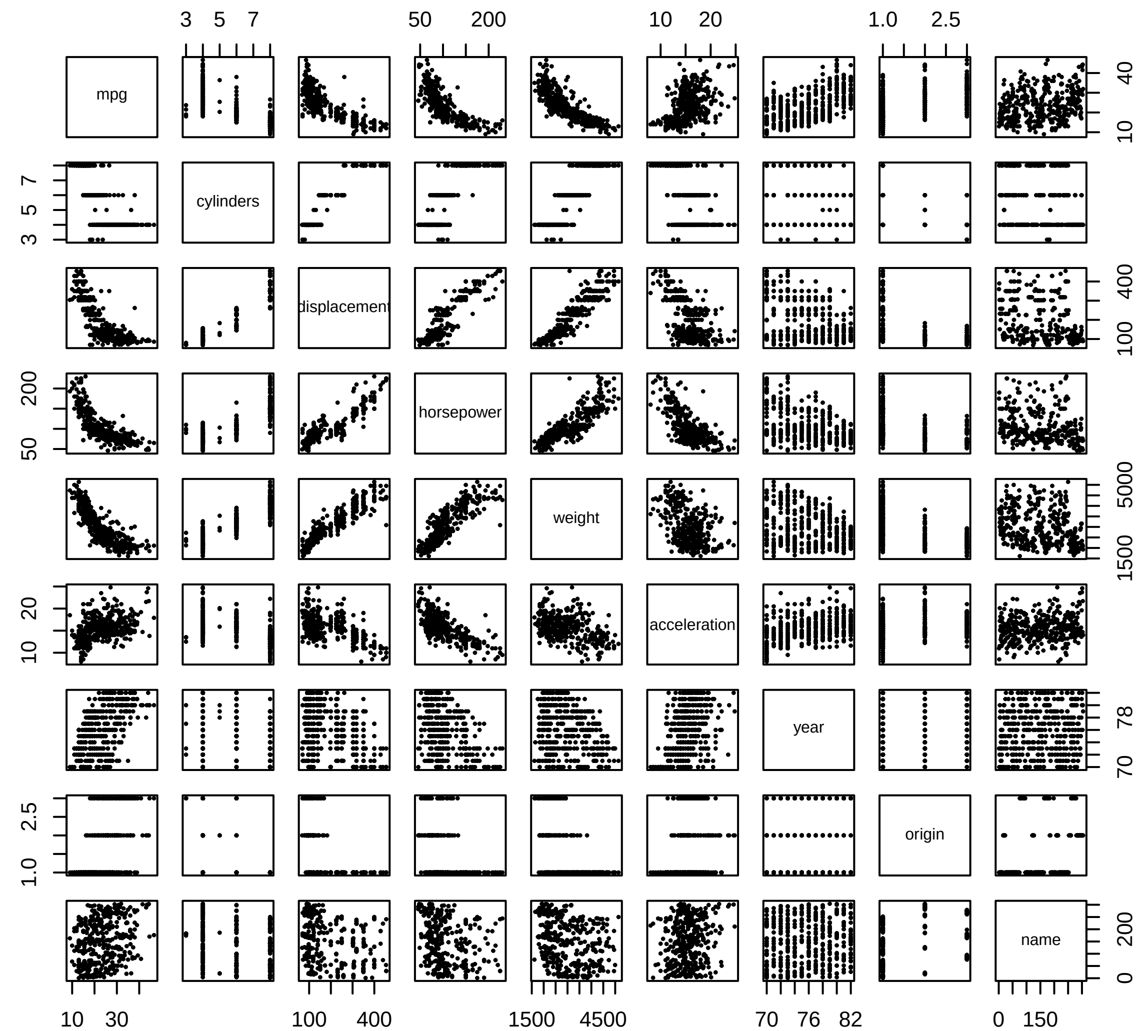
We will be using the `Auto` dataset.

```
library(ISLR)
attach(Auto)
summary(Auto)
```

```
##       mpg          cylinders      displacement     horsepower        weight
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140
##
##   acceleration        year           origin                      name
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000   amc matador       :  5
##  1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
##  Median :15.50   Median :76.00   Median :1.000   toyota corolla    :  5
##  Mean   :15.54   Mean   :75.98   Mean   :1.577   amc gremlin       :  4
##  3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
##  Max.   :24.80   Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
##                                                  (Other)           :365
```

Let's plot the data:

```
pairs(Auto, cex=.25)
```



First of all, we will set the seed to ensure reproducibility:

```
set.seed(1)
```

1. The Validation Set approach

In the previous examples we split the data in two subsets, a *training set* and a *test set*, and the split was done by hand, at a given index of the data.

Random splits can be done using `sample()`; this function "samples" of a specified size from a set. Instead of a source set, an integet number can be passed, taking samples from $1 : n$.

We take half the data as training set and the remaining half for testing:

```
dim(Auto)
```

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

```
N = dim(Auto)[1]
train = sample(N, N/2, replace = FALSE)
```

We have created a vector with $392/2 = 196$ values, from 1 to $N = \text{dim(Auto)[1]}$ that will be used as indices for the training samples.

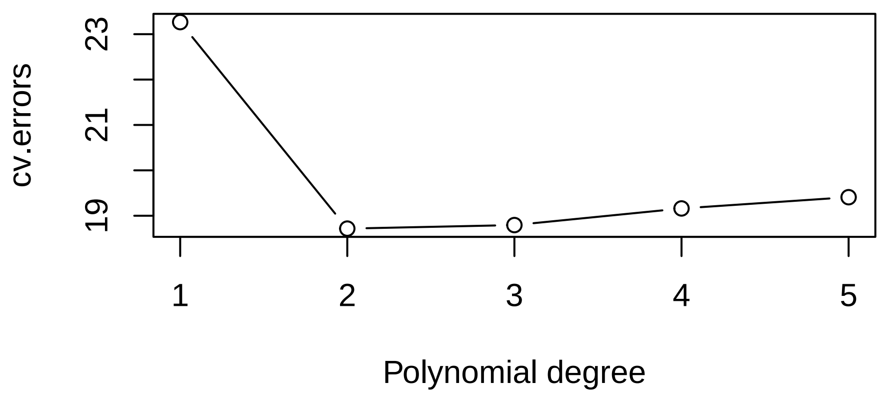
Fitting the model

We fit a simple Linear Model using `horsepower` as predictor for `mpg`. We will choose between 3 possible models, with polynomial degrees up to 3 for `horsepower`.

```
cv.errors = rep(NA, 5)
for (d in 1:5) {
  lm.fit = lm(mpg ~ poly(horsepower, d), data = Auto, subset = train)
  lm.pred = predict(lm.fit, newdata = Auto[-train])
  cv.errors[d] = mean((lm.pred - mpg)[-train]^2)
}
cv.errors
```

```
## [1] 23.26601 18.71646 18.79401 19.16017 19.40812
```

```
plot(cv.errors, type='b', xlab = 'Polynomial degree')
```



Based on this results we would choose the model with the smallest mean LSE error, in this case the model with the second-order polynomial.

2. Leave-One-Out Cross-Validation (*LOOCV*)

LOOCV can be done using the `boot` library:

```
library(boot)
```

This library includes methods for computing Cross-Validation for any generalized linear model using `glm()` and `cv.glm()`. `glm()` was used before to perform logistic regression using `family="binomial"`, but if no `type` is passed then it will perform linear regression, like `lm()`, with the advantage that we can use `cv.glm()` for cross-validation.

We will fit a simple model to show the syntax for `cv.glm()`:

```r
glm.fit = glm(mpg ~ horsepower, data = Auto)
cv.err = cv.glm(data = Auto, glmfit = glm.fit, K = 10)
```

We obtain an object with the following components:

```r
names(cv.err)
```

```
## [1] "call"  "K"     "delta" "seed"
```

`delta` contains the CV results:

```r
cv.err$delta
```

```
## [1] 24.28315 24.26490
```

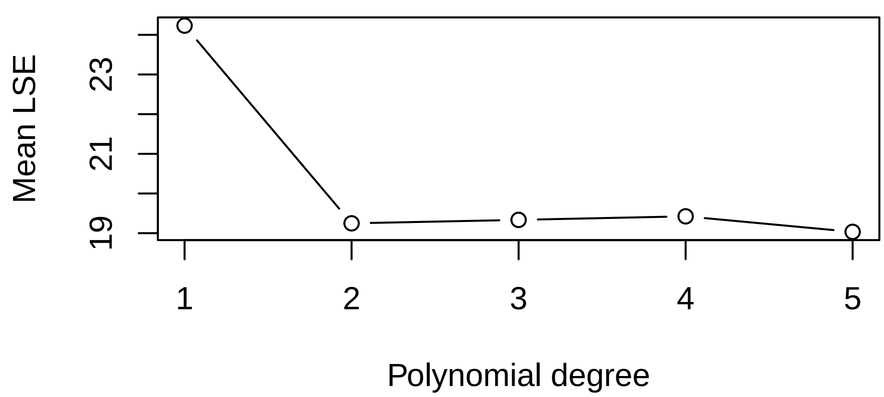
The first element of `delta` is the standard CV estimate, while the second is a bias-compensated estimation. These two values will be different, specially when using LOOCV, and will be much more similar when doing K-Fold CV.

We will now perform *LOOCV* to find the best polynomial degree for the fit:

```r
cv.errors.loo = rep(NA, 5)
for (d in 1:5) {
  glm.fit = glm(mpg ~ poly(horsepower, d), data = Auto)
  cv.errors.loo[d] = cv.glm(Auto, glm.fit)$delta[1]
}
cv.errors.loo
```

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

```r
plot(cv.errors.loo, type='b', xlab = 'Polynomial degree', ylab='Mean LSE')
```



The results show the mean LSE for each degree.

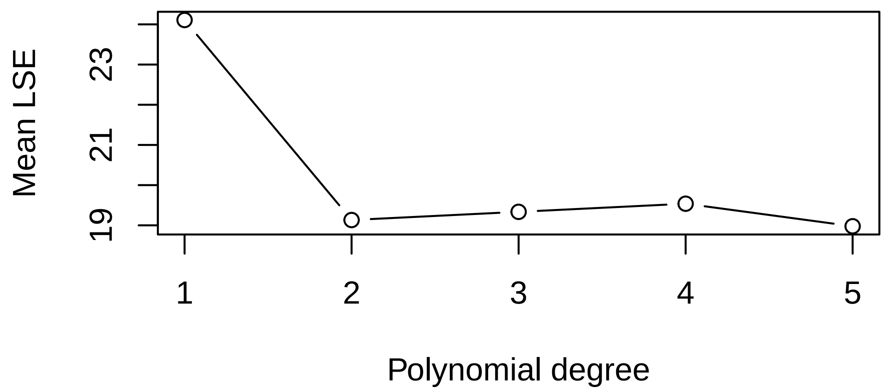
3. K-Fold Cross-Validation

K-Fold CV can be done using `cv.glm()` with another parameter K, the number of folds in which the data will be split. K defaults to 10, meaning that 9 folds will be used to train the data and the remaining one will be used to make predictions. This step will be done K times, using $K-1$ of the K folds to train the model and the remaining one to test the performance.

```
cv.errors.kfold = rep(0, 5)
for (d in 1:5) {
  glm.fit = glm(mpg ~ poly(horsepower, d), data = Auto)
  cv.errors.kfold[d] = cv.glm(Auto, glm.fit, K = 10)$delta[1]
}
cv.errors.kfold
```

```
## [1] 24.10821 19.13160 19.33517 19.53766 18.97701
```

```
plot(cv.errors.kfold, type='b', xlab = 'Polynomial degree', ylab='Mean LSE')
```



4. The Bootstrap

The bootstrap is a widely applicable and powerful statistical method used to quantify the *uncertainty* of a given estimator or statistical learning method.

To perform a bootstrap analysis two steps are necessary:

a. Create a function that computes the statistic of interest.
b. Use the `boot()` method, from the `boot` library, to perform the bootstrap by repeatedly sampling observations from the data.

Estimating the accuracy of a statistic

We will use the `Portfolio` dataset, included in the `ISLR` library.

```
summary(Portfolio)
```

```
##        X                  Y
##  Min.   :-2.43276   Min.   :-2.72528
##  1st Qu.:-0.88847   1st Qu.:-0.88572
##  Median :-0.26889   Median :-0.22871
##  Mean   :-0.07713   Mean   :-0.09694
##  3rd Qu.: 0.55809   3rd Qu.: 0.80671
##  Max.   : 2.46034   Max.   : 2.56599
```

```
dim(Portfolio)
```

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

We wish to invest a fixed sum of money in two financial assets that yield returns X and Y. We will invest a fraction α of the money in X and the remaining, $1-\alpha$, in Y. We wish to choose α that minimizes the risk (variance) of the investment, i.e. that minimizes $\mathrm{Var}(\alpha X + (1-\alpha)Y)$, being the result:

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$$

Create the function that computes the statistic α

We are going to sample observations from the data, obtaining a vector of indices to subset the data. So, we will pass the function both the data and the vector of indices so slice the data.

```
alpha.fn = function(data, index) {
  X = data$X[index]
  Y = data$Y[index]
  res = (var(Y) - cov(X, Y)) / (var(X) + var(Y) - 2 * cov(X, Y))
  return(res)
}
```

Each time we call `alpha.fn()` with the dataset and a vector of indices it will return the value of α that minimizes the variance for that subset.

```
alpha.fn(Portfolio, 1:100)
```

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

To generate the indices we will use the `sample()` method to randomly select N observations from the range $1 : N$, **with replacement**, where N is the total number of observations in the dataset.

```
set.seed(17)
ix = sample(100:100, replace = TRUE)
alpha.fn(Portfolio, ix)
```

```
## [1] 0.5817589
```

Perform the bootstrap

To implement the bootstrap analysis we repeatedly call `alpha.fn()`, using different samples from the dataset.

This can be automatically done using `boot()`; we can specify the number of iterations with the parameter `R`:

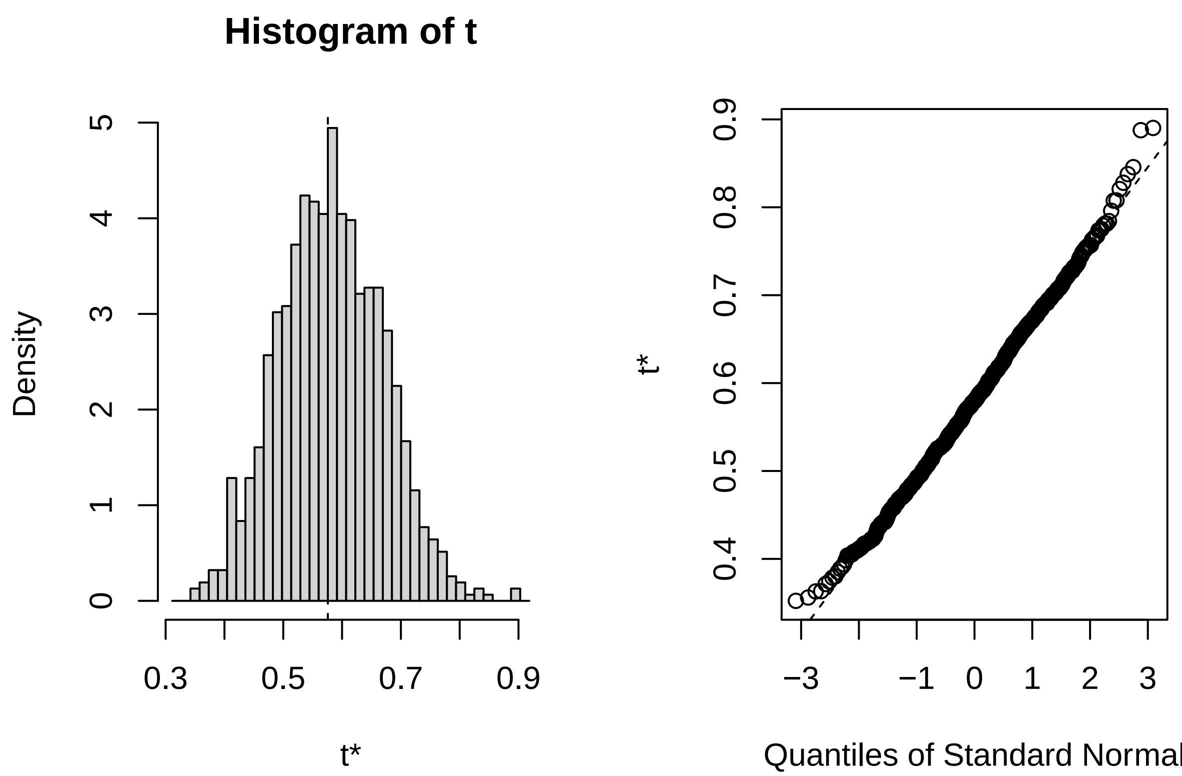
```
boot =boot(Portfolio, alpha.fn, R = 1000)
boot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Portfolio, statistic = alpha.fn, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.5758321 0.005565359  0.08811696
```

The method returns an estimated value for $\hat{\alpha} = 0.5758$ with a bootstrap estimate for its standard error, $SE(\hat{\alpha}) = 0.0881$.

We can plot the output:

```
plot(boot)
```



We can access the results:

```
alpha = boot$t0
se = sd(boot$t)
cat(sprintf("alpha = %0.3f, SE = %0.4f", alpha, se))
```

```
## alpha = 0.576, SE = 0.0881
```

Estimating the accuracy of a Linear Regression Model

The bootstrap can be used to asses the variability of the coefficient estimates and predictions from a statistical learning method.

We will use the `Auto` dataset.

We first create a function to compute the values of interest, in this case the intercept and slope of the Linear Regression model:

```
lr.fn = function(data, index) {
  model = lm(mpg ~ horsepower, data = Auto, subset = index)
  coefs = coef(model)
  return(coefs)
}
```

This function will return the intercept and slope for a Linear Regression model fitted with the subset defined by `index`:

```
lr.fn(Auto, 1:100)
```

```
## (Intercept)  horsepower
##  31.4601036  -0.1010577
```

Now we perform the bootstrap analysis:

```
boot = boot(Auto, lr.fn, R = 1000)
boot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = lr.fn, R = 1000)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 39.9358610  0.0655811735  0.85119330
## t2* -0.1578447 -0.0006899326  0.00725396
```

The output shows the bootstrap estimates $\hat{\beta}_0 = 39.936$ and $\hat{\beta}_1 = -0.158$ with estimated standard errors $SE(\hat{\beta}_0) = 0.8512$ and $SE(\hat{\beta}_1) = 0.0072$.