5 Resampling Methods

Notes

- Used in the absence of very large designated test set (which is very commmon)

Cross Validation (CV)

- Leave-One-Out CV
  - using a single observation (x_i, y_i) for the validation set while remaining observations make up training set
  - and thus $MSE_i = (y_i - \hat{y}_i)^2$
  - repeating this n times, we get the test MSE is the avg of n test error estimates:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$$

- doesn't tend to overestimate test error
- will always yield same results since there is no randomness in training/validation set splits
- k-Fold CV
  - randomly dividing observations in to k groups of equal/approximate size
  - first fold treated as validation set, and method is fit on remaining k-1 folds
  - calculate MSE on held-out folds, up to k times
  - averaging these MSEs we get:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

- k = 5 or 10 usually
- bias-variance trade-off in CV
  - from a bias reduction POV, LOOCV is preferred to k-fold
  - from a variance reduction POV, k < n folds CV is preferred
- for classification problems, we use misclassified observations instead of MSE

Bootstrap

- obtain distinct data sets by repeatedly sampling observations with replacement from the original data set
    - usually used to estimate the accuracy of a Statistic of Interest (such as variability of coefficient estimates and predictions)
- does not rely on the assumption that all variability comes from the error terms ϵ_i
    - likely give a more accurate estimate of standard errors of coefficents β_i than from the summary() in the original model

Applied

```r
library(boot)
library(ISLR)
library(MASS)
```

7) (Calculating LOOCV error)

a)

```r
week.glm <- glm(Direction ~ Lag1 + Lag2, data = Weekly, family = "binomial")
```

b)

```r
#logistic model without 1st observation
week.glm1 <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-1,], family = "binomial")
```

c)

```r
#predict 1st observation using trained log model
week.probs <- predict(week.glm1, newdata = Weekly[1,], type = "response")
week.pred <- rep("Down", length(week.probs))
week.pred[week.probs > .5] <- "Up"

mean(week.pred == Weekly$Direction[1])
```

```
## [1] 0
```

No, it did not classify this observation correctly.

d)

```r
error <- rep(0, nrow(Weekly))

for(i in 1:nrow(Weekly)){
  #fitting log model using all but i'th observation up to n
  week.glmi <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-i, ], family = "binomial")

  #if prob > 0.5, classify response as Up
  pred.up <- predict.glm(week.glmi, Weekly[i,], type = "response") > 0.5
  true.up <- Weekly[i,]$Direction == "Up"

  #if model prediction doesn't match actual response, give 1 on the corresponding error index
  if(pred.up != true.up){
    error[i] <- 1
  }
}
mean(error)
```

```
## [1] 0.4499541
```

Thus, the model has around a 45% test error for the LOOCV method. The relatively high error rate could be attributed to the fact that LOOCV overfitted on the training data and thus has poor prediction rates.

8) (LOOCV of a generated dataset)

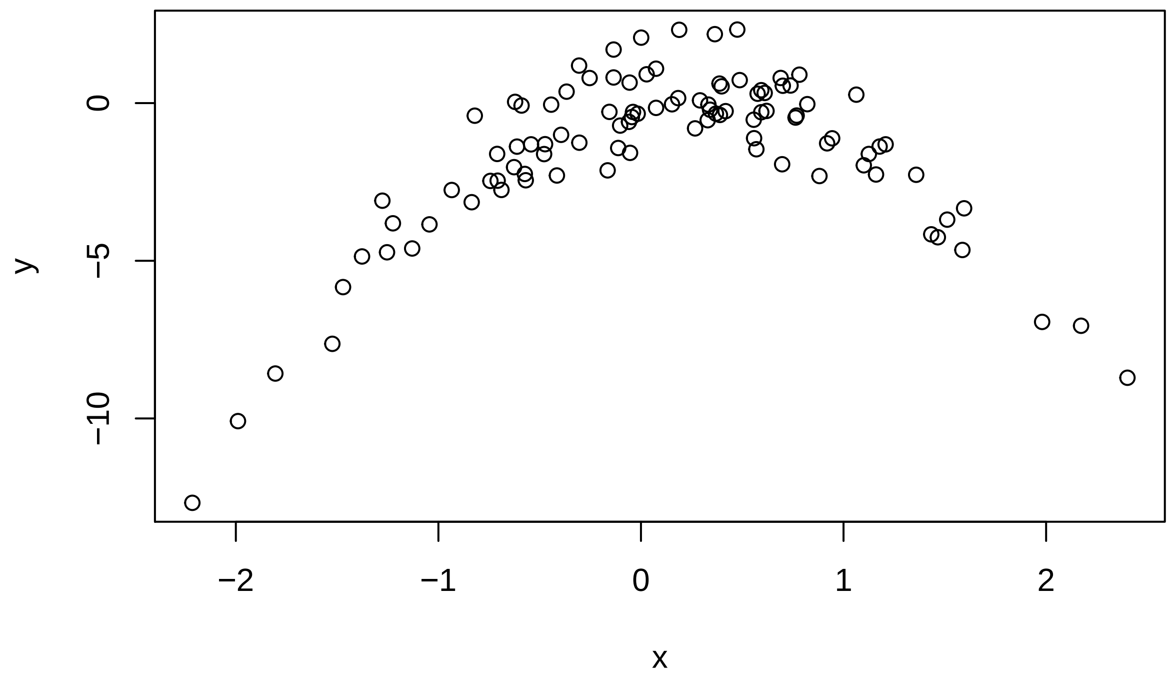
a)

```r
set.seed(1)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

In this generated dataset, n (# of observations) is 100 and p (# of predictors) is 2.

b)

```r
plot(x,y)
```



There seems to be an inverse quadratic relationship between X and Y.

c)

```r
set.seed(1)

df <- data.frame(x,y)

#calculating LOOCV errors for x predictor up to power of 4
cv.err1 <- rep(0,4) #to store LOOCV errors
for(i in 1:4){
  lm1.fit <- glm(y ~ poly(x,i), data = df,)
  cv.err1[i] <- cv.glm(df,lm1.fit, K = nrow(df))$delta[1]
}
cv.err1
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

d)

```r
#repeat part c) with different seed
set.seed(2)

df <- data.frame(x,y)

cv.err2 <- rep(0,4)
for(i in 1:4){
  lm2.fit <- glm(y ~ poly(x,i), data = df)
  cv.err2[i] <- cv.glm(df,lm2.fit, K = nrow(df))$delta[1]
}
cv.err2
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

Yes, since we are using LOOCV with k = n folds, we have the exact same test error values as the ones in part c)

e)

It seems that the quadratic model (power of 2) had the lowest LOOCV error. This is expected since it looked like the relationship between X and Y were a quadratic form

f)

```r
summary(lm1.fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, i)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, i)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, i)3   0.26411    0.95905   0.275    0.784
## poly(x, i)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

```
summary(lm2.fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x, i), data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(x, i)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(x, i)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(x, i)3   0.26411    0.95905   0.275    0.784
## poly(x, i)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

From the output, it looks like the model with the 2nd order term had the most statistical significance. Thus, it agreed with the conclusions from the CV results.

9) (Bootstrapping statistics)

a)

```
mu.hat <- mean(Boston$medv); mu.hat
```

```
## [1] 22.53281
```

$\hat{\mu}$ of medv is roughly 22.5 from the data

b)

```r
muhat.se <- sd(Boston$medv)/sqrt(nrow(Boston))
```

The standard error of $\hat{\mu}$ is roughly around 0.41, which indicates that most of the data is closely centered around $\hat{\mu}$.

c)

```r
set.seed(1)

#create function to sample mean for bootstrap
mean.fn <- function(data,index){
  mu <- mean(data[index])
  return(mu)
}

boot.muhat <- boot(Boston$medv,mean.fn, R = 1000) #1000 bootstrap iterations
boot.muhat
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = mean.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 22.53281 0.007650791   0.4106622
```

The SE is very close to that from part b)

d)

```r
#95% CI for bootstrap muhat
CI.bootmuhat <- c(boot.muhat$t0 - 2*.4106622, boot.muhat$t0 + 2*.4106622)
CI.bootmuhat
```

```
## [1] 21.71148 23.35413
```

```r
t.test(Boston$medv)
```

```
##
##  One Sample t-test
##
## data:  Boston$medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281
```

The CI's for both methods are very close

e)

```
med.hat <- median(Boston$medv)
med.hat
```

```
## [1] 21.2
```

sample median is 21.2

f)

```
#function for median
med.fn <- function(data,index){
  med <- median(data[index])
  return(med)
}

boot.medhat <- boot(Boston$medv,med.fn, R = 1000)
boot.medhat
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = med.fn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*     21.2 -0.0386   0.3770241
```

We can see that the SE of the bootstrapped median is around .369, which is pretty small relative to the median value

g)

```r
quant10 <- quantile(Boston$medv, probs = 0.1); quant10
```

```
##   10%
## 12.75
```

10th quantile of medv is 12.75

f)

```r
quant10.cn <- function(data,index) {
  quant10th <- quantile(data[index], probs = .1)
  return(quant10th)
}

boot.quant10 <- boot(Boston$medv,quant10.cn, R = 1000)
boot.quant10
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Boston$medv, statistic = quant10.cn, R = 1000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    12.75  0.0186   0.4925766
```

SE of the 10th quantile is around .51, which is still pretty low relative to the median