Chapter 5: Resampling Methods

Brook Chuang

Ch 5 Resampling Methods

Re-sampling methods include repeatedly drawing samples from a training set and refitting the model of interest on each sample in order to obtain additional information about the fitted model.

The most common re-sampling method 1. cross-validation 2. bootstrap

Cross validation can be used to estimate the test error associated with a given statistical learning method in order to evaluate performance or to select level of flexibility.

Model assessment: evaluating model's performance *model selection*: process of selecting the proper level of flexibility

Bootstrap is used to measure accuracy of a parameter estimate or learning method

5.1 Cross-validation

Test error rate: average error that results from using a statistical learning method to predict the response on a new observation - a learning method can be used if the test error rate is low - easily calculated if test set is available (bu usually unlikely)

But the test error rate is not a good measure for the training error rate!!

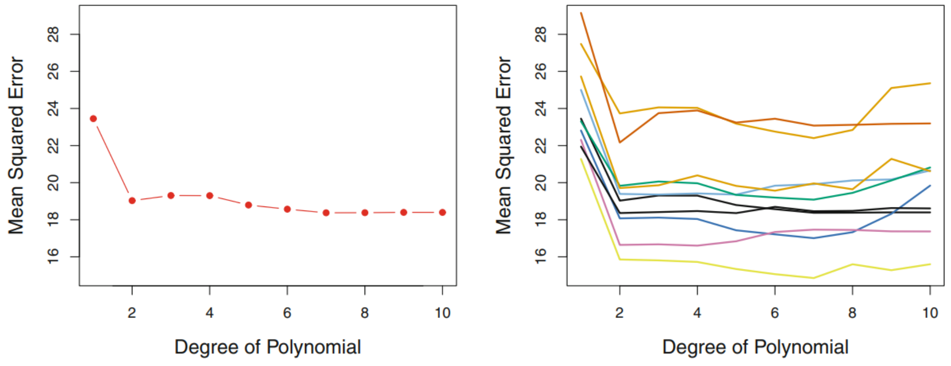
Holding out: consider a class of methods that estimate the test error rate by holding out a subset of the training observations from the fitting process and apply the learning method to those held out observations.

Validation set approach *validation set approach*: used to estimate test error rate associated with fitting a particular statistical learning method on a set of observations.

Strategy: - randomly dividing the available set of observations into two parts *training* and *validation/hold out set*. - model fit on the training set - fitted model used to predict the responses for the observations in the validation set - this results in *validation set error* (typically assessed using MSE) and provides an estimate of the test error rate

When looking at higher order fits, we can use the MSE as a validation set error and can therefore determine if a quadratic fit is better.

If we repeat the process and divide the sample set into two, we get a somewhat different estimate for the test of the MSE but we can still evaluate and compare different models based on MSE and variance of MSE between re sampled sets.



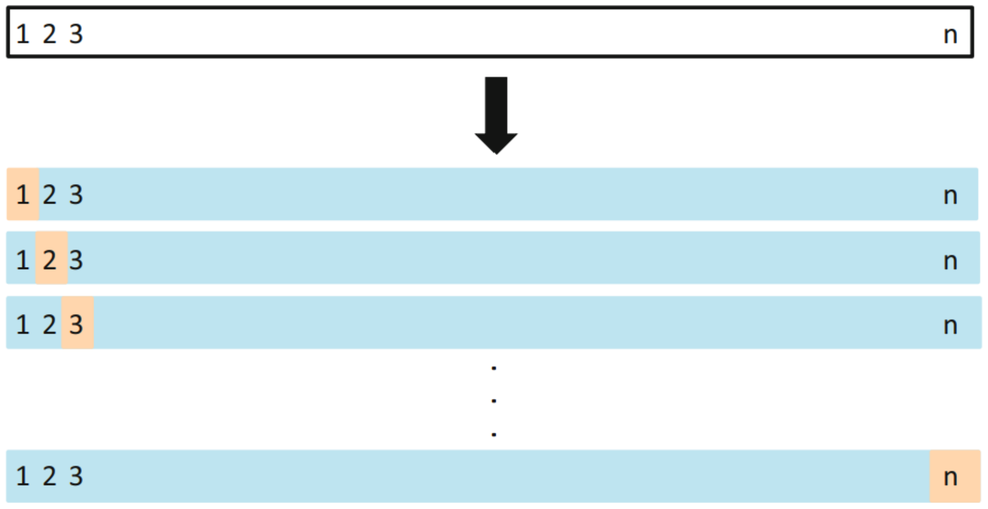
The validation set has two potential drawbacks: 1. seen on the right hand side of the image,validation estimate of the test error can be highly variable depending on split of the sample set 2. in the validation approach only a subset of the observations (in the training set) are used to fit the mode. This can suggest an *overestimate* of the test error rate.

Leave-one-out cross validation *leave one out* cross validation is closely related to the validation set approach but it - splits the set of observations into two parts - a single observation is used for the validation set and the remaining observation make up the training set

The learning method is fit on $n-1$ training observations and a prediction $\hat{y}_1$ is made for the excluded observation using its x_1 value. Since (x_1, y_1) is not fitted in the process, $MSE_1 = (y_1 - \hat{y}_1)^2$ provides an approx unbiased estimate for the test error. But even is MSE_1 is unbiased, it is a poor estimate because it is highly variable

We can repeat the process by now choosing (x_2, y_2) for the validation set and so on, such that the estimate for the test MSE is the average of these n test error estimates,

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i \tag{5.1}$$



Advantages: - we repeatedly fit the learning method using the training sets that contain n-1 observations almost as many as are in the entire data set - tends not to overestimate the test error rate as much as the validation set - LOOCV will always product the same results (no randomness)

Disadvantages - expensive to implement - time consuming

But if we use least squares linear of polynomial we can use a short cut:

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}(\frac{y_i - \hat{y}_i}{1-h_i})^2 \tag{5.2}$$

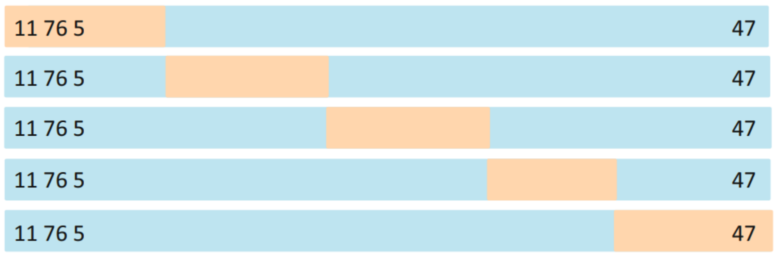
This is like the ordinary MSE but the *ith* residual is divided by the term $1 - h_i$. The leverage reflects the amount that an observation influences its own fit and high-leverage points are inflated in this formal to let the inequality hold.

k -fold cross-validation k-fold CV involves - randomly dividing the set of observations into k groups or *folds* of approx equal size. - first fold is treated as a validation set and the method is fit on the remaining $k-1$ folds. - MSE is then computed on the observations in the hold-out fold - this is repeated k times each time a different group of observations is treated as the validation set

the k-fold CV estimate is computed by the average,

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i \tag{5.3}$$

LOOCV is a special case of k-fold CV in which k is set to n. The obvious advantage of not having n folds is computational efficiency. Performing LOOCV may pose computational problems if n is large.



The image above visualizes a 5-fold CV.

There is some variability with k-fold but is much lower than the validation set approach. We don't know th true test MSE so its difficult to determine the accuracy of the CV estimate. But we can examine the simulated data then compute the true test MSE and evaluate the accuracy of our CV results.

Sometimes we are interested in the location of the *minimum point* int eh estimated test MSE curve. This is because we might be performing CV on a number of statistical learning methods or on a single method using different levels of flexibility in order to identify the method that results in the lowest test error.

Bias-variance trade off for k fold CV k-fold usually gives more accurate estimates of the test error rate than does LOOCV. This has to do with bias-variance trade off.

The validation set approach can lead to overestimates of the test error rate since the fit is from only half the observations. LOOCV will give approx unbiased estimates of the test error because of the n-1 observations trained on. K-fold leads to intermediate bias because of the (k-1)n/k observations.

We also need to keep in mind the *procedure variance.* LOOCV has higher variance than k-folds with $k < n$ because we are essentially averaging the outputs of k fitted models that are somewhat less correlated with each other because the overall of each traing set is smaller compared to LOOCV, which averages n fitted models with almost identical sets of observations.

A mean with highly correlated quantities has a higher variance than does the mean of lesser correlated quantities, the test error from LOOCV tens to have higher variance than compared to k-fold.

Note: Correlation is proportional to the the covariance of two variables. The divisor in the equation acts has a scaling effect on the covariance so that the resulting correlation will lie between -1 and +1. So, all other things being equal, reducing the covariance will reduce the correlation.

To summarize, there is a bias-variance trade-off associated with the choice of k in k-fold cross validation. Typically, k-fold with k= 5, 10, yield to test-error rates that don't suffer from high bias or high variance.

Cross-validation on classification problems We can use cross-validation when classifying outcome Y when qualitative. Cross validation works just as described except rather than use MSE to quantify the error, we use the number of misclassified observations. In the classification setting,

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

where $Err_i = I(y_i \neq \hat{y}_i)$.

For example, fitting various logistic regression models shows different decision boundaries. We can now compare the true test error rate which is 0.201 and the bayes error rate. Because our error rate is larger, we can see the logistic model does not have enough flexibility to model the bayes boundary, We can use polynomial functions to increase flexibility.

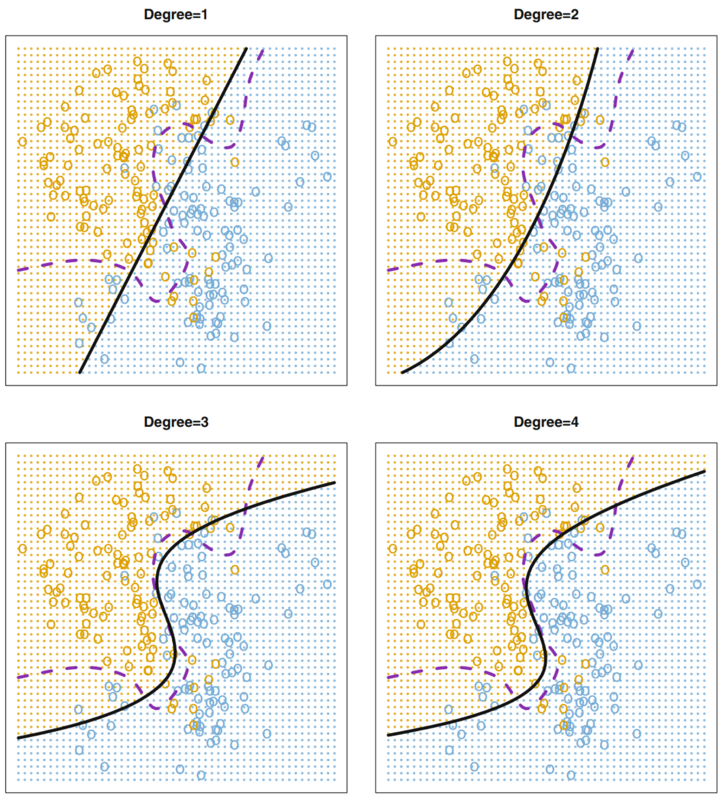


Figure 1: log regression sim

For example the quadratic logistic regression model is given by,

$$log(\frac{p}{1-p}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_2^2) \tag{5.5}$$

But in practice the bayes decisions boundary and test error rates are unknown, so using the 10 fold CV error rates resulted from fitting ten logistic regression models to the data can be used. The test errors are shown in brown and the training errors are shown in blue. The training error decreases as the flexibility of the fit increases. In contrast, the test error displays a characteristic U-shape.

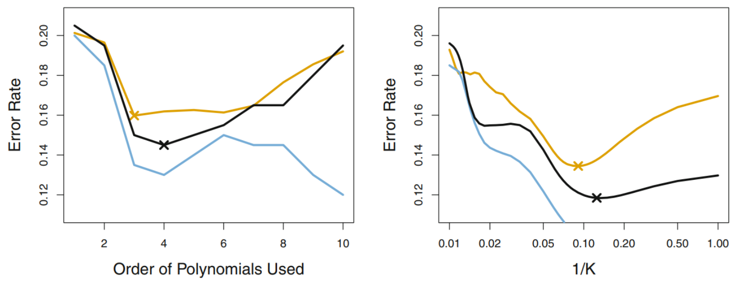


Figure 2: error rate

The bootstrap

The bootstrap is used to quantify the uncertainty associated with a given estimator or statistical learning method. It can be used to estimate the standard errors of the coefficients from a linear regression fit. It can easily be applied to a wide range of statistical methods.

Example: We invest a fixed sum of money into two financial assets that yield returns fo X and Y (random quantities). We will invest a fraction of α of our money in X and will invest the remaining quantity $1-\alpha$ in Y. Given variability, we wish to minimize α to minimize the total risk/variance of the investment.

We want to minimize $var(\alpha X + (1-\alpha)Y)$. One can show that the value to minimize risk is given by,

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}} = \frac{Var(Y) - Cov(X,Y)}{Var(x) + Var(Y) - 2Cov(X,Y)} \tag{5.6}$$

We can estimate α using

$$\hat{\alpha} = \frac{\hat{\sigma_Y^2} - \sigma_{\hat{XY}}}{\hat{\sigma_X^2} + \hat{\sigma_Y^2} - 2\sigma_{\hat{XY}}} \tag{5.7}$$

To quantify the accuracy of $\hat{\alpha}$ repeat simulation of 100 paired X and Y and estimate α 1000 times. We obtain 1000 estimates for α and average the values our such that,

$$\bar{\alpha} = \frac{1}{1000}\sum_{r=1}^{1000}\hat{\alpha}_r = 0.5996$$

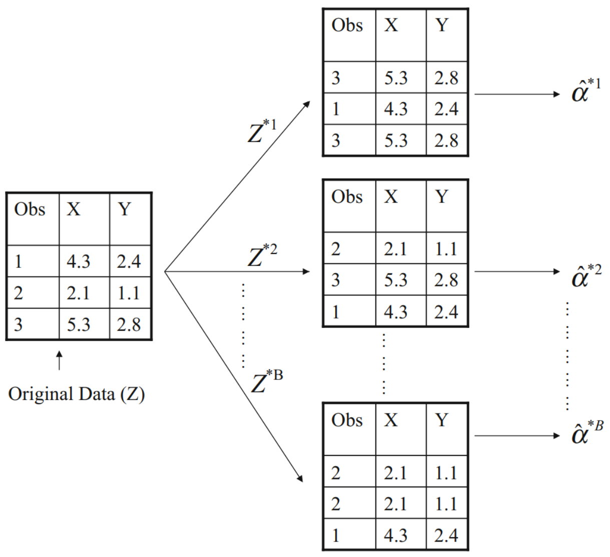
the standard deviation of the estimates is

$$\sqrt{\frac{1}{1000-1}\sum_{r=1}^{1000}(\hat{\alpha}_r - \bar{\alpha})^2} = 0.083$$

The accuracy of $\hat{\alpha}$ can be measured as $SE(\hat{\alpha}) \approx 0.083$. Roughly speaking, for a random sample from the population, we would expect a $\hat{\alpha}$ to different from α by approx 0.08 on average.

The bootstrap approach allows us to use a computer to emulate the process of obtaining a new sample set so we can estimate the variability of $\hat{\alpha}$ without generating additional samples. We can obtain distinct data sets by repeatedly sampling observations *from the original data set.*

Below visualizes the bootstrap method for $n = 3$. Each bootstrap set contains n observations sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α.



We randomly select n observations from the data set in order to produce the bootstrap data set Z^{*1}. The sampling is done with replacement and can be used to produce a new bootstrap estimate for α which is α^{*1}. This is repeated B times from some large value B to produce $Z^{*1}, \dots, Z^{*B}$ and corresponding estimates of α. We can compute the standard error of these bootstrap estimates using,

$$SE_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'})} \tag{5.8}$$

5.3 Lab: cross validation and the bootstrap

```
library(ISLR)
```

validation set approach

```
## Warning: package 'ISLR' was built under R version 4.2.3
```

```
# rand subset of 196 observations
set.seed(1)
train <- sample(392, 196)

# linear regression
lm.fit <- lm(mpg~horsepower, data = Auto, subset = train)

# estimate responses
attach(Auto)
mean((mpg-predict(lm.fit, Auto))[-train]^2)
```

```
## [1] 23.26601
```

```r
# polynomial regression
lm.fit2 <- lm(mpg~poly(horsepower, 2), data = Auto, subset = train)
mean((mpg-predict(lm.fit2, Auto))[-train]^2)
```

```
## [1] 18.71646
```

```r
lm.fit3 <- lm(mpg~poly(horsepower, 3), data = Auto, subset = train)
mean((mpg-predict(lm.fit3, Auto))[-train]^2)
```

```
## [1] 18.79401
```

```r
# choosing different training set
set.seed(2)
train <- sample(392, 196)

# new linear regression
lm.fit <- lm(mpg~horsepower, subset =)
mean((mpg-predict(lm.fit, Auto))[-train]^2)
```

```
## [1] 25.3808
```

```r
# new poly regression
lm.fit2 <- lm(mpg~poly(horsepower, 2), data = Auto, subset = train)
mean((mpg-predict(lm.fit2, Auto))[-train]^2)
```

```
## [1] 20.43036
```

```r
lm.fit3 <- lm(mpg~poly(horsepower, 3), data = Auto, subset = train)
mean((mpg - predict(lm.fit3, Auto))[-train]^2)
```

```
## [1] 20.38533
```

A model that predicts mpg using quadratic performs better compared to linear models.

LOOCV Use cv.glm() and glm() to automatically compute generalized LOOCV model. Note: If we use glm using famil= binomial, this is the same as lm() and a OLR.

```r
library(boot)
```

```
## Warning: package 'boot' was built under R version 4.2.3
```

```r
glm.fit <- glm(mpg~horsepower, data = Auto)
cv.err = cv.glm(Auto, glm.fit)
cv.err$delta
```

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

The two numbers in the delta vector contain cross validation results.

Use a for loop for different poly orders

```r
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
```

k-fold cross validation cv.glm() can be used for k-fold CV

```r
# k-fold (k = 10)
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] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061 19.14666
##  [9] 18.87013 20.95520
```

The computation time is shorter because of shortcut 5.2. Weh we perform k-fold CV the two numbers associated with delta differ sightly. The first is the *standard k-fold CV estimate* and the second is the *bias-corrected* version.

The bootstrap **Example: Estimating the accuracy of a statistic of interest**

using the boot() function we can perform bootstrap by repeatedly sampling observations from the data with replacement

```r
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)))
}

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

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

use sample() function to randomly select 100 observations from the range 1 to 100 with replacement. This is equivalent to constructing a new bootstrap data set recomputing $\hat{\alpha}$ based on new data.

```r
set.seed(1)
alpha.fn(Portfolio, sample(100, 100, replace = T))
```

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

Implement bootstrap analysis by performing this many times and cording corresponding alpha estimates and std dev. boot() function automates this.

```
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 -0.001695873  0.09366347
```

Example: Estimating the accuracy of a linear reg model The bootstrap method can be used to assess the variability of the coefficient estimates and predictions from a statistical learning method. WE use the bootstrap approach to assess the variability of β_1, β_0.

First create boot.fn() which returns intercept and slope estimates of observations.

```
boot.fn <- function(data, index) {
  return(coef(lm(mpg~horsepower, data = data, subset = index)))
}

boot.fn(Auto, 1:392)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

boot.fn() function can also be used in order to create bootstrap estimates for the intercept and slop terms by randomly sampling from among the observations with replacement.

```
# two random samples
set.seed(1)
boot.fn(Auto, sample(392, 392, replace = T))
```

```
## (Intercept)  horsepower
##  40.3404517  -0.1634868
```

```
boot.fn(Auto, sample(392, 392, replace = T))
```

```
## (Intercept)  horsepower
##  40.1186906  -0.1577063
```

```
# compute standard errors
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  0.0544513229 0.841289790
## t2* -0.1578447 -0.0006170901 0.007343073
```

This shows that the std dev $SE(\hat{\beta}_0) = 0.86$ and $SE(\hat{\beta}_1) = 0.0074$. We can find the std errors for the regression coefficients in a linear model through summary().

```
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
```

Compute the bootstrap standard error estimate and the standard linear regression estimates that result from fitting the quadratic model to the data.

```
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  3.511640e-02 2.0300222526
## t2* -0.466189630 -7.080834e-04 0.0324241984
## t3*  0.001230536  2.840324e-06 0.0001172164
```

Excersises

Applied: 5.

```r
attach(Default)

log.sim <- function(seed.num) {
set.seed(seed.num)
print(sprintf("Using seed: %d", seed.num))

# log regression
glm.fit <- glm(default~income + balance, data = Default, family = binomial)
summary(glm.fit)

# split into training and validation
train <- sample(nrow(Default), nrow(Default)/2)

# MLR using train
glm.fit2 <- glm(default~income + balance, data = Default[train, ], family = binomial)

# predict using posterior
glm.fit2.probs <- predict(glm.fit2, newdata = Default[-train, ], type = "response")

# classify
glm.fit2.pred <- rep("No", length(glm.fit2.probs))
glm.fit2.pred[glm.fit2.probs > 0.5] <- "Yes"

table(glm.fit2.pred, Default[-train, "default"])

# validation set error
error <- mean(glm.fit2.pred != Default[-train, "default"])
print(sprintf("validation set error: %f", error))
}

log.sim(1)
```

```
## [1] "Using seed: 1"
## [1] "validation set error: 0.025400"
```

```r
log.sim(2)
```

```
## [1] "Using seed: 2"
## [1] "validation set error: 0.023800"
```

```r
log.sim(3)
```

```
## [1] "Using seed: 3"
## [1] "validation set error: 0.026400"
```

```r
set.seed(1)
# split into training and validation
train <- sample(nrow(Default), nrow(Default)/2)

# with student
glm.fit.student <- glm(default~income + balance + student, data = Default, family = binomial)
glm.fit.student.probs <- predict(glm.fit.student, newdata = Default[-train, ], type = "response")

glm.fit.student.pred <- rep("No", length(glm.fit.student.probs))
glm.fit.student.pred[glm.fit.student.probs > 0.5] <- "Yes"

table(glm.fit.student.pred, Default[-train, "default"])
```

```
##
## glm.fit.student.pred   No  Yes
##                  No  4826  112
##                  Yes   17   45
```

```r
error <- mean(glm.fit.student.pred != Default[-train, "default"])
print(sprintf("validation set error with student: %f", error))
```

```
## [1] "validation set error with student: 0.025800"
```

```r
# wo student
glm.fit <- glm(default~income + balance, data = Default, family = binomial)

glm.fit.probs <- predict(glm.fit, newdata = Default[-train, ], type = "response")

glm.fit.pred <- rep("No", length(glm.fit.probs))
glm.fit.pred[glm.fit.probs > 0.5] <- "Yes"

table(glm.fit.pred, Default[-train, "default"])
```

```
##
## glm.fit.pred   No  Yes
##          No  4826  107
##          Yes   17   50
```

```r
error <- mean(glm.fit.pred != Default[-train, "default"])
print(sprintf("validation set error wo student: %f", error))
```

```
## [1] "validation set error wo student: 0.024800"
```

7.

```r
# a
attach(Weekly)
glm.fit <- glm(Direction~ Lag1 + Lag2, data = Weekly, family = binomial)

# b
glm.fit2 <- glm(Direction~ Lag1 + Lag2, data = Weekly[-1, ], family = binomial)

# c
first_observation_probs <- predict(glm.fit, newdata = Weekly[1, , drop = FALSE], type = "response")

first_observation_pred <- ifelse(first_observation_probs > 0.5, "Up", "Down")
```

```r
# classification NOT correct
first_observation_pred
```

```
##    1
## "Up"
```

```r
Weekly[1, ]$Direction
```

```
## [1] Down
## Levels: Down Up
```

```r
# d/e
loocv.sim <- function(n) {
  correctly_classified <- rep(FALSE, n)
  for (i in 1:n) {
    glm.fit <- glm(Direction ~ Lag1 + Lag2, data = Weekly[-i, ], family = binomial)

    first_observation_probs <- predict(glm.fit, newdata = Weekly[i, , drop = FALSE], type = "response")

    first_observation_pred <- ifelse(first_observation_probs > 0.5, "Up", "Down")

    actual_direction <- Weekly$Direction[i]

    correctly_classified[i] <- first_observation_pred == actual_direction
  }
accuracy <- mean(correctly_classified)

  # Calculate the LOOCV test error
  loocv_error <- 1 - accuracy

  # Print the results
  print(sprintf("LOOCV Accuracy: %f", accuracy))
  print(sprintf("LOOCV Test Error: %f", loocv_error))
}

loocv.sim(4)
```

```
## [1] "LOOCV Accuracy: 0.250000"
## [1] "LOOCV Test Error: 0.750000"
```