Lab 3: Resampling methods - Solutions

```r
library(boot)
library(ISLR)
library(readr)
rm(list=ls())
options(digits = 3) # set default number of decimals
```

Exercise 1

(1)

Store data and your code in the same folder. Set working directory as source file location and load your data.

```r
set.seed(1)  # set seed for reproducibility
# read data and eliminate missing values
data <- na.omit(read.csv("homeloan.csv", na = ""))
summary(data) # summarize data
```

```
##      Loan_ID       Gender     Married   Dependents        Education
##  LP001003:  1   Female: 86   No :169   0 :274     Graduate    :383
##  LP001005:  1   Male  :394   Yes:311   1 : 80     Not Graduate: 97
##  LP001006:  1                          2 : 85
##  LP001008:  1                          3+: 41
##  LP001011:  1
##  LP001013:  1
##  (Other) :474
##  Self_Employed ApplicantIncome CoapplicantIncome   LoanAmount  Loan_Amount_Term
##  No :414       Min.   :  150   Min.   :    0     Min.   :  9   Min.   : 36
##  Yes: 66       1st Qu.: 2899   1st Qu.:    0     1st Qu.:100   1st Qu.:360
##                Median : 3859   Median : 1084     Median :128   Median :360
##                Mean   : 5364   Mean   : 1581     Mean   :145   Mean   :342
##                3rd Qu.: 5852   3rd Qu.: 2253     3rd Qu.:170   3rd Qu.:360
##                Max.   :81000   Max.   :33837     Max.   :600   Max.   :480
##
##  Credit_History    Property_Area Loan_Status
##  Min.   :0.000   Rural    :139   N:148
##  1st Qu.:1.000   Semiurban:191   Y:332
##  Median :1.000   Urban    :150
##  Mean   :0.854
##  3rd Qu.:1.000
##  Max.   :1.000
##
```

```r
# Convert numeric variables to factors
data$Property_Area <- factor(data$Property_Area)
data$Education <- factor(data$Education)
data$Married <- factor(data$Married)
data$Gender <- factor(data$Gender)
data$Dependents <- factor(data$Dependents)
data$Self_Employed <- factor(data$Self_Employed)
data$Loan_Status <- data$Loan_Status =="Y"
```

(2)

We run a logistic regression and summarize its output.

```r
# You can try different specifications by changing regressors
glm.fit = glm(Loan_Status ~ LoanAmount + Self_Employed + Education + Married + Gender +
        ApplicantIncome + Credit_History+ Property_Area, data = data, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Loan_Status ~ LoanAmount + Self_Employed + Education +
##     Married + Gender + ApplicantIncome + Credit_History + Property_Area,
##     family = binomial, data = data)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.284  -0.419   0.508   0.700   2.383
##
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -2.75e+00   5.68e-01   -4.85  1.2e-06 ***
## LoanAmount             -3.33e-03   1.72e-03   -1.93   0.0536 .
## Self_EmployedYes       -1.51e-01   3.48e-01   -0.43   0.6637
## EducationNot Graduate  -3.78e-01   2.98e-01   -1.27   0.2053
## MarriedYes              6.01e-01   2.68e-01    2.24   0.0249 *
## GenderMale              3.33e-01   3.23e-01    1.03   0.3025
## ApplicantIncome         1.51e-05   2.75e-05    0.55   0.5837
## Credit_History          3.61e+00   4.28e-01    8.44  < 2e-16 ***
## Property_AreaSemiurban  9.43e-01   3.01e-01    3.14   0.0017 **
## Property_AreaUrban      9.27e-02   2.93e-01    0.32   0.7515
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 593.05  on 479  degrees of freedom
## Residual deviance: 439.75  on 470  degrees of freedom
## AIC: 459.8
##
## Number of Fisher Scoring iterations: 4
```

(3)

We will count the number of mispredicted outcomes. If predicted probability of acceptance is above 0.5, we will assume that the loan is given. We make 91 mistakes in the training dataset and error rate is 19 percent.

```
trueresult <- data$Loan_Status ==TRUE
predictions <- predict.glm(glm.fit, data, type = "response") > 0.5
errors <- predictions != trueresult
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Loan_Status ~ LoanAmount + Self_Employed + Education +
##     Married + Gender + ApplicantIncome + Credit_History + Property_Area,
##     family = binomial, data = data)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -2.284  -0.419   0.508   0.700   2.383
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -2.75e+00   5.68e-01   -4.85  1.2e-06 ***
## LoanAmount              -3.33e-03   1.72e-03   -1.93   0.0536 .
## Self_EmployedYes        -1.51e-01   3.48e-01   -0.43   0.6637
## EducationNot Graduate   -3.78e-01   2.98e-01   -1.27   0.2053
## MarriedYes               6.01e-01   2.68e-01    2.24   0.0249 *
## GenderMale               3.33e-01   3.23e-01    1.03   0.3025
## ApplicantIncome          1.51e-05   2.75e-05    0.55   0.5837
## Credit_History           3.61e+00   4.28e-01    8.44  < 2e-16 ***
## Property_AreaSemiurban   9.43e-01   3.01e-01    3.14   0.0017 **
## Property_AreaUrban       9.27e-02   2.93e-01    0.32   0.7515
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 593.05  on 479  degrees of freedom
## Residual deviance: 439.75  on 470  degrees of freedom
## AIC: 459.8
##
## Number of Fisher Scoring iterations: 4
```

```
sum(errors)
```

```
## [1] 91
```

```
mean(errors)
```

```
## [1] 0.19
```

(4)

In each iteration, we leave the indexed observation out and estimate the model. Then we predict the outcome of the observation that we left outside the sample. We make 95 mistakes and test error rate is approximately 20 percent.

```
LOOCV = rep(0, dim(data)[1])
for (i in 1:(dim(data)[1])) {
    glm.fit = glm(Loan_Status ~ LoanAmount + Self_Employed + Education + Married + Gender +
    Credit_History + ApplicantIncome + Property_Area, data = data[-i,], family = binomial)
    # data = data[-i,] takes data without ith observation
    is_yes = predict.glm(glm.fit, data[i, ], type = "response") > 0.5
    # data[i, ] predicts ith observation's outcome
    if (is_yes != trueresult[i])
        LOOCV[i] = 1 # if it is mispredicted, puts 1
}
sum(LOOCV)
```

```
## [1] 95
```

```
mean(LOOCV)
```

```
## [1] 0.198
```

(5)

As theory suggests, test error rate is higher compared to error rate we have when we use the full dataset.

```
mean(errors)
```

```
## [1] 0.19
```

```
mean(LOOCV)
```

```
## [1] 0.198
```

(6)

We calculate standard errors of coefficients using bootstrap method. To do this, we draw random samples with replacement from our dataset and repeat estimation multiple times. As a result we get a distribution of coefficient estimates and we calculate standard deviation of this distribution in order to obtain standard errors.

```
N <-as.integer(nrow(data))
nrsim <- 1e3
coefrecord = array(0, c(nrsim,10))
for (i in 1:nrsim) {
    glm.fit = glm(Loan_Status ~ LoanAmount + Self_Employed + Education + Married + Gender +
    ApplicantIncome + Credit_History+ Property_Area, data = data[sample(1:N,N,replace = TRUE),],
          family = binomial)
```

```r
    # data = data[sample(1:N,N,replace = TRUE) draws a new sample with replacement
    coefrecord[i,] = glm.fit$coefficients
}
```

(7)

As you can see, two standard errors are very close. So, if we know that estimates have a limiting distribution with finite variance, we can use bootstrapping method to calculate standard errors. This will be useful especially if we contruct our own estimator,e.g. simulated maximum likelihood estimators without closed form likelihood function.

```r
glm.fit = glm(Loan_Status ~ LoanAmount + Self_Employed + Education + Married + Gender +
        ApplicantIncome + Credit_History+ Property_Area, data = data, family = binomial)
# apply(coefrecord,2,sd)
# summary(glm.fit)$coefficients[,2]
cbind(apply(coefrecord,2,sd),summary(glm.fit)$coefficients[,2])
```

```
##                             [,1]     [,2]
## (Intercept)             8.03e-01 5.68e-01
## LoanAmount              1.96e-03 1.72e-03
## Self_EmployedYes        3.73e-01 3.48e-01
## EducationNot Graduate   3.01e-01 2.98e-01
## MarriedYes              2.77e-01 2.68e-01
## GenderMale              3.48e-01 3.23e-01
## ApplicantIncome         3.95e-05 2.75e-05
## Credit_History          7.11e-01 4.28e-01
## Property_AreaSemiurban  3.18e-01 3.01e-01
## Property_AreaUrban      2.94e-01 2.93e-01
```

Note: Heads-up for CV with logistic regression

Instead of using a loop, we can use cv.glm() to do the LOOCV. But if you run the following codes:

```r
glm.fit = glm(Loan_Status ~ LoanAmount + Self_Employed + Education + Married + Gender +
        ApplicantIncome + Credit_History+ Property_Area, data = data, family = binomial)
cv.err2=cv.glm(data,glm.fit)
cv.err2$delta
```

```
## [1] 0.153 0.153
```

The test error is much smaller than in (5).

But if you slightly change the codes into the following:

```r
glm.fit = glm(Loan_Status ~ LoanAmount + Self_Employed + Education + Married + Gender +
        ApplicantIncome + Credit_History+ Property_Area, data = data, family = binomial)
cost <- function(real, predicted = 0) mean(abs(real-predicted) > 0.5)
cv.err1=cv.glm(data,glm.fit,cost)
cv.err1$delta
```

```
## [1] 0.198 0.198
```

It becomes equivalent to (5).

Why is that?

The key difference is the "cost function".

The cost function specifies how to compute the test error.

By default, cv.glm() uses the mean squared error as cost function. It is calculated as: mean((actual response variable y - predicted probability)^2):

```
mean((glm.fit$y-glm.fit$fitted.values)^2)
```

```
## [1] 0.146
```

Because our resonpse varibale y is binary, using mean squared error is not proper. Instead, we need to use test error rate: number of wrong predictions/number of total predictions.

We specify the cost function in "cost <- function(real, predicted = 0) mean(abs(real-predicted) > 0.5)": Setting 0.5 as the threshold and compute the share of wrong predictions. What this function does is similar to the following:

```
mean(abs(glm.fit$y-glm.fit$fitted.values)>0.5)
```

```
## [1] 0.19
```

To summarize:

You need to specify the cost function as the test error rate when you do cross validation using cv.glm() with binary response variable and logistic regression.