Multiple Linear Regression

Rob Leonard (robleonard@tamu.edu (mailto:robleonard@tamu.edu))

We should have our saved R object ‘2019flights.Rdata’. Either change the pathname in the load statement or copy the file to the directory this Rmd file is in.

```
load('2019flights.Rdata')
flightsNotCancelled = df %>%
  filter(CANCELLED == 0) %>%
  na.omit
```

In order to speed up some computations for the purposes of this project, let’s go ahead and subsample from the entire database of flights. Note that this is a typical path taken in industry, particularly when developing new products/methods. Lastly, it is a good practice to clean up large objects in R’s memory if you are not going to reference them again (or have them available on the hard drive to load in later).

```
set.seed(1)
nTotal = nrow(flightsNotCancelled)
flightsNotCancelled = flightsNotCancelled[sample(1:nTotal,round(nTotal/2)),]

# Let's clean up the name space to free up memory
rm(df)
```

Problem 1. The bootstrap

Suppose we want to estimate the difference in the mean ‘arrival delay’ of flights between the winter and the summer. **Let’s define October through March as the winter and April through September the summer.**

1.1. Using normality assumption

Here, we will apply the classic “pooled two sample t-test” for testing the population means between two groups.

1.1.1. Getting the difference in sample means

Get the sample means for these two groups, the sample variances for each group, and the sample size for each group

```
winter = flightsNotCancelled %>%
  filter(MONTH >= 10 | MONTH <= 3) %>%
  summarise(mean = mean(ARR_DELAY), var = var(ARR_DELAY), n = n())
summer = flightsNotCancelled %>%
  filter(MONTH >= 4 & MONTH <= 9) %>%
  summarise(mean = mean(ARR_DELAY), var = var(ARR_DELAY), n = n())
```

The sample mean difference between winter and summer is -2.8129296.

1.1.2. Getting the pooled variance

Getting the pooled sample variance and the quantile from the t-distribution, we can get the pooled two-sample t-test confidence interval

```
winterMinusSummer = winter$mean - summer$mean
n                 = winter$n + summer$n
pooledVar         = ((winter$n - 1)*winter$var + (summer$n - 1)*summer$var)/(n - 2)
tQuantile         = qt(1-0.05/2,n-2)

winterMinusSummerSE = sqrt(pooledVar)*sqrt(1/winter$n + 1/summer$n)
normalCI            = c(winterMinusSummer - tQuantile * winterMinusSummerSE,
                        winterMinusSummer + tQuantile * winterMinusSummerSE)
```

Provide an interpretation of this confidence interval.
The 95% confidence interval for a difference in mean arrival delay times between the winter and summer seasons is (-2.918055,-2.7078043. If we were to repeat the sampling and confidence interval construction process many times, we would expect that the true mean difference in arrival delay time would lie in the interval 95% of the time. Since 0 is not in the confidence interval, we reject a null hypothesis that there isn't any difference in arrival delay times. There is strong evidence that summer arrival delay times are higher than winter delay times.

1.2. Using bootstrap

Let's do 10 bootstrap draws and compare the confidence interval (we should do a lot more, but this is a HW afterall). We could use caret, but that allocates all the bootstrap draws at once, which takes a lot of memory for this problem. We can use "sample' instead. Note that with this many observations, we won't need to worry about stratified sampling over season.

```
nBootstrapDraws = 10

bootstrapResults = rep(0,nBootstrapDraws)
srt = proc.time()[3]
for(b in 1:nBootstrapDraws){
  flightsNotCancelled_boot = flightsNotCancelled[sample(1:n,n,replace=TRUE), ]
  winter_boot = flightsNotCancelled_boot %>%
  filter(MONTH >= 10 | MONTH <= 3) %>%
  summarise(mean = mean(ARR_DELAY))
summer_boot = flightsNotCancelled_boot %>%
  filter(MONTH >= 4 & MONTH <= 9) %>%
  summarise(mean = mean(ARR_DELAY)) # compute the two sample means

  bootstrapResults[b] = winter_boot$mean - summer_boot$mean #compute the sample mean difference
}
end = proc.time()[3]

bootstrapCI = c(quantile(bootstrapResults, probs=.025),
                quantile(bootstrapResults, probs=.975))

#again, let's clean up a bit to keep our memory fresh(er)
rm(flightsNotCancelled_boot)
```

A 95% bootstrap confidence interval would be (-2.8615287, -2.7261712). This took 47.13 seconds to compute.

Now, let’s parallelize this so that it will run faster

```
nCores = detectCores()
nCores
```

```
## [1] 24
```

This system has 24 cores. However, the data set takes up a fair amount of memory

```
format(object.size( flightsNotCancelled ),units='MB')
```

```
## [1] "805.7 Mb"
```

Note that we have to re-require(dplyr) inside the parallel call. This is due to the fact that a new R session is created at each core.

```
cl = makeCluster(nCores-4)
registerDoParallel(cl)

bootstrapResults = rep(0,nBootstrapDraws)

srt = proc.time()[3]
foreach (b=1:nBootstrapDraws) %dopar% {#### Answer 1.2.5 put the parallel for loop call here
  require(dplyr)
  flightsNotCancelled_boot = flightsNotCancelled[sample(1:n,n,replace=TRUE), ]
  winter = flightsNotCancelled_boot %>%
    filter(MONTH >= 10 | MONTH <= 3) %>%
    summarise(mean = mean(ARR_DELAY))
  summer = flightsNotCancelled_boot %>%
    filter(MONTH >= 4 & MONTH <= 9) %>%
    summarise(mean = mean(ARR_DELAY)) #### Answer 1.2.6

  bootstrapResults[b] = winter$mean - summer$mean
}
```

```
## [[1]]
## [1] -2.820979
##
## [[2]]
## [1] -2.878692
##
## [[3]]
## [1] -2.756755
##
## [[4]]
## [1] -2.794178
##
## [[5]]
## [1] -2.844209
##
## [[6]]
## [1] -2.761841
##
## [[7]]
## [1] -2.844284
##
## [[8]]
## [1] -2.812306
##
## [[9]]
## [1] -2.790093
##
## [[10]]
## [1] -2.855369
```

```
end = proc.time()[3]

stopCluster(cl)
registerDoSEQ()# This returns the session to "serial" instead of "parallel"
```

This took 130.05 seconds to compute.

Problem 2. Measuring performance in regression

Let’s process the data similarly to last time, keeping DEP_DELAY and ARR_DELAY

```
rm(list=ls())# Just to clean up the memory again

load('2019flights.Rdata')
flightsNotCancelled = df %>%
  filter(CANCELLED == 0) %>%
  select(ARR_DELAY,DEP_DELAY) %>%
  na.omit()

rm(df)
```

2.1 Training/Validation/Test Split

Let's get a training/validation/test split for evaluating our regression model's performance. Split the data into 50% train and 25% validation and 25% test. **Follow the code and object naming convention in caretPackage.rmd**

```
trainIndex       = createDataPartition(flightsNotCancelled$ARR_DELAY, p = .5 , list = FALSE) %>%
as.vector(.)
validSplit       = createDataPartition(flightsNotCancelled$ARR_DELAY[-trainIndex], p = .5 , list
= FALSE) %>% as.vector(.)
n                = nrow(flightsNotCancelled)
testIndex        = (1:n)[-trainIndex][-validSplit]
validIndex       = (1:n)[-trainIndex][validSplit]
role             = rep('train',n)
role[testIndex]  = 'test'
role[validIndex] = 'validation'
```

2.2. Do early departures matter?

We want to start to think about choosing features in a principled manner. Let's compare using the raw DEP_DELAY to DEP_DELAY with the negative numbers truncated at zero, we will call DEP_DELAY_TRUNC. First, we created a new variable in the exsiting dataframe.

```
flightsNotCancelled = flightsNotCancelled %>%
  mutate(DEP_DELAY_TRUNC = DEP_DELAY*(ifelse(DEP_DELAY<0,0,1)))
```

The fundamental idea behind data splitting is to not use the same data for multiple purposes, which could lead to use getting overly optimistic results.

Hence, we want to do the following: * compute the both models using the training data * choose which model you prefer using the validation data by comparing the squared error * then use the test data to get a good estimate of the test error.

Training data

```
Xtrain      = data.frame(X = flightsNotCancelled$DEP_DELAY[role == 'train'])
XtruncTrain = data.frame(X = flightsNotCancelled$DEP_DELAY_TRUNC[role == 'train'])
Ytrain      = flightsNotCancelled$ARR_DELAY[role == 'train']

lmOut      = lm(Ytrain ~ ., data = Xtrain)
lmTruncOut = lm(Ytrain ~ ., data = XtruncTrain)
```

Validation data

```
Xvalid      = data.frame(X = flightsNotCancelled$DEP_DELAY[role == 'validation'])
XtruncValid = data.frame(X = flightsNotCancelled$DEP_DELAY_TRUNC[role == 'validation'])
Yvalid      = flightsNotCancelled$ARR_DELAY[role == 'validation']

Yhat       = predict.lm(lmOut, Xvalid)
YtruncHat  = predict.lm(lmOut, XtruncValid)

validSqError = list('original' = sum( (Yhat - Yvalid)**2 ),
                    'truncated' = sum( (YtruncHat - Yvalid)**2 ))
```

Is the original or truncated feature better?

The original feature is better as we get less squared error.

We just used the validation data for an important purpose. **Explain what that purpose is and why we should appeal to test data now for getting a fair estimate of the test error**
The validation data was used to choose which model (using original or truncated delay data) to use to compute our test error/ apply to our test data. This will result in a better estimate of the true error as we aren't using the same data subset to both choose which model to use and then to estimate the squared error.

2.2.1 Test data

Now, we can get a fair estimate of the test error

```
Xtest      = data.frame(X = flightsNotCancelled$DEP_DELAY[role == 'test'])
Ytest      = flightsNotCancelled$ARR_DELAY[role == 'test']

Yhat       = predict.lm(lmOut, Xtest)

testSqError = sum( (Yhat - Ytest)**2)
```

What is the estimate of the test squared error using the test data?

It is 3.3793796^{8}.

Problem 3.

We will be computing the coefficient of determination (i.e. R squared) for both these features and using the training/validation/test sets for different purposes.

Analogously to the previous problem, compute the both models using the training data, choose which model you prefer using the validation data, and then use the test data to get a good estimate of the true R squared. **Train data**

```
YhatTrain          = predict.lm(lmOut, Xtrain)
YhattruncTrain     = predict(lmTruncOut,XtruncTrain)
trainSqError       = sum( (YhatTrain - Ytrain)**2)
traintruncSqError = sum( (YhattruncTrain - Ytrain)**2)
trainTotalSS       = sum((mean(Ytrain)-Ytrain)**2)
trainRsq           = 1 - (trainSqError/trainTotalSS)
traintruncRsq      = 1 - (traintruncSqError/trainTotalSS)
```

R^2 for the original training data is 0.9281736. R^2 for the truncated training data is 0.9239121.

Validation data

```
YhatValid          = predict.lm(lmOut, Xvalid)
YhattruncValid     = predict(lmTruncOut,XtruncValid)
validSqError       = sum( (YhatValid - Yvalid)**2)
validtruncSqError = sum( (YhattruncValid - Yvalid)**2)
validTotalSS       = sum((mean(Yvalid)-Yvalid)**2)
validRsq           = 1 - (validSqError/validTotalSS)
validtruncRsq      = 1 - (validtruncSqError/validTotalSS)
```

R^2 for the original validation data is 0.9279712.
R^2 for the truncated validation data is 0.923657.
The original valid data results in a slightly higher R^2. So for the test data, we will use the original data.

Test Data

```
YhatTest          = predict.lm(lmOut, Xtest)
testSqError       = sum( (YhatTest - Ytest)**2)
testTotalSS       = sum((mean(Ytest)-Ytest)**2)
testRsq           = 1 - (testSqError/testTotalSS)
```

The true R^2 estimate using the original (non truncated) test data is 0.9290469.