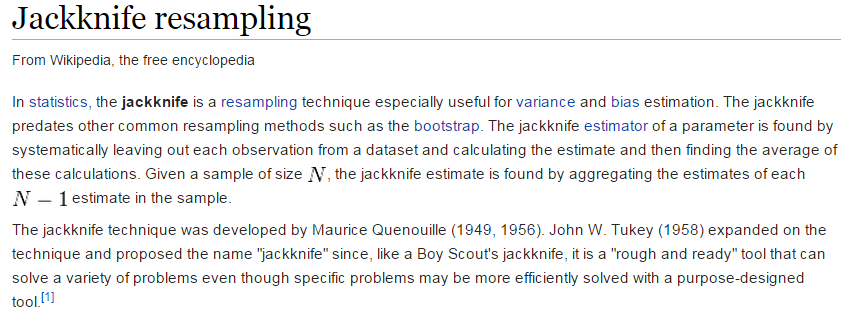
R: Jackknife and Cross-Validation in R

Section 1: The “Leave-One-Out” Concept for Simple Mean

The “leave-one-out” notion involves understanding the effect of a single observation on your outcomes. The “leave-one-out” approach could be used to identify observations with large leverage by investigating an observations effect on the outcome of interest.

It should be noted that the “leave-one-out” notion extends to a variety of problems. This notion is more commonly known as **jackknife** resampling. A snip-it of the wiki entry for jackknife resampling is provided here. The natural extension of jackknifing resampling would be “leave-several-out.” This is known as **cross-validation** when the goal is understanding the predictive ability of a model.

Source: <http://en.wikipedia.org/wiki/Jackknife_resampling>.



“Leave-one-out” for a Mean

To begin, type the following into R. This creates a vector whose elements are (2,3,5,8,10).

> y=c(2,3,5,8,10)

Recall, the mean() function is used to obtain the mean of y.

> mean(y)

[1] 5.6

The minus argument can be used to *temporarily* withhold an observation from calculations done on an object. For example, the following will calculate the mean of y without the 1st observation. The outcome here would be the same as calculating the mean using elements 2 through 5 which would be accomplished in R as mean( y[2:5] ).

> mean(y[-1])

[1] 6.5

The following shows the effect of the “leave-one-out” method applied to y for a mean calculation.

> mean(y[-2])

[1] 6.25

> mean(y[-3])

[1] 5.75

> mean(y[-4])

[1] 5

> mean(y[-5])

[1] 4.5

The mean of the complete y vector was 5.6. Leaving out the 1st observation appears to have the largest impact on the average. The following can be used to store the mean from each “leave-one-out” iteration.

Step #1: Create an initial vector to store the “leave-one-out” mean from each iteration.

> output = rep(0,5)

>

> #Looking at output

> output

[1] 0 0 0 0 0

Step #2: Use a for() loop to cycle through for each of the five observations

> for(i in 1:5){

+ output[i]=mean( y[-i] )

+ }

Step #3: Observing the output vector allows you to observe the effect of removing each

observation on the mean

> #Looking at output after for() loop

> output

[1] 6.50 6.25 5.75 5.00 4.50

Writing a function for “leave-one-out” for a mean

There are a variety of methods to create or build a function in R. The name of the function to be created is called mean.jackknife and I will use the edit() function to create my function. The edit() function produces a separate window.

> mean.jackknife=edit()

Some comments regarding functions in R.

* Arguments that need to be passed into a function are done so within the parentheses attached to function, i.e. function( ). The labeling of arguments within functions is separate from outside the function, e.g. in the following “x” is an argument only within my mean.jackknife() function.
* The code for functions must be contained with a set of curly brackets, i.e. {}.
* The return() will return a single object from a function. The list() function can be used when more than a single object is to be returned from the function.

|  |  |
| --- | --- |
| The finished mean.jackknife() function in R. | The following can be used to cut-and-paste this function in to R.  mean.jackknife = function(x){    #Find the length of x  n = length(x)    #Setup output vector  output = rep(0,n)    #Loop for iterations  for(i in 1:n){  output[i]=mean( x[-i])  }    #Return the output vector  return(output)  }  Note: If pasting this into the edit() window, delete mean.jackknife and the equal sign. |

After the function has been successfully created, you are able to use the function just as another function in R. The following produces the same outcomes as above.

> mean.jackknife(y)

[1] 6.50 6.25 5.75 5.00 4.50

The outcomes from this function can be put into a vector, say outcomes. Note: The outcomes vector does \*not\* need to be setup ahead of time in this situation.

> outcomes=mean.jackknife(y)

> outcomes

[1] 6.50 6.25 5.75 5.00 4.50

Creating a function to process the “leave-one-out” computations does take a bit more time; however, a function is more flexible in that this function will work with vector. Furthermore, assuming you save the workspace image upon exit, this function will be permanently saved into your workspace so that the “leave-one-out” function written here is available later.

Section 2: The “Leave-One-Out” Approach for Regression Coefficients

Again, for the sake of understanding, let’s create a simple response vector, Y, and a predictor variable, x.

and

Putting these into R can be done as follows.

> y=c(2,3,5,8,10)

> x=c(1,2,3,4,5)

Next, put each of these vectors into a data frame. A data frame will allow us to easily refer to particular elements of the data within various R functions.

> mydata=data.frame(y,x)

Fitting a simple linear regression model in R is done through the use of the lm() function. The data being used for this fit is contained in the data frame named mydata. The y and x used within the lm() function must be variable names in the mydata data frame.

* Mean Function:
* Variance Function:

> lm(y~x,data=mydata)

Call:

lm(formula = y ~ x, data = mydata)

Coefficients:

(Intercept) x

-0.7 2.1

The following can be used to save the regression output into an R object, say myfit.

> myfit=lm(y~x,data=mydata)

> myfit

Call:

lm(formula = y ~ x, data = mydata)

Coefficients:

(Intercept) x

-0.7 2.1

Next, let’s remove the 1st row from the mydata data.frame and refit the data. This is easily done in R using mydata[-1,]. The output here is being saved into an object called myfit.minus1. The regression coefficients from this model can easily be obtained from this object using myfit.minus1$coefficients as is shown here.

> myfit.minus1=lm(y~x,data=mydata[-1,])

> myfit.minus1$coefficients

(Intercept) x

-1.9 2.4

The object myfit.minus1 is actually a vector in R; thus, using the follow will return the second coefficient, i.e. the estimated slope or from the model.

> myfit.minus1$coefficients[2]

x

2.4

Understanding the effect of the removing additional observations on the estimated regression coefficients through the “leave-one-out” process follows.

> myfit.minus2=lm(y~x,data=mydata[-2,])

> myfit.minus2$coefficients[2]

x

2.028571

> myfit.minus3=lm(y~x,data=mydata[-3,])

> myfit.minus3$coefficients[2]

x

2.1

> myfit.minus4=lm(y~x,data=mydata[-4,])

> myfit.minus4$coefficients[2]

x

2.057143

> myfit.minus5=lm(y~x,data=mydata[-5,])

> myfit.minus5$coefficients[2]

x

2

We can again automate this process through the use of a for() loop.

Step #1: Create a vector to store the estimated slope from the model from each iteration.

> output = rep(0,5)

>

> #Looking at output

> output

[1] 0 0 0 0 0

Step #2: Using a for() loop to cycle through each observation

> for(i in 1:5){

+ fit = lm(y~x,data=mydata[-i,])

+ output[i]=fit$coefficients[2]

+ }

Step #3: Observe the effect of removing each observation on the estimated slope from the

simple linear regression model.

> #Looking at output after for() loop

> output

[1] 2.400000 2.028571 2.100000 2.057143 2.000000

Recall, that from the model with all the observations was 2.1. The following simple subtraction can be used to understand how much the estimated slope changes when the “leave-one-out” approach is used.

> 2.1-output

[1] -0.30000000 0.07142857 0.00000000 0.04285714 0.10000000

Comments:

* The “leave-one-out” approach to understand the effect on the estimated regression coefficients is called **DFBETAs**. For our model, a DFBETA exists for the estimated y-intercept and the estimated slope.
* DFBETAs are often standardized with respect to the standard error. The DFBETA for would be calculated as follows where the notation simply implies the observation has been removed.
* The following function was written in R to conduct the “leave-one-out” procedure for a simple linear regression model.

|  |  |
| --- | --- |
|  | betahat.jackknife=function(slr.object,data){    #Getting the number of rows in data  n = dim(data)[1]    #Creating the output data frame, column 1 for beta0hat and  # column 2 for beta1hat  output = data.frame(beta0hat=rep(0,n),beta1hat=rep(0,n))  #Looping through data, Beta0hat will be put into column 1  # and Beta1hat will be put into column 2  for(i in 1:n){  fit=lm(formula(slr.object),data=data[-i,])  output[i,1]=fit$coefficients[1]  output[i,2]=fit$coefficients[2]  }    #Return the output vector  return(output)    } |

Using this function in R

> fit=lm(y~x,data=mydata)

> betahat.jackknife(fit,mydata)

beta0hat beta1hat

1 -1.9000000 2.400000

2 -0.3428571 2.028571

3 -0.5500000 2.100000

4 -0.6571429 2.057143

5 -0.5000000 2.000000

Section 3: The “Leave-One-Out” Approach for Prediction

Consider again the response and predictor variable from the previous section.

|  |  |
| --- | --- |
| and |  |

First, let’s fit the standard simple linear regression model in R.

|  |  |
| --- | --- |
| The Data    Mean and Variance Functions | Simple Regression Output |

|  |  |  |
| --- | --- | --- |
|  | Theoretical Model Setup | Model Estimates |
| Mean |  |  |
| Standard Deviation (RMSE) |  |  |

The goal of using the “leave-one-out” approach in this section is **to understand the predictive ability** of a model. The preferred measure to understand predictive ability is Root Mean Square Error.

There are shortcomings in using Root Mean Square Error from a regression model to understand “average” distance to mean or “average” residual. The most significant are discussed here.

1. The purpose of the RMSE value is to provide the *best possible unbiased* estimate for the standard deviation in the response distribution after conditioning on x, .
2. Using the RMSE value from the model is likely to *underestimate* a models true ability to predict because observations for which predictions are being made were used to build the model.

|  |  |
| --- | --- |
| Issue #1: The purpose of RMSE is to attain the best possible estimate of . | Issue #2: Using residuals from observations used to build the model results in an underestimate of a models true predictive ability. |

|  |
| --- |
| Goal of “Leave-one-out” procedure for Prediction |
| Obtain a reasonable measure of RMSE which reflects the true predictive ability of a model. |

Putting the response and predictor vectors from above into R and creating a data frame.

> y=c(2,3,5,8,10)

> x=c(1,2,3,4,5)

> yx=data.frame(y,x)

Fitting the simple linear regression model in R and placing the output into an R object called fit. The summary() function displays much of the regression output.

> fit <- lm(y~x, data=yx)

> summary(fit)

Call:

lm(formula = y ~ x, data = yx)

Residuals:

1 2 3 4 5

0.6 -0.5 -0.6 0.3 0.2

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) -0.7000 0.6351 -1.102 0.35086

x 2.1000 0.1915 10.967 0.00162 \*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6055 on 3 degrees of freedom

Multiple R-squared: 0.9757, Adjusted R-squared: 0.9676

F-statistic: 120.3 on 1 and 3 DF, p-value: 0.001623

The estimated regression equation for this example is given by the following.

In R, getting the predicted values can be done directly using the predict() function. The predict function needs two arguments: i) the model object and ii) the new data for which predictions should be made.

Note: The column names for the predictors (in the newdata data frame) should match those in the data frame used to fit the model. In the following, predictions are being made on the original data – which is the yx data frame.

> predict(fit,newdata=yx)

1 2 3 4 5

1.4 3.5 5.6 7.7 9.8

You can simply save these predicted values into its own vector as follows.

> predictedy = predict(fit,newdata=yx)

Once the predicted values are obtained, residuals can easily be computed in R.

> y-predictedy

1 2 3 4 5

0.6 -0.5 -0.6 0.3 0.2

Finding the Room Mean Squared Error by way of brute force in R is shown next. It should be noted that the denominator is given by and not the number of residuals. Using provides an *unbiased* estimate of .

> sqrt(sum((y-predictedy)^2)/3)

[1] 0.6055301

|  |
| --- |
| **Mean Absolute Error**  *This is often used in predictive modeling instead of root mean squared error.*  > mean(abs(y-predictedy))  [1] 0.44 |

Obtaining “Leave-one-out” Predictions in R

The “leave-one-out” approach for predictions can be accomplished easily in R through the following sequence of commands.

Step 1: Fit the model withholding the 1st observation

> fit.minus1=lm(y~x,data=mydata[-1,])

Step 2: Make the prediction for the 1st observation

> predictedy1=predict(fit.minus1,newdata=mydata[1,])

Step 3: Obtain the squared residual for this prediction

> (y[1]-predictedy1)^2

1

2.25

The above steps need to be done for each of the five observations in our data frame. This can be done easily using a for() loop in R.

First, setup an output vector to store the results.

> output=rep(0,5)

> output

[1] 0 0 0 0 0

Using a for() loop to cycle through data frame one row at a time.

> for(i in 1:5){

+ fit.minus = lm(y~x,data=mydata[-i,])

+ predictedy = predict(fit.minus,newdata=mydata[i,])

+ output[i] = (y[i]-predictedy)^2

+ }

The desired output has been placed in the output vector.

> output

[1] 2.2500000 0.5102041 0.5625000 0.1836735 0.2500000

The average squared residual via the “leave-one-out” method is about 0.75. The estimated Root Means Squared Error via “leave-one-out” is about 0.87.

> mean(output)

[1] 0.7512755

> sqrt(mean(output))

[1] 0.8667615

The RMSE value for the “leave-one-out” approach is somewhat higher as expected, i.e. our ability to make a predictions is harder when an observation is not being used to build the model. The degree of difference, i.e. 43% increase, is somewhat exaggerated in this simple example due to having only 5 observations in the dataset.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | Original Model | “Leave-one-out”  Method | | RMSE | 0.61 | 0.87 | | % increase with “leave-one-out” |

The following predict.jackknife() function was written in R to automate the process developed above. The list() function is used to return output from this function as more than one object is being returned.

> predict.jackknife=function(lm.object,data){

#Getting the number of rows in data

n = dim(data)[1]

#Keeping a copy of orginial y (used in computed residual)

originaly = lm.object$model[,1]

#Creating the output vector to save squared residuals

output = rep(0,n)

#Looping through data

for(i in 1:n){

fit.minus=lm(formula(lm.object),data=data[-i,])

predictedy = predict(fit.minus,newdata=data[i,])

output[i]=(originaly[i]-predictedy)^2

}

#Return the output vector

list(SquaredResids=output,Jackknife\_RMSE=sqrt(mean(output)))

}

The following produces the same output as obtained above.

> predict.jackknife(fit,yx)

$SquaredResids

[1] 2.2500000 0.5102041 0.5625000 0.1836735 0.2500000

$Jackknife\_RMSE

[1] 0.8667615