Jackknife of the Mean

Wei Wang

Octobor 4, 2017

Project Goals

The goal of this project is to work through building functions around the Jackkinfe. We will start with some simple functions and then determine a pattern to build a more general function in the end.

Jackknife of the Mean!

- Have an estimator $\hat{\theta}$ of parameter θ
  want the standard error of our estimate, $se_{\hat{\theta}}$
- The jackknife approximation:
  - omit case i, get estimate $\hat{\theta}_{(-i)}$
  - Take the variance of all the $\hat{\theta}_{(-i)}$
  - multiply that variance by $\frac{(n-1)^2}{n}$ to get $\approx$ variance of $\hat{\theta}$
- then $se_{\hat{\theta}} =$ square root of that variance

PHP 2560 Only (Why $(n-1)^2/n$? Think about just getting the standard error of the mean)

Write a function called `mean.jackknife` that takes argument `a_vector` and returns a jackknife estimate of the standard error.

```
mean.jackknife <- function(a_vector) {
  a<-rep(0,length(a_vector))
  for (i in 1:length(a_vector)){
    a[i]<-mean(a_vector[-i])
  }
  jackknife.variance<-((length(a_vector) - 1)/length(a_vector)) * sum((a - mean(a))^2)
  jackknife.stderr<-sqrt(jackknife.variance)
  return(jackknife.stderr)
}
```

Jackknife for the mean Example: test your code here

```
some_normals <- rnorm(100,mean=7,sd=5)
mean(some_normals)
```

```
## [1] 5.88366
```

```
(formula_se_of_mean <- sd(some_normals)/sqrt(length(some_normals)))
```

```
## [1] 0.4800171
```

```
all.equal(formula_se_of_mean, mean.jackknife(some_normals))
```

```
## [1] TRUE
```

Jackknife for Gamma Parameters

The following function is a way to calculate the method of moments estimators for the gamma distribution:

```
gamma.est <- function(the_data) {
  m <- mean(the_data)
  v <- var(the_data)
  a <- m^2/v
  s <- v/m
  return(c(a=a,s=s))
}
```

Jackknife for Gamma Parameters Function

Write a function called `gamma.jackknife` that takes argument `a_vector` and returns jackknife standard error estimates on the gamma parameters.

```
gamma.jackknife <- function(a_vector) {
  n = length(a_vector)
  estimate_a=c(n)
  estimate_s=c(n)
  for(i in 1:n){
  estimate_a[i]=gamma.est(a_vector[-i])[1]
  estimate_s[i]=gamma.est(a_vector[-i])[2]
  }
  standard_error_a=sqrt((var(estimate_a))*((n-1)^2)/n)
  standard_error_s=sqrt((var(estimate_s))*((n-1)^2)/n)
  jackknife.stderrs=c(standard_error_a,standard_error_s)
  return(jackknife.stderrs)
}
```

Jackknife for Gamma Parameters Example

```
input <- rgamma(1000, shape=0.1, scale=10)
gamma.est(input)
```

```
##           a           s
##  0.08773682 10.57488657
```

```
gamma.jackknife(input)
```

```
## [1] 0.01624308 2.12750804
```

Jackknife for linear regression coefficients

Write a function called `jackknife.lm` that takes arguments `df`, `formula` and `p` and returns jackknife standard error estimates on the coefficients of a linear regression model.

```
jackknife.lm <- function(df,formula,p) {
 n=nrow(df)
 jackknife.ests = matrix(nrow=p, ncol=n)
 for (i in 1:n){
   new.coefficients = lm(as.formula(formula), data=df[-i,])$coefficients
   jackknife.ests[,i ]=new.coefficients
 }
 var = apply(jackknife.ests, 1, var)
 jackknife.var = (n-1)^2/n*var
 jackknife.stderr = sqrt(jackknife.var)
  return(jackknife.stderr)
}
```

Jackknife for linear regression coefficients Example

```
output <- 1.2 + 0.6*input +  rnorm(1000, 0, 2.1)
data <- data.frame(output, input)
my.lm <- lm(output~input, data = data)
coefficients(my.lm)
```

```
## (Intercept)       input
##   1.1636503   0.6319668
```

```
# "Official" standard errors
sqrt(diag(vcov(my.lm)))
```

```
## (Intercept)       input
##  0.06839190  0.02094478
```

```
jackknife.lm(df=data,formula="output~input",p=2)
```

```
## [1] 0.06868515 0.01803669
```

Refactoring the Jackknife

- Omitting one point or row is a common sub-task
- The general pattern:

```
figure out the size of the data
for each case
   omit that case
   repeat some estimation and get a vector of numbers
take variances across cases
scale up variances
take the square roots
```

- Refactor by extracting the common “omit one” operation
- Refactor by defining a general “jackknife” operation

The Common Operation

- *Problem*: Omit one particular data point from a larger structure
- *Difficulty*: Do we need a comma in the index or not?
- *Solution*: Works for vectors, lists, 1D and 2D arrays, matrices, data frames:

Goal:

- Make the function select the correct dimensions
  - length for a 1d object
  - number of rows for 2d
- Write a function `omit.case` that omits a point given the data and returns the data minus that point. Make sure it can handle higher dimensions.

```
omit.case <- function(the_data,omitted_point) {
# This should take the data and omit one point at a time and return the new data
  dim = dim(the_data)
  if (is.null(dim)||length(dim)==1){
    return(the_data[-omitted_point])
    }
  else{
    return(the_data[-omitted_point,])
    }
}
```

- Write a function `omit_and_est` that takes the data with an omitted point and returns whatever function your estimator does.

```
        omit_and_est <- function(omit) {
# This function should take the output of omit.case and use it as input for the estimator
        estimator(omit.case(the_data,omit))
}
```

```
jackknife <- function(estimator,the_data) {
  if(is.null(dim(the_data))){n=length(the_data)}
  else{n=nrow(the_data)}
  omit_and_est <- function(omit) {
    estimator(omit.case(the_data,omit))
    }
  jackknife.ests <- matrix(sapply(1:n, omit_and_est), ncol=n)
  var.of.reestimates <- apply(jackknife.ests,1,var)
  jackknife.var <- ((n-1)^2/n)* var.of.reestimates
  jackknife.stderr <- sqrt(jackknife.var)
  return(jackknife.stderr)
}
```

It works

```
jackknife(estimator = mean, the_data = some_normals)
```

```
## [1] 0.4800171
```

```
all.equal(jackknife(estimator = mean,the_data = some_normals),
          mean.jackknife(some_normals))
```

```
## [1] TRUE
```

```
all.equal(jackknife(estimator=gamma.est,the_data=data$input),
          gamma.jackknife(data$input))
```

```
## [1] TRUE
```

```
all.equal(jackknife(estimator=gamma.est,the_data=data$input),
          gamma.jackknife(data$input), check.names=FALSE)
```

```
## [1] TRUE
```

```
est.coefs <- function(the_data) {
  return(lm(output~input,data=the_data)$coefficients)
}
est.coefs(data)
```

```
## (Intercept)       input
##   1.1636503   0.6319668
```

```
all.equal(est.coefs(data), coefficients(my.lm))
```

```
## [1] TRUE
```

```
jackknife(estimator=est.coefs,the_data=data)
```

```
## [1] 0.06868515 0.01803669
```

```
all.equal(jackknife(estimator=est.coefs,the_data=data),
          jackknife.lm(df=data,formula="output~input",p=2))
```

```
## [1] TRUE
```

Further Refactoring of jackknife()

The code for `jackknife()` is still a bit clunky: - Ugly `if-else` for finding `n` - Bit at the end for scaling variances down to standard errors

- write a function that calculates the `n` needed for the above code:

```
      data_size <- function(the_data) {
        if (is.null(dim(the_data))) { n <- length(the_data) }
            else { n <- nrow(the_data) }
    return(n)
      }
```

- Write a function that calculate the variance of all the estimates and returns the standard error

```
scale_and_sqrt_vars <- function(jackknife.ests,n) {
    var.of.reestimates <- apply(jackknife.ests,1,var)
    jackknife.var <- ((n-1)^2/n)* var.of.reestimates
    jackknife.stderr <- sqrt(jackknife.var)
  return(jackknife.stderr)
}
```

Now invoke those functions

```
jackknife <- function(estimator,the_data) {
  n <- data_size(the_data)
  omit_and_est <- function(omit) {
    estimator(omit.case(the_data,omit))
  }
  jackknife.ests <- matrix(sapply(1:n, omit_and_est), ncol=n)
  return(scale_and_sqrt_vars(jackknife.ests,n))
}
```