Lecture 14: The Split-Apply-Combine Paradigm

Statistical Computing, 36-350

Monday November 9, 2015

Outline

- A quick reminder of what R can do
- How to make life easier with repeated tasks on large data sets

Refresher

We’ve used some tools for iterating over objects in R without for() loops:

- subset(): retrieve part of the data according to some condition
- apply(): takes a matrix and a margin, applies a function
- sapply() or lapply(): takes a list (or vector), applies a function
- c() or rbind() or cbind(): concatenate these objects in a known pattern

General strategy

Today we will learn a workflow that can be summarized in three general steps:

- **Split** whatever data object we have into meaningful chunks
- **Apply** the function of interest to this division
- **Combine** the results into a new object

Sounds simple? It is, but it’s powerful when combined with data structures (see Hadoop/MapReduce for how this makes you billions)

Why is this useful without the billions?

- This reinforces the pattern/function approach: what you want to do versus how you want to do it
- If the full data set is big, and we’ve already done the splitting, this makes it tractable on smaller machines

An important application: ragged data

Our previous functions work well with well-composed data: apply() on matrices, sapply() or lapply() on lists and vectors. What about **ragged data**—where the dimensions of each object aren’t necessarily the same?

For this: start with data frames, though we’ll be going beyond this eventually
A sociological application

Politics and labor action: does having a friendlier government make labor action more or less likely?

Data: political economy of strikes

Compiled by Bruce Western, Sociology Dept., Harvard.

- Data frame of 8 columns: country, year, days on strike per 1000 workers, unemployment, inflation, left-wing share of government, centralization of unions, union density
- 625 observations from 18 countries, 1951–1985
- Note that since $18 \times 35 = 630 > 625$, some years missing from some countries

`strikes = read.csv("http://www.stat.cmu.edu/~ryantibs/statcomp/lectures/strikes.csv")`

`head(strikes)`

```r
# country year strike.volume unemployment inflation left.parliament
# 1 Australia 1951 296 1.3 19.8 43.0
# 2 Australia 1952 397 2.2 17.2 43.0
# 3 Australia 1953 360 2.5 4.3 43.0
# 4 Australia 1954 3 1.7 0.7 47.0
```
Research question

“Does having a friendlier government make labor action more or less likely?” becomes

“Is there a relationship between a country’s ruling party alignment (left versus right) and the volume of strikes?”

Lots of ways to approach this problem: simplest is to split it by country.

Functions subset(), split(), tapply()

Take Italy:

```r
italy.strikes = subset(strikes, country="Italy")
```

Or, if you prefer,

```r
italy.strikes = strikes[strikes$country=="Italy",]
head(italy.strikes)
```
italy.fit = lm(strike.volume ~ left.parliament, data=italy.strikes)
plot(italy.strikes$left.parliament, italy.strikes$strike.volume,
     main="Italy strike volume versus left-wing alignment",
     xlab="Strike volume", ylab="Left-wing alignment")
abline(italy.fit, col=2)

One down, seventeen to go

Tedious and dangerous to do this repeatedly—typos abound. How can we do this in an easier way?

First: we need subsets for every country. split() does this nicely:

strikes.split = split(strikes, strikes$country)
class(strikes.split)

## [1] "list"

names(strikes.split)

## [1] "Australia"  "Austria"   "Belgium"   "Canada"   "Denmark"
## [6] "Finland"    "France"    "Germany"   "Ireland"  "Italy"
## [16] "Switzerland" "UK"        "USA"
Now, let’s generalize our function. We want the linear model coefficients:

```r
my.strike.lm = function (country.df) {
  return(lm(strike.volume ~ left.parliament, data=country.df)$coefficients)
}
my.strike.lm(subset(strikes, country=="Italy"))
```

```
## (Intercept) left.parliament
## -738.74531  40.29109
```

We could use a `for()` loop ...

```r
strike.coefs = NULL
my.countries = c("France", "Italy", "USA")
for (this.country in my.countries) {
  strike.coefs = cbind(strike.coefs, 
    my.strike.lm(subset(strikes, country==this.country)))
}
colnames(strike.coefs) = my.countries
strike.coefs
```

```
## France   Italy   USA
## (Intercept)  202.4261408 -738.74531  111.440651
## left.parliament -0.4255319  40.29109  5.918647
```

Easier if we’ve split!

```r
strike.coefs = lapply(strikes.split[1:3], my.strike.lm)
strike.coefs
```

```
## $Australia
## (Intercept) left.parliament
##    414.7712254   -0.8638052
##
## $Austria
## (Intercept) left.parliament
##   423.077279    -8.210886
##
## $Belgium
## (Intercept) left.parliament
##   -56.926780     8.447463
```

Combine step
do.call(cbind, strike.coefs)

## Australia Austria Belgium
## (Intercept) 414.7712254 423.077279 -56.926780
## left.parliament -0.8638052 -8.210886 8.447463

Or, in one step:

strike.coefs = sapply(strikes.split[1:3], my.strike.lm)
strike.coefs

## Australia Austria Belgium
## (Intercept) 414.7712254 423.077279 -56.926780
## left.parliament -0.8638052 -8.210886 8.447463

All together now

coefs = sapply(strikes.split, my.strike.lm)
coefs

## Australia Austria Belgium Canada Denmark
## (Intercept) 414.7712254 423.077279 -56.926780 -227.8218 -1399.35735
## left.parliament -0.8638052 -8.210886 8.447463 17.6766 34.34477
## Finland France Germany Ireland Italy
## (Intercept) 108.2245 202.4261408 95.657134 -94.78661 -738.74531
## left.parliament 12.8422 -0.4255319 -1.312305 55.46721 40.29109
## Japan Netherlands New.Zealand Norway Sweden
## (Intercept) 964.73750 -32.627678 721.3464 -458.22397 513.16704
## left.parliament -24.07595 1.694387 -10.0106 10.46523 -8.62072
## Switzerland UK USA
## (Intercept) -5.1988836 936.10154 111.440651
## left.parliament 0.3203399 -13.42792 5.918647

plot(coefs[2,], xaxt="n", xlab="", ylab="Regression coefficient", main="Countrywise labor ativity by left-wing score")
axis(side=1, at=seq(along=colnames(coefs)), labels=colnames(coefs), las=2, cex.axis=0.5)
abline(h=0, col="grey")
Add some more coefficients

```r
my.strike.lm.better = function(country.df) {
  return(lm(strike.volume ~ left.parliament + unemployment + inflation,
             data=country.df)$coefficients)
}
coefs2 = sapply(strikes.split, my.strike.lm.better)
coefs2[,1:4]

## (Intercept) 157.9191118 600.6777769 -243.4822938 167.07123
## left.parliament 0.5658674 -11.2441604 12.4516118 13.43864
## unemployment -1.1181489 -10.9216990 0.3578217 -48.17903
## inflation 30.4666061 -0.5923972 10.2673539 27.21807
```

```
plot(coefs[2,],xaxt="n",xlab="",ylab="Regression coefficient",
     main="Countrywise labor ativity by left-wing score"
axis(side=1,at=seq(along=colnames(coefs)),labels=colnames(coefs),
     las=2,cex.axis=0.5)
points(coefs2[2,], col="red")
abline(h=0,col="grey")
```
Summary

- Split-apply-combine is a commonly used strategy for dealing with repeated calculations over large data sets.
- It is helpful conceptually (forces you to think: i. what are the main chunks of data, ii. what are the main functions to apply, iii. how to put things back together in a sensible way).
- It is also helpful computationally for large data sets.
- Turns out there are a whole family of `apply()` like functions that will make your life even easier, and these are also extra helpful for large data sets (next time).