Big data in R

EPIC 2015

Big Data: the new 'The Future'

In which Forbes magazine finds common ground with Nancy Krieger (for the first time ever?), by arguing the need for theory-driven analysis



This future brings money (?)

• NIH recently (2012) created the BD2K initiative to advance understanding of disease through 'big data', whatever that means



The V's of 'Big Data'

- Volume
 - Tall data
 - Wide data
- Variety
 - Secondary data
- Velocity
 - Real-time data

What is Big? (for this lecture)

- When R doesn't work for you because you have too much data
 - i.e. High volume, maybe due to the variety of secondary sources
- What gets more difficult when data is big?
 - The data may not load into memory
 - Analyzing the data may take a long time
 - Visualizations get messy
 - Etc.,

How much data can R load?

• R sets a limit on the most memory it will allocate from the operating system

```
memory.limit()
```

```
?memory.limit
```

R and SAS with large datasets

- Under the hood:
 - R loads all data into memory (by default)
 - SAS allocates memory dynamically to keep data on disk (by default)
 - Result: by default, SAS handles very large datasets better

Changing the limit

- Can use memory.size() to change R's allocation limit. But...
 - Memory limits are dependent on your configuration
 - If you're running 32-bit R on any OS, it'll be 2 or 3Gb
 - If you're running 64-bit R on a 64-bit OS, the upper limit is effectively infinite, but...
 - ...you still shouldn't load huge datasets into memory
 - Virtual memory, swapping, etc.
- Under any circumstances, you cannot have more than (2^31)-1 = 2,147,483,647 rows or columns

What is a 2GB (or 3GB) memory limit?

- 2GB of memory used by R is not the same as 2GB on disk
 - Overhead for R to keep track of your data
 - Memory used for analysis, etc.
 - Probably not more than about 500MB on disk

Making this number meaningful

data(esoph)

object.size(esoph)

- Download BRFSS as XPT file and unzip to a local file
 - URL: http://www.cdc.gov/brfss/annual_data/2013/files/LLCP2013XPT.ZIP
- Now open it using sasxport.get (from Hmisc) library(Hmisc) brfss <- sasxport.get(<PATH TO XPT FILE>) dim(brfss) object.size(brfss)

Too much data?

 Challenge: use logistic regression to look at whether self-reported race/ethnicity predicts having a health care plan:

```
brfss$has_plan <- brfss$hlthpln1 == 1
summary(glm(has_plan ~ as.factor(x.race),
data=brfss, family=binomial))</pre>
```

• This is reasonably quick, at least for me

Too much data?

 Let's look at odds of having a health care plan by race using epitools:

library(epitools)

oddsratio(brfss\$has_plan, as.factor(brfss\$x.race))

• I see:

Error in fisher.test(xx) : FEXACT error 40. Out of workspace.

• But... changing the amount of available memory does not solve this

Suppose you have too much data...

- If your data is just too big, there are several things you can do:
 - Make the data smaller
 - Get a bigger computer
 - Access the data differently
 - Split up the dataset for analysis

Option I: Make the data smaller

- The best initial option is often to ensure you really need to deal with the problem
- Run your analysis on a slice of the data – you may get all you need in order to move forward
- Challenge: can you slice 500 random rows from brfss and try computing an odds ratio?

Try a slice first



Image from http://localfoods.about.com/od/fishseafood/ss/Slice Abalone_2.htm

```
rows_to_select <- sample(1:nrow(brfss), 500, replace=F)
brfss_sample <- brfss[rows_to_select,]
oddsratio(brfss_sample$has_plan, as.factor(brfss_sample$x.race))</pre>
```

Option Ia: start with smaller data

- If your data come from a database, you may be able issuing a SQL query directly from R to get just the subset of the data you want.
 - Look into the 'RODBC' or 'RMySQL' packages if this is appropriate for your scenario
 - (But I can't demo it without a DB to connect to)



SQL is the lingua franca of relational databases

Option 2: Get a bigger computer

- You may be lucky enough to have budget for a bigger PC
- More likely, get some temporary space:
 - Use one machine on the high-performance cluster
 - Rent some cloud
 computing time (if IRB allows)

Glamorous computing power, 1962 edition



Image from: http://blog.longnow.org/02010/08/05/mainframedark-age/

Option 3a: data table rather than data frame

• data.table package

Some optimizations to data frame, but slightly different syntax

brfss_dt <- data.table(brfss)
object.size(brfss_dt)
object.size(brfss)</pre>

Option 3b: Buffer the data set on disk

- SAS-style
 - ffdf object ff package
 - Works a lot like a standard date frame, only reading in data only on demand
 - Quirky with respect to column types proceed with caution...

Option 4: split it up

- E.g. with big database.
 - Ask for 200 MB of records at a time
 - Analyze each
 - Combine the results
 - (This is analogous to mapreduce/split-apply-combine, which we'll come back to)
- Can use computing clusters to parallelize analysis

Split turns out the be the name of a pretty-looking town in Croatia



http://www.split.info/

What if it's just painfully slow?

- Sometimes you can load the data, but analyzing it is slow
- Two possibilities:
 - Might be you don't have any memory left
 - Or that there's a lot of work to do because the data is big and you're doing it over and over

You don't have to embrace the sloth



http://whatthehellisrosedoingnow.blogspot.com/201 2/04/weird-extinct-giant-sloth.html

If you don't have any memory left

- Then you really have too much data, you just happen to be able to load it.
 - Depending on your analysis, you may be able to create a new dataset with just the subset you need, then remove the larger dataset from your memory space.

```
rows <- [1:500]
columns <- [1:30]
subset <- bigdata[rows, columns]
rm(bigdata)</pre>
```

Maybe he could have just removed the larger dataset



Image from: http://blog.lib.umn.edu/wlas0006/1001a/2011/10/a-forgettable-momento.html

If you're doing a lot of computation

- First, profile (i.e. time your code)
- No, seriously: profile first
- It's never what you think it is. Some examples of performance issues you might not anticipate:
 - Some modeling code defaults to bootstrapping confidence intervals that you don't care about with 1000 iterations per model
 - You accidentally wrote the code so that it does some slow operation for every column in your 5000 column dataset, then selects the column you want rather than does the operation only on the one you care about
 - You so something for every line of your huge data frame and then combine results using c() or rbind() rather than assigning to a pre-allocated vector or matrix

Profiling

- Simple profiling
 - Option I:

system.time(<call>)

- Option 2:

start_time <- proc.time()
<call>
proc.time() - start_time

Not this kind of profiling



Image from : www.therightperspective.org/2009/07/24/the-racialprofiling-lie/

• Note: with option 2, you need to send all three commands to R at once (i.e. highlight all three). Otherwise you're including the time it takes to send the command in your estimate of the time it takes to process it.

Inexplicable error in option 2

When testing this out, I sometimes saw:

Error: unexpected input in "proc.time() -"

On the second proc.time() call. When I executed the call again, I didn't see the error again. If you see this, you can do:

```
end_time <- proc.time()
end time - start time</pre>
```

to compute the elapsed time

Profiling challenge I

Load ggplot and the diamonds dataset. Model price as a function of color, cut, depth, and clarity. Use system.time to see how long the regression takes

library(ggplot)
data(diamonds)
system.time(lm(price~color+cut+depth+clarity, data=diamonds))

Note: User/System/Elapsed: when you use system.time, you mostly just care about elapsed.

Profiling challenge 2

- Now use cut() to create a dichotomous high_price variable that is true for any diamonds above the median price.
- Use logistic regression to model high_price as a function of color, cut, depth, and clarity. Use system.time to see how long the logistic regression takes.

• How does logistic regression compare to linear regression?

Another profiling example

Do some operation on every row using apply (which pre-allocates memory):

```
start_time <- proc.time()
apply(diamonds, 1, function(row) { row['color'] == 'E' })
proc.time() - start_time</pre>
```

Do the same operation but build the response vector through concatenation:

More advanced profiling options

- Rprof is a function in the utils library that creates an external file with deep profiling results
- This is probably more than you need
- Some examples of using Rprof provided by Phil Spector: <u>http://www.stat.berkeley.edu/~nolan/stat133/Fall05/lectures/profilingEx.htm</u>
 I



Not the music producing/starlet murdering Phil Spector, presumably

No, it's not a mistake, it's just slow

- Okay, so you either need to do it faster on one CPU, or use more than one CPU
- Using more than one CPU is called parallelizing
- Parallelization is very powerful, but it makes things less simple, which is bad.
 Simple = fewer problems



This doesn't strike me as simple, but it came up on an image search for simplify

Image from: http://www.bradleygauthier.com/blog/71technologies-and-resources-to-help-simplify-anentrepreneurs-life/

Faster without parallelism

- Try the compiler package
 - By default, R is an interpreted language, which means R code gets turned into machine code every time you run it.
 - Compiling means translating the code you write into something the machine interprets only once. There's an upfront cost, but it can be faster once compiled
 - Can use compile() on a single function or enableJIT(), which auto-compiles every function at first use

That wasn't enough! I still want my analysis to be done sooner

- It may be possible to get parallel results without explicitly doing parallel programming.
 - Plyr
 - aaply:like apply, but with an option to parallelize
 - Foreach
 - Lets you write for loops that can be parallelized (assuming only effect of code is the return value)
 - Multicore
 - If you have a machine with a multi-core processor, the multicore apply functions (e.g. mclapply) can farm some set of the data on which the apply is being called to the other core to speed up the process
 - Multicore also has functions for more detailed parallel programming, which can make things fast but forces you to confront concurrency issues

Honey Badger don't give a %*^\$% what makes it faster



Image from: http://knowyourmeme.com/memes /honey-badger

Dang, it's still not enough: I have to go parallel for reals

- Parallel processing is basically farming out subtasks to independent processors, then merging results
- Concurrency issues:
 - How do you know when subsections of a task are independent?
 - How do you know when you are done?
 - New classes of potential mistakes: Race conditions, mutual exclusion, and deadlocks



Image from http://support.sas.com/rnd/scalability/connect/piping.html

True parallel programming: pdbR

• Programming with Big Data in R project

- <u>www.r-pdb.org</u>

- Packages designed to help use R for analysis of really really big data on high-performance computing clusters
- Beyond the scope of this class, and probably of nearly all epidemiology

How much can be parallelized?



Image from: http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Quick introduction to MapReduce

- The technology behind 60% of what Google does (and about a million other web applications do)
- Two phase algorithm:
 - Map (farm out parallelizable task to many machines)
 - Reduce (combine results)
- It used to be that really understanding map/reduce model & underlying technologies (e.g. Hadoop) => big \$\$\$\$ in salary.
 - 2015 Update: this is still true

MapReduce in R

- MapReduce library in R: library (mapReduce); mapReduce (map, reduce, data)
- But you have to think about what the map function is and what the reduce function is.
- Can test on your machine (may speed things a little), but more importantly: mapReduce package has support for farming out to cloud services like Amazon Elastic Map-Reduce, etc.
 - Note: IRBs may not like cloud computing.
 - Not sure mapReduce package it can be hooked up to our HPC
- Takeaway: if you think your data needs MapReduce scale processing, talk to me; I'd like to explore it further

Side note: the no R infrastructure MapReduce

- I had a 6000 row, 10000 column table and wanted to compute ROC curves for (almost) every column
 - Wrote code to compute ROC curves for n columns and write the result to disks
 - Executed for 300 columns at a time on high-performance cluster
 - Then wrote code to combine those results
- I didn't think about it this way at the time, but this is basically a manual map/reduce

High-performance computing cluster

- Basically a bunch of servers that you can use for scalable analysis
 - I've used the cluster at the Morningside campus
 - There is an Uptown cluster, but
 I've never used it

Glamorous computing power, 201? edition



Image from:

http://www.gamexeon.com/forum/h ardware-lobby/50639-supercomputer-system-tercepat-duniawarning-gambarnya-besar-besar.html

Summary

- Big data presents opportunity, but is also a pain in the neck
- Computing power mitigates some of the pain
- Because big data is "the future", there's a lot of interest around how we'll handle it, but technologies and systems are still getting figured out

References

- Large Datasets and You: <u>http://www.mattblackwell.org/files/papers/bigdata.pdf</u>
- FasteR! HigheR! StrongeR! A Guide to Speeding Up R Code for Busy People: <u>http://www.noamross.net/blog/2013/4/25/faster-talk.html</u>
- Scalable Strategies for Computing with Massive Data: <u>http://www.slideshare.net/fullscreen/joshpaulson/big-memory/l</u>
- Taking R to the Limit, Part II: Working with Large Datasets: <u>http://www.bytemining.com/wp-content/uploads/2010/08/r_hpc_II.pdf</u>