Large Datasets and You: A Field Guide[*]

Matthew Blackwell[†]
m.blackwell@rochester.edu

Maya Sen[‡]
msen@ur.rochester.edu

August 3, 2012

A wind of streaming data, social data and unstructured data is knocking at the door, and we're starting to let it in. It's a scary place at the moment.

Unidentified bank IT executive, as quoted by *The American Banker*

Error: cannot allocate vector of size 75.1 Mb

R

Introduction

The last five years have seen an explosion in the amount of data available to social scientists. Thanks to Twitter, blogs, online government databases, and advances in text analysis techniques, data sets with millions and millions of observations are no longer a rarity (Lohr, 2012). Although a blessing, these extremely large data sets can cause problems for political scientists working with standard statistical software programs, which are poorly suited to analyzing big data sets. At best, analyzing massive data sets can result in prohibitively long computing time; at worst, it can lead to repeated crashing, making anything beyond calculating the simplest of summary statistics impossible. The volume of data available to

[*]Comments and suggestions welcome.

[†]Assistant Professor, Department of Political Science, University of Rochester; Harkness Hall 322, Rochester, NY 14627 (http://www.mattblackwell.org).

[‡]Assistant Professor, Department of Political Science, University of Rochester; Harkness Hall 307, Rochester, NY 14627 (http://www.msen.co).

researchers is, however, growing faster than computational capacities, making developing techniques for how to handle "Big Data" is essential.

In this article, we describe a few approaches to handling these Big Data problems within the R programming language, both at the command line prior to R and after we fire up R.[1] We show that handling large datasets is about either (1) choosing tools that can shrink the problem or (2) fine-tuning R to handle massive data files.

Why Big Data Present Big Problems

It is no secret that current statistical software programs are not well equipped to handle extremely large datasets. R (R Development Core Team, 2012), for example, works by holding objects in its virtual memory, and big datasets are often larger then the size of the RAM that is available to researchers using their operating software. Many of these problems are compounded by the fact that not only do the raw loaded data take up RAM once loaded, but so do any analyses. Basic functions like `lm` and `glm` store multiple copies of the data within the workspace. Thus, even if the original data set is smaller than the allocated RAM, once multiple copies of the data are stored (via an `lm` function, for example), R will quickly run out of memory.

Purchasing more RAM is an option, as is moving to a server that can allocate more RAM. In addition, moving from a 32-bit to a 64-bit version of R can alleviate some problems. (Unix-like systems — e.g, Linux, Mac OS X — impose a 4Gb limit on 32-bit systems and no limit on 64-bit systems. On Windows, the limits are 2Gb and 4Gb for 32-bit and 64-bit respectively.) However, these fixes largely postpone the inevitable – scholars will (hopefully) continue to collect even larger datasets and push the boundaries of what is computationally possible. This will be compounded by running increasing numbers of more sophisticated analyses. In addition, all R builds have will have a maximum vector length of $2^{31} - 1$, or around two billion. A combination of any of these memory issues will result in the dreaded "cannot allocate vector size" error, which will swiftly derail a researcher's attempt at analyzing a large data set.

First Pass: Subset the Data

As simple as it sounds, the easiest work-around to the Big Data Problem is to avoid it if possible. After all, data files are often much larger than we need them to be; they usually contain more variables than we need for our analysis, or we plan to run our models on subsets of the data. In these cases, loading the excess data into the R workspace only to purge it (or ignore it) with a few commands later is incredibly wasteful in terms of memory. A better approach is to remove the excess data from the data file *before* loading it into R. This often appears difficult because we are used to performing data manipulation using R

[1]Here we focus on R, but this problem extends to other memory-based statistical environments, namely Stata. Other statistical packages, such as SAS and SPSS, have a file-based approach, which avoids some of these memory allocation issues.

(this is probably why we are using R in the first place!). Luckily, there are a handful of Unix command-line utilities that can help parse data files without running into memory issues.

We demonstrate this using a data file called `iris.tab`, which is tab-delimited and contains many rows. The dataset measures, in centimeters, (1) sepal length and width and (2) petal length and width for 50 flowers from three species of irises (Fisher, 1936). We can use the Unix `head` command to investigate the first ten lines of the data file:

```
mactwo$ head iris.tab
```

```
Sepal.Length     Sepal.Width     Petal.Length     Petal.Width     Species
5.1              3.5             1.4              0.2             setosa
4.9              3               1.4              0.2             setosa
4.7              3.2             1.3              0.2             setosa
4.6              3.1             1.5              0.2             setosa
5                3.6             1.4              0.2             setosa
5.4              3.9             1.7              0.4             setosa
4.6              3.4             1.4              0.3             setosa
5                3.4             1.5              0.2             setosa
4.4              2.9             1.4              0.2             setosa
```

Suppose that we only need the first four numeric variables for our analysis (we don't care about the iris species). We can remove the `Species` variable using the `cut` utility, which takes in a data file and a set of column numbers and returns the data file with only those columns.[2] For example, the following command:

```
mactwo$ head iris.tab | cut -f1,2,3,4
```

will return the data without the `Species` variable:

```
Sepal.Length     Sepal.Width     Petal.Length     Petal.Width
5.1              3.5             1.4              0.2
4.9              3               1.4              0.2
4.7              3.2             1.3              0.2
4.6              3.1             1.5              0.2
5                3.6             1.4              0.2
5.4              3.9             1.7              0.4
4.6              3.4             1.4              0.3
5                3.4             1.5              0.2
4.4              2.9             1.4              0.2
```

A few points of clarification. First, note that we are "piping" the output of the `head` command to the `cut` command to avoid running `cut` on the entire dataset.[3] This is useful for testing our approach at the command line. Once we have our syntax, we can run `cut` on the entire data as follows: `cut -f1,2,3,4 iris.tab >> iris-new.tab`. Note that this will create a new file, which may be quite large. Second, the `-f1,2,3,4` arugment specifies which columns to keep and can be specified by ranges such as `-f1-4`.

[2]We can also selectively load columns using the R command `read.table`; however, the approach we suggest is more efficient and is compatible with the `bigmemory` package below.

[3]In Unix environments, the "pipe character" (`|`) takes the output of one command and passes it as an input the next command.

In addition to removing variables, we often want to remove certain rows of the data (say, if we were running the analysis only on a subset of the data). To do this efficiently on large text-based data files, we can use `awk`, which comes standard on most Unix systems. The `awk` utility is a powerful data extraction tool, but we will only show its most basic features for selecting observations from a dataset. The command requires an expression that describes which rows of the data file to keep. For instance, if we wanted to keep the top row (with the variable names) and any row with a `Sepal.length` greater than 5, we could use the following:

```
mactwo$ head iris.tab | awk 'NR ==1 || $1 > 5'
```

This gives the following result:

```
Sepal.Length    Sepal.Width     Petal.Length    Petal.Width     Species
5.1             3.5             1.4             0.2             setosa
5.4             3.9             1.7             0.4             setosa
```

Here, `NR` refers to the row number, so that `NR == 1` selects the first row of the file, which contains the variable names. The `$` operator refers to column numbers, so that `$1 > 5` selects any row where the first column is greater than 5. The `||` operator simply tells `awk` to select rows that match either of the two criteria.

There are many ways to preprocess our data before loading it into `R` to reduce its size and make it more manageable. Besides these Unix tools we've discussed there are more complicated approaches, including scripting languages such as Python or relational database interfaces with `R` such as `sqldf` or `RODBC`. These are more powerful approaches, but often simple one-line Unix commands can wrangle data as effectively and more efficiently. In any case, this approach can resolve many of the supposed Big Data problems out in the wild without any further complications. There are times, though, when Big Data problems remain, even after whittling away the data to only the necessary bits.

The Bite-Sized-Chunk Approach to Big Data

It's impossible to eat a big steak in one bite; instead, we cut our steak into smaller pieces and eat it one bite after another. Nearly all direct fixes to the Big Data conundrum rely on the same principle: if we need all the data (not just some subsets of it), we can break up the data into more manageable chunks that are then small enough to fit within the allocated memory. Essentially, we upload into the workspace only as much of the data as is necessary to run specific analyses. Indeed, many operations can be done piece-meal or sequentially on different chunks on data – e.g., a thousand rows a time, or only a few columns. For some simple calculations, such as the sample mean, this process is straightforward. For others, though, this is more daunting—how do we piece together regressions from different subsets of the data?

Fortunately, there are a handful of packages that have facilitated the use of big data in `R` and they work by automating and simplifying the bite-sized data approach.[4] Generally,

[4]This bite-sized-chunk approach, sometimes called "split-apply-combine" (Wickham, 2011), has a strong history in computer science. Google's `MapReduce` programming model is essentially the same approach.

they allow most of the data to stay in the working directory (on a file in the hard drive); this means that the data do not have to be loaded into the memory (thereby using up valuable allocated memory). They create an R object within the memory that acts like a `matrix` object, but in reality it's just a way for you to efficiently access different parts of the data file (still on the hard drive). In addition, they provide intuitive functions that allow users to access the data and calculate summary statistics of the entire data.

A Romp Through `bigmemory`

The `bigmemory` package (Kane and Emerson, 2011), along with its sister packages, allows users to interact with and analyze incredibly large datasets. To illustrate, we work through an example using U.S. lending data from 2006. Federal law mandates that all applications for a real estate mortgage be recorded by the lending agency and reported to the relevant U.S. agencies (who then make the data publicly available). This results in a wealth of data – some 11 million observations per year. However, the size of this data means that loading the data into an R workspace is essentially impossible, let alone running linear or generalized linear models.

To get started, we load the data into R as a `big.matrix` object. With large datasets, it is important to create a "backing file," which will reduce the amount of memory that R needs to access the data. To do this, we load the relevant packages and use the `read.big.matrix` function:

```
> library(bigmemory)

bigmemory >= 4.0 is a major revision since 3.1.2; please see package
biganalytics and http://www.bigmemory.org for more information.

> library(biglm)
Loading required package: DBI
> library(biganalytics)
> library(bigtabulate)
> mortgages <- read.big.matrix("allstates.txt", sep = "\t", header = TRUE,
+                              type = "double",
+                              backingfile = "allstates.bin",
+                              descriptor = "allstates.desc")
```

The resulting object, `mortgages`, is a `big.matrix` object and takes up very little memory in R. The process of creating this backing file takes around 25-30 minutes with these data, but after this is complete, it is fast and easy to load the `big.matrix` object in a fresh R session using the following:

```
> library(bigmemory)

bigmemory >= 4.0 is a major revision since 3.1.2; please see package
biganalytics and http://www.bigmemory.org for more information.

> library(biglm)
Loading required package: DBI
```

```
> library(biganalytics)
> library(bigtabulate)
> xdesc <- dget("allstates-clean.desc")
> mortgages <- attach.big.matrix(xdesc)
```

This process takes just seconds with same low memory overhead.

In many ways, we can interact with `big.matrix` objects in much the same way we do with `matrix` objects:

```
> dim(mortgages)
[1] 10875481       40
> head(mortgages)[,1:7]
     ID agency loan.type property loan.purpose occupancy loan.amount
[1,]  1      1         1        1            1         1          36
[2,]  1      1         1        1            1         1          61
[3,]  1      1         1        1            2         1          10
[4,]  1      1         1        1            3         1          76
[5,]  1      1         1        1            3         1         148
[6,]  1      1         1        1            3         1         132
```

We can see here that this dataset has over 10.8 million observations with 40 variables. Using functions from the sister package `biganalytics` (Emerson and Kane, 2010), we can quickly and easily find summary statistics on different columns:

```
> mean(mortgages[,"income"])
[1] 100.3027
> median(mortgages[,"income"])
[1] 76
```

These calculations take just a few seconds on Apple iMac with a 3.06 Ghz Intel Core 2 Duo processor and 4Gb of RAM. Note that `big.matrix` objects mimic `matrix` objects so we cannot use the `mortgages$income` syntax as we would with a `data.frame`.

Our data analyses often require more than simple summary statistics and `bigmemory` has a way to efficiently subset data. This is the `mwhich` command, which returns a vector of indicies that match a set of criteria similar to the base `which` command. The function takes in a `big.matrix` object, a variable name, a value, and a comparison to perform on that value. For instance, with a tradition R `matrix`, we might choose the males with `which(mortgages[,"sex"] == 1)`, but the syntax is slightly different with a `big.matrix`:

```
> median(mortgages[mwhich(mortgages, "sex", 1, "eq"), "loan.amount"])
[1] 141
> median(mortgages[mwhich(mortgages, "sex", 2, "eq"), "loan.amount"])
[1] 127
```

The first call to `mwhich` selects the observations with `sex` equal (`eq`) to 1. The `mwhich` function can compare multiple variables or values at once. This allows us to create complex cross-tabulations on extremely large datasets with minimal memory or speed overhead.

Finally, the `biganalytics` also provides a method for passing `big.matrix` objects to the `biglm` function (Lumley, 2011), which efficiently computes ordinary least squares on large datasets. This approach is similar to running a regression with a normal `matrix`:

```
> mod1 <- biglm.big.matrix(high.rate ~ sex, data = mortgages)
> summary(mod1)
Large data regression model: biglm(formula = formula, data = data, ...)
Sample size =  10875481
               Coef   (95%    CI)    SE p
(Intercept) 0.2331 0.2324 0.2338 4e-04 0
sex         0.0371 0.0367 0.0376 2e-04 0
```

Remarkably, this regression takes less than a minute to run (on all 10.8 million observations!). In general, the `bigmemory` suite of functions helps users with extremely large datasets avoid per-computation memory and speed issues by creating a file-backed version of the data. Further, they have put together a great set of functions to help users with the most common statistical tasks. These tasks can be sped up even more by parallel processing through the `foreach` package.

Conclusion

As journalists and public intellectuals have noted, the age of "Big Data" has dawned. However, advances in computational speed and memory size are not moving fast enough to allow analysis of this data with traditional techniques. To this end, researchers analyzing extremely large data sets will have to start start using different kinds of approaches – parallel processing, big data packages. Here, we have reviewed only some techniques in tackling "Big Data," but there are others. Ultimately, as the wealth of data only grows, those who can quickly and easily digest this information will be able to explore new and exciting research questions; those who can't will, unfortunately, be left out.

References

Emerson, John W. and Michael J. Kane. 2010. *biganalytics: A library of utilities for big.matrix objects of package bigmemory.* R package version 1.0.14.
URL: *http://CRAN.R-project.org/package=biganalytics*

Fisher, R.A. 1936. "The use of multiple measurements in taxonomic problems." *Annals of Human Genetics* 7(2):179–188.

Kane, Michael J. and John W. Emerson. 2011. *bigmemory: Manage massive matrices with shared memory and memory-mapped files.* R package version 4.2.11.
URL: *http://CRAN.R-project.org/package=bigmemory*

Lohr, Steve. 2012. "The Age of Big Data." *The New York Times* p. SR1.
URL: *http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html*

Lumley, Thomas. 2011. *biglm: bounded memory linear and generalized linear models.* R package version 0.8.
URL: *http://CRAN.R-project.org/package=biglm*

R Development Core Team. 2012. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0.
URL: *http://www.R-project.org/*

Wickham, Hadley. 2011. "The split-apply-combine strategy for data analysis." *Journal of Statistical Software* 40(1):129.
URL: *http://www.jstatsoft.org/v40/i01/*