Databases and Big Data

October 17, 2013

References:

- Murrell: Introduction to Data Technologies
- Adler: R in a Nutshell

I've also pulled material from a variety of other sources, some mentioned in context below.

1 A few preparatory notes

1.1 An editorial on 'big data'

Big data is trendy these days. Personally, I think some of the hype is justified and some is hype. Large datasets allow us to address questions that we can't with smaller datasets, and they allow us to consider more sophisticated (e.g., nonlinear) relationships than we might with a small dataset. But they do not directly help with the problem of correlation not being causation. Having medical data on every American still doesn't tell me if higher salt intake causes hypertension. Internet transaction data does not tell me if one website feature causes increased viewership or sales. One either needs to carry out a designed experiment or think carefully about how to infer causation from observational data. Nor does big data help with the problem that an ad hoc 'sample' is not a statistical sample and does not provides the ability to directly infer properties of a population. A well-chosen smaller dataset may be much more informative than a much larger, more ad hoc dataset. However, having big datasets might allow you to select from the dataset in a way that helps get at causation or in a way that allows you to construct a population-representative sample.

Different people define the 'big' in big data differently. Our efforts here will focus on dataset sizes that are large for traditional statistical work but would probably not be thought of as large in some contexts such as Google or the NSA.

1.2 Logistics

One of the main drawbacks with R in working with big data is that all objects are stored in memory, so you can't directly work with datasets that are more than 1-20 Gb or so, depending on the memory on your machine.

Note: in handling big data files, it's best to have the data on the local disk of the machine you are using to reduce traffic and delays from moving data over the network.

1.3 What we already know about handling big data!

UNIX operations are generally very fast, so if you can manipulate your data via UNIX commands and piping, that will allow you to do a lot. We've already seen UNIX commands for extracting columns. And various commands such as *grep*, *head*, *tail*, etc. allow you to pick out rows based on certain criteria. As some of you have done in problem sets, one can use *awk* to extract rows. So basic shell scripting may allow you to reduce your data to a more manageable size.

Also, the example datasets in Section 3 are not good illustrations of this, but as we'll see scattered throughout the Unit, there are more compact ways of storing data than in flat text (e.g., csv) files.

2 Databases

2.1 Overview

A relational database stores data as a set of tables (or relations), which are rather similar to R data frames, in that a table is made up of columns or fields, each containing a single type (numeric, character, date, currency, ...) and rows or records containing the observations for one entity. One principle of databases is that if a category is repeated in a given variable, you can more efficiently store information about each level of the category in a separate table; consider information about people living in a state and information about each state - you don't want to include variables that only vary by state in the table containing information about individuals (at least until you're doing the actual analysis that needs the information in a single table). Or consider students nested within classes nested within schools. Databases are set up to allow for fast querying and merging (called *joins* in database terminology).

You can interact with databases in a variety of database systems (DBMS=database management system) (some systems are *SQLite*, *MySQL*, *postgreSQL*, *Oracle*, *Access*). We'll concentrate on accessing data in a database rather than management of databases. SQL is the *Structured Query*

Language and is a special-purpose language for managing databases and making queries. Variations on SQL are used in many different DBMS.

Many DBMS have a client-server model. Clients connect to the server, with some authentication, and make requests. We'll concentrate here on a simple DBMS, *SQLite*, that allows us to just work on our local machine, with the database stored as a single file.

There are often multiple ways to interact with a DBMS, including directly using command line tools provided by the DBMS or via Python or R, among others.

We'll use an SQLite database available on any SCF machine at */mirror/data/pub/html/scf/cis.db* as our example database. This is a database of the metadata (authors, titles, years, journal, etc.) for articles published in Statistics journals over the last century. First, let's talk through how one would set up a relational database to store journal article information.

2.2 Accessing databases in R

In R, the *DBI* package provides a front-end for manipulating databases from a variety of DBMS (MySQL, SQLite, Oracle, among others). Basically, you tell the package what DBMS is being used on the backend, link to the actual database, and then you can use the syntax in the package.

First we'll connect to the database and get some information on the *schema*, i.e., the structure of the database.

```
library(RSQLite)

## Loading required package:  DBI


fileName <- "/mirror/data/pub/html/scf/cis.db"
drv <- dbDriver("SQLite")
db <- dbConnect(drv, dbname = fileName)  # using a connection once again!
# con <- dbConnect(SQLite(), dbname = fileName) # alternative

# get information on the database schema
dbListTables(db)

##  [1] "articles"      "authors"       "authorships"   "books"
##  [5] "contacts"      "delayed_jobs"  "isbns"         "issns"
##  [9] "issues"        "journals"      "tag_relations" "taggings"
## [13] "tags"          "volumes"
```

```
dbListFields(db, "articles")
```

```
##  [1] "id"         "type"       "id_entity"  "id_title"   "title"
##  [6] "year"       "volume"     "number"     "page_start" "page_end"
## [11] "url"        "journal"    "journal_id" "volume_id"  "issue_id"
## [16] "zmath"
```

```
dbListFields(db, "authors")
```

```
## [1] "id"   "name"
```

```
dbListFields(db, "authorships")
```

```
## [1] "id"               "id_title"         "author_id"
## [4] "editor"           "sequence"         "publication_id"
## [7] "publication_type"
```

For queries, SQL has statements like:

`SELECT var1, var2, var3 FROM tableX WHERE condition1 AND condition2 ORDER BY var4`

E.g., *condition1* might be `latitude > 80` or `name = 'Breiman'` or `company in ('IBM', 'Apple', 'Dell')`. Now we'll do some queries to pull together information we want. Because of the relational structure, to extract the titles for a given author, we need to do a series of queries.

```
auth <- dbSendQuery(db, "select * from authorships")
fetch(auth, 5)
```

```
##   id id_title author_id editor sequence publication_id publication_type
## 1  3        2         1      f        0              1        Article\n
## 2  4        3         3      f        0              2        Article\n
## 3  5        4         4      f        0              3        Article\n
## 4  6        5         5      f        0              4        Article\n
## 5  7        6         6      f        0              5        Article\n
```

```
dbClearResult(auth)
```

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

```
query <- "select id from authors where name like 'Breiman%'"
a_ids <- dbGetQuery(db, query)


a_ids <- as.list(unlist(a_ids))
query <- paste("select id_title from authorships where author_id in (",
               paste(rep("?", length(a_ids)), collapse = ","), ")")
query
```

```
## [1] "select id_title from authorships where author_id in ( ?,? )"
```

```
a_ids
```

```
## $id1
## [1] 532
##
## $id2
## [1] 1141
```

```
t_ids <- dbGetQuery(db, query, a_ids)
t_ids$id_title[1:5]
```

```
## [1]  593 1062 1087 1089 1440
```

```
t_ids <- as.list(unlist(t_ids))
query <- paste("select * from articles where id_title in (",
               paste(rep("?", length(t_ids)), collapse = ","), ")")
titles <- dbGetQuery(db, query, t_ids)
head(titles)
```

```
##     id    type  id_entity id_title
## 1  445 Article 1000000073      593
## 2  913 Article 1000000105     1062
## 3  938 Article 1000000105     1087
## 4  940 Article 1000000105     1089
## 5 1863 Article 1000000145     2156
```

```
## 6 2287 Article 1000000161     2580
##                                                                        tit
## 1 The individual ergodic theorem of information theory (Corr: V31 p809-81
## 2             The capacities of certain channel classes under random codi
## 3                                  On the completeness of order statisti
## 4              The strong law of large numbers for a class of Markov chai
## 5                                The Poisson tendency in traffic distorti
## 6                                    Consistent estimates and zero-one se
##   year volume number page_start page_end url journal journal_id volume_id
## 1 1957     28      0        809      811                  1748      7998
## 2 1960     31      0        558      567                  1748      9117
## 3 1960     31      0        794      797                  1748      9117
## 4 1960     31      0        801      803                  1748      9117
## 5 1963     34      0        308      311                  1748      9865
## 6 1964     35      0        157      161                  1748     10084
##   issue_id zmath
## 1       74    \n
## 2      106    \n
## 3      106    \n
## 4      106    \n
## 5      146    \n
## 6      162    \n

# do a google scholar check to see that things seem to be ok
```

Note that we were able to insert values from R into the set used to do the selection.

Now let's see a *join* (by default this is an "*inner join*" – see below) of multiple tables, combined with a query. This allows us to extract the information on Breiman's articles more easily.

```
# alternatively, we can do a query that involves multiple tables
info <- dbGetQuery(db, "select * from articles, authors, authorships where a
# 'select * from articles, authors, authorships where authors.name like
# 'Breiman%' and authors.id = authorships.author_id and
# authorships.id_title = articles.id_title'
head(info)

##     id    type  id_entity id_title
```

```
## 1  445 Article 1000000073      593
## 2  913 Article 1000000105     1062
## 3  938 Article 1000000105     1087
## 4  940 Article 1000000105     1089
## 5 1863 Article 1000000145     2156
## 6 2287 Article 1000000161     2580
##                                                                      tit
## 1 The individual ergodic theorem of information theory (Corr: V31 p809-81
## 2             The capacities of certain channel classes under random codi
## 3                                  On the completeness of order statisti
## 4              The strong law of large numbers for a class of Markov chai
## 5                                The Poisson tendency in traffic distorti
## 6                                    Consistent estimates and zero-one se
##   year volume number page_start page_end url journal journal_id volume_id
## 1 1957     28      0        809      811                  1748      7998
## 2 1960     31      0        558      567                  1748      9117
## 3 1960     31      0        794      797                  1748      9117
## 4 1960     31      0        801      803                  1748      9117
## 5 1963     34      0        308      311                  1748      9865
## 6 1964     35      0        157      161                  1748     10084
##   issue_id zmath  id           name   id id_title author_id editor
## 1       74    \n 532 Breiman, Leo\n  696      593       532      f
## 2      106    \n 532 Breiman, Leo\n 1355     1062       532      f
## 3      106    \n 532 Breiman, Leo\n 1391     1087       532      f
## 4      106    \n 532 Breiman, Leo\n 1393     1089       532      f
## 5      146    \n 532 Breiman, Leo\n 2847     2156       532      f
## 6      162    \n 532 Breiman, Leo\n 3413     2580       532      f
##   sequence publication_id publication_type
## 1        0            445        Article\n
## 2        1            913        Article\n
## 3        2            938        Article\n
## 4        0            940        Article\n
## 5        0           1863        Article\n
## 6        0           2287        Article\n
```

Finally, let's see the idea of creating a *view*, which you can think of as a new table, though the

DBMS is not actually explicitly constructing such a table.

```
# that db is read-only; to create a view we need to be able to modify it
system(paste0("cp ", fileName, " /tmp/."))
dbDisconnect(db)

## [1] TRUE

db <- dbConnect(drv, dbname = "/tmp/cis.db")

# finally, we can create a view that amounts to joining the tables
fullAuthorInfo <- dbSendQuery(db, "create view fullAuthorInfo as select * fr
# 'create view fullAuthorInfo as select * from authors join authorships on
# authorships.author_id = authors.id'

partialArticleInfo <- dbSendQuery(db, "create view partialArticleInfo as sel
# 'create view partialArticleInfo as select * from articles join
# fullAuthorInfo on articles.id_title=fullAuthorInfo.id_title'

fullInfo <- dbSendQuery(db, "select * from journals join partialArticleInfo
# 'select * from journals join partialArticleInfo on journals.id =
# partialArticleInfo.journal_id')
subData <- fetch(fullInfo, 3)
subData

##     id                                name articles_count min_year
## 1 1748 The Annals of Mathematical Statistics              0      \\N
## 2  452                          Econometrica              0      \\N
## 3 1746            The American Statistician              0      \\N
##   max_year                             publisher url mathscinet_id
## 1      \\N  Institute of Mathematical Statistics               \\N
## 2      \\N Blackwell Scientific Publications Ltd               \\N
## 3      \\N      American Statistical Association               \\N
##   english_only electronic_only url_only publisher_society admin_comments
## 1          \\N             \\N      \\N               \\N            \\N
## 2          \\N             \\N      \\N               \\N            \\N
## 3          \\N             \\N      \\N               \\N            \\N
```

```
##    core id    type  id_entity id_title
## 1 \\N\n  1 Article 1000000001        2
## 2 \\N\n  2 Article 1000000002        3
## 3 \\N\n  3 Article 1000000003        4
##
## 1 The non-central Wishart distribution and certain problems of multivaria
## 2                  Capital expansion, rate of growth, and employment (Re
## 3
##   year volume number page_start page_end url journal journal_id volume_id
## 1 1946     17      0        409      431                  1748      4170
## 2 1946     14      0        137      147                   452      2723
## 3 1947      1      0          7       11                  1746       273
##   issue_id zmath id:1                  name id:2 id_title:1 author_id
## 1        2    \n    1      Anderson, T. W.\n    3          2         1
## 2        3    \n    3      Domar, Evsey D.\n    4          3         3
## 3        4    \n    4 Tumbleson, Robert C.\n    5          4         4
##   editor sequence publication_id publication_type
## 1      f        0              1        Article\n
## 2      f        0              2        Article\n
## 3      f        0              3        Article\n

dbClearResult(fullInfo)

## [1] TRUE
```

As seen above, you can also use *dbSendQuery()* combined with *fetch()* to pull in a fixed number of records at a time, if you're working with a big database.

2.3 Details on joins

A bit more on joins - as we saw with *merge()* in R, there are various possibilities for how to do the merge depending on whether there are rows in one table that are not in another table. In other words, we need to think about whether the relationship between tables is one-to-one, one-to-many, or many-to-many. In database terminology an *inner join* is when you get the rows for which there is data in both tables. A *left outer join* gives all the rows from the first table but only those from the second table that match a row in the first table. A *right outer join* is the reverse, while a *full outer join* returns all rows from both tables. A *cross join* gives the Cartesian product, namely the

combination of every row from each table, analogous to *expand.grid()* in R. However a *cross join* with a *where* statement can duplicate the result of an *inner join*:

```
select * from table1 cross join table2 where table1.id = table2.id
select * from table1 join table2 on table1.id = table2.id
```

2.4 Keys and indices

A key is a field or collection of fields that gives a unique value for every row/observation. A table in a database should then have a primary key that is the main unique identifier used by the DBMS. Foreign keys are columns in one table that give the value of the primary key in another table.

An index is an ordering of rows based on one or more fields. DBMS use indices to look up values quickly. (Recall our discussion in Unit 6 on looking up values by name vs. index and the benefits of hashing.) So in general you want your tables to have indices. And having indices on the columns used in the matching for a join allows for quick joins. DBMS use indexing to provide sub-linear time lookup, so that lookup is faster than linear time ($O(n)$ when there are n rows), which is what would occur if one had to look at each row sequentially. Lookup may be logarithmic [$O(log(n))$] or constant time [$O(1)$]. A binary search is logarithmic while looking up based on numeric position is $O(1)$.

So if you're working with a database and speed is important, check to see if there are indices.

2.5 Creating SQLite database tables from R

I won't do a full demo of this, but the basic syntax for this is as follows. You can read from a CSV to create the table or from an R dataframe. The following assumes you have two tables stored as CSVs, with one table of student info and one table of class info.

```
dbWriteTable(conn = db, name = "student", value = "student.csv",
   row.names = FALSE, header = TRUE)
dbWriteTable(conn = db, name = "class", value = "class.csv",
   row.names = FALSE, header = TRUE)
# alternatively
school <- read.csv("school.csv") # Read csv files into R
class <- read.csv("class.csv")
# Import data frames into database
dbWriteTable(conn = db, name = "student", value = student,
   row.names = FALSE)
dbWriteTable(conn = db, name = "class", value = class,
```

```
   row.names = FALSE)
```

2.6 SAS

SAS is quite good at handling large datasets, storing them on disk rather than in memory. I have used SAS in the past for subsetting and merging large datasets. Then I will generally extract the data I need for statistical modeling and do the analysis in R.

Here's an example of some SAS code for reading in a CSV followed by some subsetting and merging and then output.

```
/* we can use a pipe - in this case to remove carriage returns, */
/* presumably because the CSV file was created in Windows */
filename tmp pipe "cat ~/shared/hei/gis/100w4kmgrid.csv | tr -d '\r'";

/* read in one data file */
data grid;
infile tmp
lrecl=500 truncover dsd firstobs=2;
informat gridID x y landMask dataMask;
input gridID x y landMask dataMask;
run ;

filename tmp pipe "cat ~/shared/hei/GOES12/goes/Goes_int4km.csv | tr -d '\r

/* read in second data file */
data match;
infile tmp
lrecl=500 truncover dsd firstobs=2;
informat goesID gridID areaInt areaPix;
input goesID gridID areaInt areaPix;
run ;

/* need to sort before merging */
proc sort data=grid;
    by gridID;
run;
```

```
proc sort data=match;
    by gridID;
run;

/* notice some similarity to SQL */
data merged;
merge match(in=in1) grid(in=in2);
by gridID;  /* key field */
if in1=1;   /* also do some subsetting */
/* only keep certain fields */
keep gridID goesID x y landMask dataMask areaInt areaPix;
run;

/* do some subsetting */
data PA;   /* new dataset */
    set merged;  /* original dataset */
    if x<1900000 and x>1200000 and y<2300000 and y>1900000;
run;


%let filename="~/shared/hei/code/model/GOES-gridMatchPA.csv";
/* output to CSV */
PROC EXPORT DATA= WORK.PA
            OUTFILE= &filename
            DBMS=CSV REPLACE;
RUN;
```

Note that SAS is oriented towards working with data in a "data frame"-style format; i.e., rows as observations and columns as fields, with different fields of possibly different types. As you can see in the syntax above, the operations concentrate on transforming one dataset into another dataset.

3 R and big data

There has been a lot of work in recent years to allow R to work with big datasets.

The *ff* and *bigmemory* packages provide the ability to load datasets into R without having them

in memory, but rather stored in clever ways on disk that allow for fast access. Metadata is stored in R.

The *biglm* package provides the ability to fit linear models and GLMs to big datasets, with integration with *ff* and *bigmemory*.

3.1 Working with big datasets on disk: ff and bigmemory

We'll work through an example with US government data on airline delays (1987-2008) available through the ASA 2009 Data Expo at http://stat-computing.org/dataexpo/2009/the-data.html.

First we'll use UNIX tools to download the individual yearly CSV files and make a single CSV (~12 Gb). See the demo code file for the bash code.

Now we can read the data into R using the *ff* package, in particular reading in as an *ffdf* object. Note the arguments are similar to those for *read.{table,csv}()*. *read.table.ffdf()* reads the data in chunks.

```
require(ff)
require(ffbase)

# I put the data file on local disk on the machine I am using
# (/tmp on arwen)
fileName <- 'test.csv'
dat <- read.csv.ffdf(file = fileName, header = TRUE,
     colClasses = c('integer', rep('factor', 3),
     rep('integer', 4), 'factor', 'integer', 'factor',
     rep('integer', 5), 'factor','factor', rep('integer', 4),
     'factor', rep('integer', 6)))


fileName <- 'AirlineDataAll.csv'
system.time(  dat <- read.csv.ffdf(file = fileName, header = TRUE,
    colClasses = c('integer', rep('factor', 3), rep('integer', 4),
    'factor', 'integer', 'factor', rep('integer', 5), 'factor',
    'factor', rep('integer', 4), 'factor', rep('integer', 6))) )
# takes about 40 minutes

system.time(ffsave(dat, file = '/tmp/AirlineDataAll'))
## file is saved as AirlineDataAll.ffData
```

```
## with metadata in AirlineDataAll.RData

## takes a while - I forgot to record how long

system.time(ffload('/tmp/AirlineDataAll'))
# this is much quicker:
# 78.156  15.836 169.974
```

We can write a copy of the file in the ff binary format that can be read more quickly back into R than the original reading of the CSV using *ffsave()* and *ffload()*. Also note the reduced size of the binary format file compared to the original CSV. It's good to be aware of where the binary ff file is stored. With *ff* (I think *bigmemory* is different in how it handles this) it appears to be stored in */tmp* in an R temporary directory. Note that as we work with large files we need to be more aware of the filesystem, making sure in this case that */tmp* has enough space.

Note that a copy of an *ff* object appears to be a shallow copy.

Next let's do a bit of exploration of the dataset. Of course in a real analysis we'd do a lot more and some of this would take some time.

```
# load again as previous chunk not run w/in pdf compilation

# note that ideally we'd want this on local disk

ffload("/tmp/AirlineDataAll")
# [1] 'tmp/RtmpU5Uw6z/ffdf4e684aecd7c4.ff'
# 'tmp/RtmpU5Uw6z/ffdf4e687fb73a88.ff' [3]
# 'tmp/RtmpU5Uw6z/ffdf4e6862b1033f.ff'
# 'tmp/RtmpU5Uw6z/ffdf4e6820053932.ff' [5]
# 'tmp/RtmpU5Uw6z/ffdf4e681e7d2235.ff' 'tmp/RtmpU5Uw6z/ffdf4e686aa01c8.ff'
# ...

dat$Dest
# ff (closed) integer length=123534969 (123534969) levels: BUR LAS LAX OAK
# PDX RNO SAN SFO SJC SNA ABE ABQ ACV ALB ALO AMA ANC ATL AUS AVP AZO BDL
# BFL BGR BHM BIL BLI BNA BOI BOS BTV BUF BWI CAE CAK CCR CHS CID CLE CLT
# CMH CMI COS CPR CRP CRW CVG DAB DAL DAY DCA DEN DFW DLH DRO DSM DTW ELP
# EUG EVV EWR FAI FAR FAT FLG FLL FOE FSD GCN GEG GJT GRR GSO GSP GTF HNL
```

```
# HOU HPN HRL HSV IAD IAH ICT ILG ILM IND ISP JAN JAX JFK KOA LBB LEX LGA
# LGB LIH LIT LMT LNK MAF MBS MCI MCO MDT MDW MEM MFR MHT MIA MKE MLB MLI
# MOB MRY MSN MSP MSY OGG OKC OMA ONT ORD ORF PBI PHL PHX PIA PIT PNS PSC
# ...

DestTable <- sort(table.ff(dat$Dest), decreasing = TRUE)
# table is not a generic...

# takes a while

# ORD ATL DFW LAX PHX DEN DTW IAH MSP SFO

# 6638035 6094186 5745593 4086930 3497764 3335222 2997138 2889971 2765191
# 2725676

# STL EWR LAS CLT LGA BOS PHL PIT SLC SEA

# 2720250 2708414 2629198 2553157 2292800 2287186 2162968 2079567 2004414
# 1983464

# looks right - the busiest airports are ORD (O'Hare in Chicago) and ATL
# (Atlanta)

dat$DepDelay[1:50]
# opening ff /tmp/RtmpU5Uw6z/ffdf4e682d8cd893.ff [1] 11 -1 11 -1 19 -2 -2
# 1 14 -1 5 16 17 1 21 3 13 -1 87 19 31 17 32 0 1 [26] 29 26 15 5 54 0 25
# -2 0 12 14 -1 2 1 16 15 44 20 15 3 21 -1 0 7 23

min.ff(dat$DepDelay, na.rm = TRUE)
# [1] -1410
max.ff(dat$DepDelay, na.rm = TRUE)
# [1] 2601

# tmp <- clone(dat$DepDelay) # make a deep copy
```

A note of caution. Debugging code involving *ff* can be a hassle because the size gets in the

way in various ways. Until you're familiar with the various operations on ff objects, you'd be wise to try to run your code on a small test dataset loaded in as an ff object. Also, we want to be sure that the operations we use keep any resulting large objects in the *ff* format and use *ff* methods and not standard R functions.

3.2 Fitting models to big datasets: biglm

The *biglm* package provides the ability to fit large linear models and GLMs. *ffbase* has a *bigglm.ffdf()* function that builds on *biglm* for use with ffd objects. Let's try a few basic models on the airline data.

```
require(ffbase)
require(biglm)

datUse <- subset(dat, dat$DepDelay < 60*12 & dat$DepDelay > (-30) &
                 !is.na(dat$DepDelay))

# any concern about my models?
system.time(mod <- bigglm(DepDelay ~ Year, data = datUse))

system.time(mod <- bigglm(DepDelay ~ Year + Month + DayOfWeek, data = dat))
```

Given the time involved, I ran that code separately from compiling the pdf. Here are the results:

```
# basic model
Large data regression model: bigglm(DepDelay ~ Year, data = datUse)
Sample size =  121216293
                 Coef      (95%       CI)     SE p
(Intercept) -286.7616 -288.3136 -285.2096 0.7760 0
Year           0.1475    0.1468    0.1483 0.0004 0

# expanded model
Large data regression model: bigglm(DepDelay ~ Year + Month + DayOfWeek, dat
Sample size =  123534969
                 Coef      (95%       CI)     SE p
(Intercept) -286.7663 -288.4059 -285.1266 0.8198 0
Year           0.1464    0.1456    0.1472 0.0004 0
```

```
Month11        0.8273    0.8026    0.8520 0.0124 0
Month12        5.2685    5.2439    5.2931 0.0123 0
Month1         3.1617    3.1368    3.1865 0.0124 0
Month2         2.6968    2.6714    2.7222 0.0127 0
Month3         2.3926    2.3679    2.4172 0.0123 0
Month4         0.5518    0.5270    0.5767 0.0124 0
Month5         0.5853    0.5606    0.6100 0.0123 0
Month6         3.9741    3.9493    3.9989 0.0124 0
Month7         3.5967    3.5721    3.6212 0.0123 0
Month8         2.5625    2.5380    2.5870 0.0123 0
Month9        -0.9423   -0.9673   -0.9173 0.0125 0
DayOfWeek2    -0.9901   -1.0090   -0.9712 0.0095 0
DayOfWeek3    -0.2004   -0.2193   -0.1815 0.0094 0
DayOfWeek4     1.3871    1.3682    1.4059 0.0094 0
DayOfWeek5     2.2921    2.2732    2.3110 0.0094 0
DayOfWeek6    -0.9619   -0.9814   -0.9424 0.0098 0
DayOfWeek7     0.5449    0.5258    0.5641 0.0096 0
```

Of course as good statisticians/data analysts we want to do careful assessment of our model, consideration of alternative models, etc. This is going to be harder to do with large datasets than with more manageable ones. However, one possibility is to do the diagnostic work on subsamples of the data.

Now let's consider the fact that very small substantive effects can be highly statistically significant when estimated from a large dataset. In this analysis the data are generated from $Y \sim \mathcal{N}(0 + 0.001x, 1)$, so the R^2 is essentially zero.

```
n <- 150000000  # n*4*8/1e6 Mb of RAM (~5 Gb)
# but turns out to be 11 Gb as a text file
nChunks <- 100
chunkSize <- n/nChunks

set.seed(0)

for(p in 1:nChunks) {
  x1 <- runif(chunkSize)
  x2 <- runif(chunkSize)
```

```
  x3 <- runif(chunkSize)
  y <- rnorm(chunkSize, .001*x1, 1)
  write.table(cbind(y,x1,x2,x3), file = '/tmp/signif.csv',
     sep = ',', col.names = FALSE,  row.names = FALSE,
     append = TRUE, quote = FALSE)
}


fileName <- '/tmp/signif.csv'
system.time(  dat <- read.csv.ffdf(file = fileName,
   header = FALSE, colClasses = rep('numeric', 4)))
# 922.213  18.265 951.204

names(dat) <- c('y', 'x1','x2', 'x3')
ffsave(dat, file = '/tmp/signif')
```

```
system.time(ffload('/tmp/signif'))
# 52.323   7.856  60.802

system.time(mod <- bigglm(y ~ x1 + x2 + x3, data = dat))
#  1957.358    8.900 1966.644

options(digits = 12)
summary(mod)


# R^2 on a subset (why can it be negative?)
coefs <- summary(mod)$mat[,1]
wh <- 1:1000000
1 - sum((dat$y[wh] - coefs[1] + coefs[2]*dat$x1[wh] +
  coefs[3]*dat$x2[wh] + coefs[4]*dat$x3[wh])^2) /
  sum((dat$y[wh] - mean(dat$y[wh]))^2)
```

Given the time involved, I ran that code separately from compiling the pdf. Here are the results:

```
Large data regression model: bigglm(y ~ x1 + x2 + x3, data = dat)
```

```
Sample size = 1.5e+08
               Coef          (95%       CI)           SE         p
(Intercept) -0.0001437 -0.0006601 0.0003727 0.0002582 0.5777919
x1           0.0013703  0.0008047 0.0019360 0.0002828 0.0000013
x2           0.0002371 -0.0003286 0.0008028 0.0002828 0.4018565
x3          -0.0002620 -0.0008277 0.0003037 0.0002829 0.3542728
### and here is the R^2 calculation (why can it be negative?)
[1] -1.111046828e-06
```

So, do I care the result is highly significant? Perhaps if I'm hunting the Higgs boson... As you have hopefully seen in statistics courses, statistical significance $\neq$ practical significance.

4 Sparsity

A lot of statistical methods are based on sparse matrices. These include:

- Matrices representing the neighborhood structure (i.e., conditional dependence structure) of networks/graphs.
- Matrices representing autoregressive models (neighborhood structure for temporal and spatial data)
- A statistical method called the *lasso* is used in high-dimensional contexts to give sparse results (sparse parameter vector estimates, sparse covariance matrix estimates)
- There are many others (I've been lazy here in not coming up with a comprehensive list, but trust me!)

When storing and manipulating sparse matrices, there is no need to store the zeros, nor to do any computation with elements that are zero.

R, Matlab and Python all have functionality for storing and computing with sparse matrices. We'll see this a bit more in the linear algebra unit.

```
require(spam)
mat = matrix(rnorm(1e+08), 10000)
mat[mat > (-2)] <- 0
sMat <- as.spam(mat)
print(object.size(mat), units = "Mb")
```

```
## 762.9 Mb

print(object.size(sMat), units = "Mb")

## 26.1 Mb


vec <- rnorm(10000)
system.time(mat %*% vec)

##    user  system elapsed
##   0.412   0.000   0.401

system.time(sMat %*% vec)

##    user  system elapsed
##   0.012   0.000   0.010
```

5 Using statistical concepts to deal with computational bottlenecks

As statisticians, we have a variety of tools that can aid in dealing with big data.

1. Usually we take samples because we cannot collect data on the entire population. But we can just as well take a sample because we don't have the ability to process the data from the entire population. We can use standard uncertainty estimates to tell us how close to the true quantity we are likely to be. And we can always take a bigger sample if we're not happy with the amount of uncertainty.

2. There are a variety of ideas out there for making use of sampling to address big data challenges. One idea (due in part to Prof. Mike Jordan here in Statistics/EECS) is to compute estimates on many (relatively small) bootstrap samples from the data (cleverly creating a reduced-form version of the entire dataset from each bootstrap sample) and then combine the estimates across the samples. Here's the arXiv paper on this topic.

3. Randomized algorithms: there has been a lot of attention recently to algorithms that make use of randomization. E.g., in optimizing a likelihood, you might choose the next step in the optimization based on random subset of the data rather than the full data. Or in a regression

context you might choose a subset of rows of the design matrix (the matrix of covariates) and corresponding observations, weighted based on the statistical leverage [recall the discussion of regression diagnostics in a regression course] of the observations. Here's another arXiv paper that provides some ideas in this area.

6 Hadoop and MapReduce

6.1 Overview

A basic paradigm for working with big datasets is the *MapReduce* paradigm. The basic idea is to store the data in a distributed fashion across multiple nodes and try to do the computation in pieces on the data on each node. Results can also be stored in a distributed fashion.

The basic steps of *MapReduce* are as follows:

- read individual data objects (e.g., records/lines from CSVs or individual data files)
- map: create key-value pairs using the inputs (more formally, the map step takes a key-value pair and returns a new key-value pair)
- reduce - for each key, do an operation on the associated values and create a result - i.e., aggregate within the values assigned to each key
- write out the {key,result} pair

A similar paradigm that is being implemented in some R packages by Hadley Wickham is the split-apply-combine strategy (http://www.jstatsoft.org/v40/i01/paper).

Hadoop is an infrastructure for enabling MapReduce across a network of machines. The basic idea is to hide the complexity of distributing the calculations and collecting results. Hadoop includes a file system for distributed storage (HDFS), where each piece of information is stored redundantly (on multiple machines). Calculations can then be done in a parallel fashion, often on data in place on each machine thereby limiting the amount of communication that has to be done over the network. Hadoop also monitors completion of tasks and if a node fails, it will redo the relevant tasks on another node. Hadoop is based on Java but there are projects that allow R to interact with Hadoop, in particular *RHadoop* and *RHipe*. Rhadoop provides the *rmr*, *rhdfs*, and *rhbase* packages. For more details on RHadoop see Adler and http://blog.revolutionanalytics.com/2011/09/mapreduce-hadoop-r.html.

Setting up a Hadoop cluster can be tricky. We have a basic test setup on the SCF (statistics.berkeley.edu/computing/hadoop), but it's not meant for serious calculations. Hopefully if

you're in a position to need to use Hadoop, it will be set up for you and you will be interacting with it as a user/data analyst.

6.2 MapReduce and RHadoop

Let's see some examples of the MapReduce approach using R syntax of the sort one would use with *RHadoop*. While we'll use R syntax in some cases, but the basic idea of what the map and reduce functions are is not specific to R. Note that using Hadoop with R may be rather slower than actually writing Java code for Hadoop.

First, let's consider a basic word-counting example. Suppose we have many, many individual text documents distributed as individual files in the HDFS. Here's pseudo code from Wikipedia. Here in the map function, the input {key,value} pair is the name of a document and the words in the document and the output {key, value} pairs are each word and the value 1. Then the reduce function takes each key (i.e., each word) and counts up the number of ones. The output {key, value} pair from the reduce step is the word and the count for that word.

```
function map(String name, String document):
// name (key): document name
// document (value): document contents
   for each word w in document:
      return (w, 1)

function reduce(String word, Iterator partialCounts):
// word (key): a word
// partialCounts (values): a list of aggregated partial counts
sum = 0
for each pc in partialCounts:
   sum += pc
return (word, sum)
```

Now let's consider an example where we calculate mean and standard deviation for the income of individuals in each state. Assume we have a large collection of CSVs, with each row containing information on an individual. *mapreduce()* and *keyval()* are functions in the RHadoop package. I'll assume we've written a separate helper function, *my_readline()*, that manipulates individual lines from the CSVs.

```
library(rmr)

map <- function(k, v) {
   record <- my_readline(v)
   key <- record[['state']]
   value <- record[['income']]
   keyval(key, value)
}

reduce <- function(k, v){
   keyval(k, c(length(v), mean(v), sd(v)))
}

incomeResults <- mapreduce(
   input = "incomeData",
   map = mymap,
   reduce = myreduce,
   combine = NULL,
   input.format = 'csv',
   output.format = 'csv')

from.dfs(incomeResults, format = 'csv', structured = TRUE)
```

A few additional comments. In our map function, we could exclude values or transform them in some way, including producing multiple records from a single record. And in our reduce function, we can do more complicated analysis. So one can actually do fairly sophisticated things within what may seem like a restrictive paradigm. But we are constrained such that in the map step, each record needs to be treated independently and in the reduce step each key needs to be treated independently. This allows for the parallelization.