Unit 7: Databases and Big Data

October 6, 2015

References:

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

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

Note that for a lot of the demo code I ran the code separately outside of *knitr* and this document because of the time involved in working with large datasets.

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 provide 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. Finally, having a big dataset also allows you to do a large number of statistical analyses and tests, so multiple testing is a big issue. With enough analyses, something will look interesting just by chance in the noise of the data, even if there is no underlying reality to it.

Here's a different way to summarize it.

Different people define the 'big' in big data differently. One definition involves the actual size of the data. 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. Another definition of 'big data' has more to do with how pervasive data and empirical analyses backed by data are in society and not necessarily how large the actual dataset size is.

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.

And don't forget simple things. If you have a dataset with 30 columns that takes up 10 Gb but you only need 5 of the columns, get rid of the rest and work with the smaller dataset. Or you might be able to get the same information from a random sample of your large dataset as you would from doing the analysis on the full dataset. Strategies like this will often allow you to stick with the tools you already know.

Also, the example datasets in Section 3 are not good illustrations of this, but as we'll see scattered throughout the Unit and as we saw in Unit 3, 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.

```r
library(RSQLite)

## Loading required package:  DBI

# fileName <- "/mirror/data/pub/html/scf/cis.db"
fileName <- "/tmp/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 <- a_ids[ , 1]
a_ids

## [1]  532 1141

query <- paste("select id_title from authorships where author_id in (",
               paste(a_ids, collapse = ","), ")")
query

## [1] "select id_title from authorships where author_id in ( 532,1141 )"

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

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

```r
t_ids <- t_ids[ , 1]
query <- paste("select * from articles where id_title in (",
               paste(t_ids, collapse = ","), ")")
titles <- dbGetQuery(db, query)
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
##                                                                        ti
## 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 cha
## 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_ic
## 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     1008
##   issue_id zmath
## 1       74    \n
## 2      106    \n
## 3      106    \n
## 4      106    \n
## 5      146    \n
## 6      162    \n
```

```r
# 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
   authors.name like 'Breiman%' and authors.id = authorships.author_id and
   authorships.id_title = articles.id_title")
# "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
##                                                                     ti
## 1 The individual ergodic theorem of information theory (Corr: V31 p809-81
## 2             The capacities of certain channel classes under random cod
## 3                                      On the completeness of order statist
## 4              The strong law of large numbers for a class of Markov cha
## 5                                   The Poisson tendency in traffic distort
## 6                                       Consistent estimates and zero-one se
##   year volume number page_start page_end url journal journal_id volume_i
## 1 1957     28      0        809      811                   1748      799
## 2 1960     31      0        558      567                   1748      911
## 3 1960     31      0        794      797                   1748      911
## 4 1960     31      0        801      803                   1748      911
## 5 1963     34      0        308      311                   1748      986
## 6 1964     35      0        157      161                   1748     1008
##   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.

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

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

fullInfo <- dbSendQuery(db, 'select * from journals join partialArticleInfo
   on journals.id = partialArticleInfo.journal_id')
# '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_ic
## 1 1946     17      0        409      431                  1748      417(
## 2 1946     14      0        137      147                   452      272:
## 3 1947      1      0          7       11                  1746       27:
##   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 4 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
student <- read.csv("student.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