Data Science with R
Dealing with Big Data

Graham.Williams@togaware.com

14th February 2014

Visit http://onepager.togaware.com/ for more OnePageR's.

In this module we explore how to load larger datasets into R and some hints on dealing with the larger data within R.

The required packages for this module include:

```
library(data.table)
```

```
## Error:  there is no package called 'data.table'
```

```
library(plyr)
library(rbenchmark)
```

```
## Error:  there is no package called 'rbenchmark'
```

```
library(sqldf)
```

```
## Error:  there is no package called 'sqldf'
```

As we work through this module, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This present module is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 Reading from CSV

Loading data from a CSV file is one of the simplest ways to load data into R. However, `read.csv()` performs a lot of analysis of the data it is reading, to determine the data types.

We will illustrate this with a relatively small dataset of nearly 70 thousand rows. Even with a small dataset loading takes some time—just over 1 minute here.

```
system.time(ds <- read.csv(file="data/weatherAUS.csv"))
```

```
##    user  system elapsed
##   1.193   0.012   1.222
```

When reading very much larger CSV files the times can be substantial. For a 10GB CSV file, containing 2 million observations of 1000 variables, the time taken to read the CSV is around 30 minutes.

2 Reading from CSV—A Little Help

We can help `read.csv()` quite a bit, and avoid a lot of extra processing and memory usage by telling `read.csv()` the data types of the columns of the CSV file. We do this using `colClasses=`.

Often though we are not all that sure what the classes should be, and there may be many of them, and it would not be reasonable to have to manually specify each one of them. We can get a pretty good idea by reading only a part of the CSV file as to the data types of the various columns.

A common approach is to make use of `nrows=` to read a limited number of rows. From the first 5000 rows, for example, we can extract the classes of the columns as automatically determined by R and then use that information to read the whole dataset.

Note firstly the expected significant reduction in time in reading just the first 5000 rows.

```
system.time(ds <- read.csv(file="data/weatherAUS.csv", nrows=5000))
```

```
##    user  system elapsed
##   0.044   0.000   0.044
```

```
classes <- sapply(ds, class)
classes
```

```
##          Date      Location       MinTemp       MaxTemp      Rainfall
##      "factor"      "factor"     "numeric"     "numeric"     "numeric"
##   Evaporation      Sunshine   WindGustDir WindGustSpeed    WindDir9am
##     "numeric"     "numeric"      "factor"     "integer"      "factor"
....
```

We can check that the data types all look okay. Actually, perhaps we want to ignore the date, which is the first column:

```
classes[1] <- "NULL"
classes
```

```
##          Date      Location       MinTemp       MaxTemp      Rainfall
##        "NULL"      "factor"     "numeric"     "numeric"     "numeric"
##   Evaporation      Sunshine   WindGustDir WindGustSpeed    WindDir9am
##     "numeric"     "numeric"      "factor"     "integer"      "factor"
....
```

Now to read the full dataset without R having to do so much parsing.

```
system.time(ds <- read.csv(file="data/weatherAUS.csv", colClasses=classes))
```

```
##    user  system elapsed
##   0.610   0.016   0.627
```

For a small dataset, as used here for illustration, the timing differences are in fractions of a second. For our 10GB dataset, with 2 million observations and 1000 variables, this approach reduces the read time from 30 minutes to 10 minutes.

3 Checking Object Sizes

Explore the options for listing the memory used for storing data.

```
print(object.size(ds), units="Mb")
```

```
## 8.4 Mb
```

4 Reading a File in Chunks

Demonstrate the approach of loading the data in chunks and piecing those chunks together.

```
ds <- read.csv(...., nrows=2e4)
ds <- rbind(ds, read.csv(...., skip=2e4, nrows=2e4))
ds <- rbind(ds, read.csv(...., skip=4e4, nrows=2e4))
ds <- rbind(ds, read.csv(...., skip=6e4))
```

5 Conversion to a Data Table

Explore the data.table package.

For our usual manpulation of data `ddply()` from plyr (Wickham, 2012) is particularly useful. However, as our datasets increase in size, it begins to struggle. The data.table (?) package uses similar syntax to manipulate data, but also introduces indexing to handle larger datasets more efficiently.

6 Reading from CSV—Fast Reading

Using `fread()` from data.table (?) can even more dramatically reduced load time. For our 10GB dataset, with 2 million observations and 1000 variables the load time is reduced to just 2 minutes.

Demonstrate the use of fread().

7 Interacting with Data Tables

Whereas we index a data frame by the observation (i) and the columns (j), the basic syntax of the data table takes a very different view, and one more familiar to database users. In terms of the familiar SQL syntax, we might think of a data table access as specifying the condition to be met for the observations we wish to extract (a WHERE clause), the columns we wish to extract (a SELECT clause) and how the data is to be aggregated (a GROUP BY clause).

```
ds[WHERE, SELECT, by=GROUPBY]
```

8 Data Preparation using data.table

Replicate the data loading template of the Data one pager, using data.table terminology.

```
# Required packages
library(data.table)   # fread()
library(rattle)       # normVarNames()
library(randomForest) # na.roughfix()

# Data setup
dspath     <- system.file("csv", "weather.csv", package="rattle")
weather    <- fread(dspath)
dsname     <- "weather"
ds         <- get(dsname)

class(ds)
setnames(ds, names(ds), normVarNames(names(ds))) # Optional lower case names.

vars       <- names(ds)
target     <- "rain_tomorrow"
risk       <- "risk_mm"
id         <- c("date", "location")


# Variables to ignore
mvc        <- ds[,lapply(.SD, function(x) sum(is.na(x)))]
mvn        <- names(ds)[which(mvc == nrow(ds))]
ignore     <- union(ignore, mvn)
vars       <- setdiff(vars, ignore)

# Observations to omit
omit       <- NULL
omit       <- union(omit, which(is.na(ds[,target]))  # Remove all with a missing target.

# SOME WAY TO GET THIS TO WORK?

ds[vars]   <- na.roughfix(ds[vars])   # Optional impute missing values.
mo         <- attr(na.omit(ds[vars]), "na.action")
omit       <- union(omit, mo)
if (length(omit)) ds <- ds[-omit,]    # Optional remove ommited observations.

# Finalise TO FIX FROM HERE
ds[target] <- as.factor(ds[[target]])
inputs     <- setdiff(vars, target)
nobs       <- nrow(ds)
numerics   <- intersect(inputs, vars[which(sapply(ds[vars], is.numeric))])
categorics <- intersect(inputs, vars[which(sapply(ds[vars], is.factor))])
```

```
# Save the data
dsdate     <- paste0("_", format(Sys.Date(), "%y%m%d"))
dsrdata    <- paste0(dsname, dsdate, ".RData")
save(ds, dsname, dspath, dsdate, target, risk, id, ignore, vars,
     nobs, omit, inputs, numerics, categorics, file=dsrdata)
```

9 Aggregate Big Data—Benchmark Datasets

We can compare a number of methods for aggregating data. We will use a small dataset and then a larger (if not big) dataset to illustrate some of the issues. We aggregate the data to determine the mean of a collection of integers, grouped by a factor.

For the benchmark we will make use of a variety of packages including rbenchmark (?), data.table (?), plyr (Wickham, 2012), and sqldf (?).

```
library(rbenchmark)
```

```
## Error:  there is no package called 'rbenchmark'
```

```
library(data.table)
```

```
## Error:  there is no package called 'data.table'
```

```
library(plyr)
library(sqldf)
```

```
## Error:  there is no package called 'sqldf'
```

Two random datasets will be used: small (dss) consisting of just 100 rows and large (dsl) consisting of 10 million rows. They each have two columns, `x` and `g`. The data is to be grouped by `g` and calculate the `mean` of `x`.

```
dss <- data.frame(x=sample(1:10, 1e2, replace=TRUE),
                  g=sample(c('a', 'b', 'c', 'd'), 1e2, replace=TRUE))
head(dss)
```

```
##   x g
## 1 3 c
## 2 4 d
## 3 1 a
....
```

```
dts <- data.table(dss, key="g")
```

```
## Error:  could not find function "data.table"
```

```
dsl <- data.frame(x=sample(1:10, 1e7, replace=TRUE),
                  g=sample(c('a', 'b', 'c', 'd'), 1e7, replace=TRUE))
head(dsl)
```

```
##    x g
## 1  7 b
## 2 10 b
## 3  4 a
....
```

```
system.time(dtl <- data.table(dsl, key="g"))
```

```
## Error:  could not find function "data.table"
```

```
## Timing stopped at: 0.001 0 0
```

The benchmark methodology in the following pages was inspired by a posting on Stack Overflow,

with a question asked by John Horton (25 May 2012), and answered by Chase (25 May 2012, 22 June 2012) and Tyler Rinker (25 May 2012).

10 Aggregate Big Data—Apply Functions

Our benchmark will compare several methods for aggregating data. Here we define a number of alternative approaches to grouping by using the apply type functions.

```
ds <- dss
dt <- dts
```

```
## Error:  object 'dts' not found
```

```
b.tapply <- function() unlist(tapply(ds$x, ds$g, mean))
b.tapply()
```

```
##     a     b     c     d
## 4.800 5.500 5.467 5.115
```

```
b.lapply <- function() unlist(lapply(split(ds$x, ds$g), mean))
b.lapply()
```

```
##     a     b     c     d
## 4.800 5.500 5.467 5.115
```

```
b.ddply  <- function() ddply(ds, .(g), transform, y=mean(x))
b.ddply()
```

```
##      x g     y
## 1    1 a 4.800
## 2    6 a 4.800
## 3    9 a 4.800
....
```

Note that `sapply` is a user-friendly wrapper for `lapply`, and so will deliver essentially the same times.

11 Aggregate Big Data—Aggregate Functions

Here we define a number of approaches using group-by type functions.

```
b.ave    <- function() ave(ds$x, ds$g, FUN=mean)
b.ave()
```

```
##   [1] 5.467 5.115 4.800 4.800 5.115 5.115 5.500 4.800 5.467 5.115 5.500
##  [12] 5.467 5.115 5.500 5.467 5.467 5.115 5.115 4.800 5.467 5.115 5.500
##  [23] 5.500 5.467 5.115 4.800 5.115 5.467 5.500 5.115 5.115 5.467 5.500
##  [34] 5.115 4.800 5.467 5.115 4.800 5.500 5.467 4.800 5.467 5.500 4.800
....
```

```
b.by     <- function() by(ds$x, ds$g, mean)
b.by()
```

```
## ds$g: a
## [1] 4.8
## ------------------------------------------------------------
## ds$g: b
....
```

```
b.aggr   <- function() aggregate(ds$x, list(ds$g), mean)
b.aggr()
```

```
##   Group.1     x
## 1       a 4.800
## 2       b 5.500
## 3       c 5.467
....
```

```
b.sqldf  <- function() sqldf("select g, avg(x) from ds group by g")
b.sqldf()
```

```
## Error:  could not find function "sqldf"
```

```
b.dt     <- function() dt[, mean(x), by=g]
b.dt()
```

```
## Error:  object 'x' not found
```

12 Aggregate Big Data—Benchmarks

First we run the benchmark on the smaller dataset.

```
ds <- dss
dt <- dts
```

```
## Error:  object 'dts' not found
```

```
benchmark(b.ddply(), b.tapply(), b.lapply(),
          b.ave(), b.by(), b.aggr(), b.sqldf(), b.dt(),
          replications = 10,
          columns = c("test", "elapsed", "relative"),
order = "elapsed")
```

```
## Error:  could not find function "benchmark"
```

Now we run the benchmark on the larger dataset.

```
ds <- dsl
dt <- dtl
```

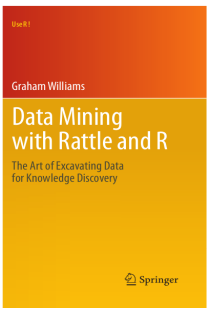
```
## Error:  object 'dtl' not found
```

```
benchmark(b.ddply(), b.tapply(), b.lapply(),
          b.ave(), b.by(), b.aggr(), b.sqldf(), b.dt(),
           replications = 2,
           columns = c("test", "elapsed", "relative"),
           order = "elapsed")
```

```
## Error:  could not find function "benchmark"
```

13 Further Reading

The Rattle Book, published by Springer, provides a comprehensive introduction data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from `http://datamining.togaware.com`, including the Datamining Desktop Survival Guide.



This module is one of many OnePageR modules available from `http://onepager.togaware.com`. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.

14 References

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Wickham H (2012). *plyr: Tools for splitting, applying and combining data*. R package version 1.8, URL http://CRAN.R-project.org/package=plyr.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.

This document, sourced from BigDataO.Rnw revision 282, was processed by KnitR version 1.5 of 2013-09-28 and took 5.3 seconds to process. It was generated by gjw on nyx running Ubuntu 13.10 with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-02-14 06:13:19.