Reading Genomics Data into R/Bioconductor

Aedín Culhane

May 16, 2012

Contents

1 Reading in Excel, csv and plain text files

Common data exchange formats are Excel, comma and tab-delimited format text files. Each of these files will be provided on the course website. Or to create a tab-delimited and csv file, do the following:

1. Download the data set "Women.xls" from the course website. Save it in your local directory.

2. Open this file "Women.xls" in Excel.

3. To export data as comma or tab delimited text files. In Excel select File -> Save as and
   Tab: select the format Text (Tab delimited) (*.txt).
   CSV: select the format CSV (Comma delimited) (*.csv).

2 Importing and reading data into R

RStudio has a nice user interface to reading in file. Click on Workspace -> Import Dataset. We will use this first to read in each of these files. Now lets learn the commands to read in different file types.

1. Using read.table()

   (a) The most commonly used function for reading data is `read.table()`. It will read the data into R as a *data.frame*.

   By Default `read.table()` assumes a file is space delimited and it will fail if the file is in a different format with the error below.

   ```
   Women<-read.table("Women.txt")
   ```

   In order to read files that are tab or comma delimited, the defaults must be changed. We also need to specify that the table has a header row

   ```
   >   # Tab Delimited
   > Women<-read.table("Women.txt", sep="\t", header=TRUE)
   > Women[1:2,]
     height weight age
   1     58    115  33
   2     59    117  34
   > summary(Women)
        height          weight           age
    Min.   :58.0   Min.   :115.0   Min.   :30.00
    1st Qu.:61.5   1st Qu.:124.5   1st Qu.:32.00
    Median :65.0   Median :135.0   Median :34.00
    Mean   :65.0   Mean   :136.7   Mean   :33.93
    3rd Qu.:68.5   3rd Qu.:148.0   3rd Qu.:35.50
    Max.   :72.0   Max.   :164.0   Max.   :39.00
   > class(Women$age)
   [1] "integer"
   ```

   Note by default, character vector (strings) are read in as factors. To turn this off, use the parameter as.is=TRUE

   (b) Important options:

header==TRUE	should be set to 'TRUE', if your file contains the column names
as.is==TRUE	otherwise the character columns will be read as factors
sep=""	field separator character (often comma ',' or tab "" eg: sep=",")
na.strings	a vector of strings which are to be interpreted as 'NA' values.
row.names	The column which contains the row names
comment.char	by default, this is the pound # symbol, use "" to turn off interpretation of commented text.

```
>   # Read the help file
>   help(read.table)
```

Note the defaults for `read.table()`, `read.csv()`, `read.delim()` are different. For example, in `read.table()` function, we specify `header=TRUE`, as the first line is a line of headings among other parameters.

2. `read.csv()` is a derivative of `read.table()` which calls `read.table()` function with the following options so it reads a comma separated file:

```
read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, comment.char="", ...)
```

Read in a comma separated file:

```
> # Comma Delimited
> Women2<-read.csv("Women.csv", header=TRUE)
> Women2[1:2,]

  height weight age
1     58    115  33
2     59    117  34
```

3. Reading directly from Website You can read a file directly from the web

```
>   myURL<-"http://bcb.dfci.harvard.edu/~aedin/courses/Bioconductor/Women.txt"
> read.table(myURL, header=TRUE)[1:2,]

  height weight age
1     58    115  33
2     59    117  34
```

4. Using scan()

   NOTE: `read.table()` is not the right tool for reading large matrices, especially those with many columns. It is designed to read 'data frames' which may have columns of very different classes. Use `scan()` instead.

   `scan()` is an older version of data reading facility. Not as flexible, and not as user-friendly as `read.table()`, but useful for Monte Carlo simulations for instance. `scan()` reads data into a *vector* or a *list* from a file.

```
> myFile <- "outfile.txt"
> # Create a file
> cat("Some data", "1 5 3.4 8", "9 11 23", file=myFile,sep="\n")
> exampleScan <- scan(myFile, skip = 1)
> print(exampleScan)

[1]  1.0  5.0  3.4  8.0  9.0 11.0 23.0
```

   Note by default `scan()` expects numeric data, if the data contains text, either specify what="text" or give an example what="some text".

   Other useful parameters in `scan()` are nmax (number of lines to be read) or n (number of items to be read.

```
> scan(myFile, what="some text", n=3)

[1] "Some" "data" "1"
```

5. Reading data from an Excel file into R

   There are several packages and functions for reading Excel data into R, however I normally export data as a .csv file and use `read.table`. See below. However if you wish to directly load Excel data, here are the options available to you. See the section "Importing-from-other-statistical-systems" in `http://cran.r-project.org/doc/manuals/R-data.html` for more information

6. Import/Export from other statistical software

   To read binary data files written by statistical software other than `R` such as Epi-Info, Minitab, S-PLUS, SAS, SPSS,Stata and Systat, R recommends using the R package *foreign*. Details can be found in the `R` manual: R data Import /Export.

7. Other considerations when reading or writing data

   It is often useful to create a variable with the path to the data directory, particular if we need to read and/or write more than one dataset. NOTE: use double backslashes to specify the path names, or the forward slash can be used.

```
> myPath <- file.path('C://Project1')
> file.exists(myPath)

[1] FALSE

> #Set myPath to be current directory
> myPath<-file.path(getwd())
>
```

   It is better to expand a path using `file.path()` rather than `paste()` as `file.path()` will expand the path with delimiting characters appropriate to the operating system in use (eg / unix, \, windows etc)

```
> myfile<-file.path(myPath, "Women.txt")
```

   Use `file.exists()` to test if a file can be found. This is very useful. For example, use this to test if a file exists, and if TRUE read the file or you could ask the R to warn or stop a script if the file does not exist

```
> if (!file.exists(myfile)) {
+   print(paste(myfile, "cannot be found"))
+   }else{
+   Women<- read.table(myfile, sep="\t", header=TRUE)
+   Women[1:2,]
+ }
```

```
  height weight age
1     58    115  33
2     59    117  34
```

3 Reading Genomics Data into R

We will mention some function for reading in your own data during the excerise, but as these are large data files, in general each platform and genomics data type has its own unqiue set of functions which are optimzied to read that data efficiently. On the bioconductor website, there are several workflows presneted for different types of data `http://bioconductor.org/help/workflows/`

1. The function `ReadAffy` can be used to read Affymetrix celfiles. we will talk about this more during class.

   ```
   > abatch<-ReadAffy(celfile.path="cels")
   > eset2<-rma(abatch)
   ```

   ReadAffy creates an AffyBatch object, and rma will create an ExpressionSet object

2. Limma and its graphical user interface LimmaGui can read most two color arrays including Agilent. The countway holds a course on LimmaGUI and it will not be covered in depth in this class. Agilent 4x44 array data produced by the Agilent Feature Extraction (AFE) image analysis software can be read by the package *Agi4x44PreProcess*. The *AgiMicroRna* is designed for Agilent microRNA. The basic limma functions for reading in data are:

   ```
   > library(limma)
   > targets <- readTargets("targets.txt")
   > RG <- read.maimages(targets, source="genepix")
   > MA <- normalizeWithinArrays(RG)
   ```

   The targets file contains information about the samples. The images list in the targets file are read using read.maimages.

3. Illumina arrays can be read using the lumni package using the `lumiR` or `lumiR.batch` The lumi package has one major class: LumiBatch, which is inherited from ExpressionSet class for better compatibility

4. SNP data can be read using the Oligo or Aroma packages which will be covered by Nicolai

5. RNA-seq data is generally processing outside of R/Bioconductor into count data which is read into Bioconductor using the package *ShortRead*. The vignettes for *edgeR* and *DEseq* have detailed tutorially. We will cover this on Thursday

4 Getting Data from Gene Expression Omnibus (GEO) or ArrayExpress database.

Another source of data is published data. The Gene Expression Omnibus (GEO) from the NCBI is a public repository of genomic data from microarrays (gene expression, ChIP-on-chip, ArrayCGH, SNP, protein,...) and high-throughput DNA and RNA sequencing (SAGE, MPSS, NGS,...). It is structured as follows:

1. Platform which is assigned a unique GEO accession number of type GPLxxx.
2. Sample an individual cell or tissue extract which is assigned a unique GEO accession number of type GSMxxx.
3. Dataset A group of samples that were obtained on the same platform in a single study which which is assigned a unique GEO accession number of type GDSxxx.
4. Series: One or more datasets often on the same set of samples form a series which is assigned a unique GEO accession number of type GSExxx. A series (GSE) may contain >1 GDS if measurements were obtained on different platforms (eg Affymetrix U133a and U133b) or different technologies (eg SNP and Gene expression)

for more information see `http://bioconductor.org/packages/release/bioc/html/GEOquery.html`

The packages GEOquery and ArrayExpress can be used to extract data from GEO and AE respectively.

```
> library(GEOquery)
> ?getGEO
> gds <- getGEO("GDS507")
> eset <- GDS2eSet(gds,do.log2=TRUE)
> eset

> gse2553 <- getGEO('GSE2553',GSEMatrix=TRUE)
> gse2553[[1]]
```

AE sometimes takes a longer time as we are in the USA

```
> library("ArrayExpress")
> AEset = ArrayExpress("E-MEXP-1416")
```

The functions `GDS2MA` and `GDS2eSet` can be used to convert a GDS file to a expressionSet. If getGEO retrieves a GSE it will be a list by default as a GSE can contain >1 dataset.

The package *GEOmetadb* compiled by Sean Davis is a database of GEO meta data compiled for Bioconductor. It can be queried using SQL statements.

5 Writing Data

1. Function `sink()` diverts the output from the console to an external file

```
> sink(file.path(myPath, "sinkTest.txt"))
> print("This is a test of sink")
> ls()
> sin(1.5*pi)
> print(1:10)
> sink()
```

2. Writing a data matrix or data.frame using the `write.table()` function `write.table()` has similiar arguments to `read.table()`

```
> myResults <- matrix(rnorm(100,mean=2), nrow=20)
> write.table(myResults, file='results.txt')
```

This will write out a space separated file.

```
> df1 <- data.frame(myResults)
> colnames(df1) <- paste("MyVar", 1:5, sep="")
> write.table(df1, file="results2.txt", row.names=FALSE, col.names=TRUE)
> read.table(file="results2.txt", head=TRUE)[1:2,]

    MyVar1   MyVar2    MyVar3   MyVar4   MyVar5
1 1.369315 2.176032 0.5793544 2.005300 3.761120
2 2.166460 1.516577 2.3495776 2.133439 2.836146
```

3. Important options

append = FALSE	create new file
sep = " "	separator (other useful possibility sep=",")
row.names = TRUE	may need to change to row.names=FALSE
col.names = TRUE	column header

4. Output to a webpage

The package R2HTML will output R objects to a webpage

```
> # Write data directly to a new webpage
> library(R2HTML)
> HTML(df1,outdir=myPath, file="results.html")
> # Capture output to a webpage
```

```
>
> print("Capturing Output")
> df1[1:2,]
> summary(df1)
> print("hello and Goodbye")
> HTMLStop()
>
```

5.1 Output to tex

```
> require(xtable)
> x <- rnorm(100)
> y <- 4 + 3 * x + rnorm(100, 0, 2)
> lmodel <- lm(y ~ x)
> lan <- anova(lmodel)
> xtable(lmodel, caption = "Example of table from Linear regression model output
+ floating = FALSE, label = "tab:coef")
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	3.8398	0.2181	17.60	0.0000
x	3.0740	0.2368	12.98	0.0000

Table 1: Example of table from Linear regression model output.

6 Exercise

We will examine the GEO dataset "GDS810" which is Affymetrix Human Genome U133A Array data contains data on Alzheimer's disease at various stages of severity into R.

1. Use the getGEO function to read it into R. What class is this object. Use str to view the data
2. Then Convert it to an ExpressionSet object. What class is the object
3. How many arrays are in this dataset? How many features (probesets) are on this arrays
4. How many severe cases of disease are there?
5. Create an ExpressionSet with just the severe cases of disease
6. How would I write these data to excel (hint have a look at the help documentation for ExpressionSet. Look at the section labelled "Methods"