Introduction to R

Nishant Gopalakrishnan, Martin Morgan

9 December, 2010

Contents

List of Figures

1 Introduction

Microarray expression data from 128 individuals with acute lymphoblastic leukemia has been provided as a comma separated file `exprsMat.csv`. Several covariates such as age, sex, type, stage of the disease etc. have been provided as another comma separated file `pData.csv`. Our goal in this exercise is to

- Read in the expression data and the covariates into R objects.
- Perform data manipulations such as subsetting to arrive at a smaller expression dataset with samples that we are interested in.
- Generate MA plots from our subsetted data using the lattice package.

2 Load Microarray Expression data into an Rsession

We shall proceed to load the data for microarray expression values and the covariates from the supplied csv files into an R session. The expression data is contained in the file `exprsMat.csv` and the covariates are in the file `pData.csv`. These files are located in the `extdata` folder of the SeattleIntro2010 package.

A convenient way to get to the path of the files is to make use of the `system.file` function after loading the SeattleIntro2010 package using the `library` function.

R provides several functions such as `read.table` for reading in meta data files into a `data.frame` with appropriate column and row names from the header information provided in the file. Convenience functions such as `count.fields` are also available for discovering problems in files (such as certain rows in the file having different number of fields) when using the `read.table` function.

Exercise 1

- *Use the `system.file` to locate the path to the files `exprsMat.csv` and `pData.csv`*
- *Make use of the `count.fields` function on the `pData.csv` file to ensure that the file has the same number of fields in each line of the file.*
- *Use the `read.table` function to read in the experimental meta data into an R variable `pdOrig`.*
- *Observe the effect, the `check.names` argument of `read.table` set to FALSE has on reading in the column names of the variable `pdOrig`. The function `colnames` can be used to extract the column names of the `pdOrig`. Hint: Observe the column name `pmol biol`.*
- *Use the `read.table` function to read in the expression matrix into a variable `expOrig` with the argument `check.names=FALSE`.*

```
> library(SeattleIntro2010)
> phenoPath <- system.file( "extdata", "pData.csv", package="SeattleIntro2010")
> pdOrig <- read.table(phenoPath, check.names = FALSE)
> colnames(pdOrig)
```

```
 [1] "cod"             "diagnosis"       "sex"
 [4] "age"             "BT"              "remission"
 [7] "CR"              "date.cr"         "t(4;11)"
[10] "t(9;22)"         "cyto.normal"     "citog"
[13] "mol biol"        "fusion protein" "mdr"
[16] "kinet"           "ccr"             "relapse"
[19] "transplant"      "f.u"             "date last seen"
```

```
> expPath <- system.file("extdata", "exprsMat.csv", package="SeattleIntro2010")
> expOrig <- read.table(expPath, check.names = FALSE)
```

3 Subset data

The `expOrig` variable is a `data.frame` that contains the expression intensities for the experiment. The column names of the `data.frame` represent the sample Id's and the row names represent the genes that are being studied. A convenient function to look at just the first couple of entries in large `data.frame` is to use the `head` function.

The `pdOrig` variable is a `data.frame` with columns representing the various covariates that describe the experiment (age, sex, etc.). The row names correspond to the sample Id's. The column BT is a factor indicating the tumour cell type (B1, B2, T1 ,T2 etc. with B indicating B-cell type and T indicating T-cell type). Similarly the column `mol biol` is a factor indicating the molecular biology of the cancer. (BCR/ABL, NEG, E2A/PBX1 etc.) The `mol biol` column can be accessed using `pdOrig[["mol biol"]]`.

In this section, we shall try to make use of some of concepts of indexing, subsetting, factors etc. that we discussed earlier to split our `expOrig` and `pdOrig` variables to narrow down to the samples that we are interested in. We are specifically interested in B-cell tumours with molecular biology type "NEG" or "BCR/ABL". We proceed to subset our `expOrig` and `pdOrig` variables to narrow down to our samples of interest.

Exercise 2

- *Use the `grep` function on the `BT` column in the `pdOrig` `data.frame` to identify the B cell tumours.*
- *Identify the samples that are "NEG" or "BCR/ABL" molecular biology type using the column `mol biol`, the `%in%` function and the `which` functions in R.*
- *Identify samples that are both B cell tumours and are "BCR/ABL" or "NEG" using the `intersect` function on the indices that we have previously computed.*
- *Subset the meta data variable `pdOrig` to create a new variable `psubData`. Note:The rows of the data.frame represent samples.(Subset along rows)*
- *Subset the expression matrix `expOrig` to create a new `data.frame` `exprsMat` with only the samples with the cell and the tumour types we interested in. Note:The columns of the expression matrix represents samples.(Subset along columns)*

```
> bcell <- grep("^B", as.character(pdOrig$BT))
> types <- c("NEG", "BCR/ABL")
> moltyp <- which(as.character(pdOrig[["mol biol"]]) %in% types)
> indx <- intersect(bcell, moltyp)
> psubData <- pdOrig[indx,]
> exprsMat <- expOrig[, indx]
```

4 Recode the factor levels

The covariate data for some variables, for example `BT` is represented using a variable of type factor with distinct levels. These variables can only take a distinct number of categorical values. The levels can be obtained by using the `levels` function on the factor variable. Operations such as subsetting on a factor variable subsets the factor but leaves the levels of the variable unchanged. In this exercise, we shall take a look at the levels of the factor variables that we have just subsetted and attempt to correct this problem.

Exercise 3

- *Observe the levels for the* `mol biol` *and* `moltyp` *variables Do you notice any problem ?.*
- *Recode the factor levels for the* `mol biol` *and moltyp variables using the* `factor` *function.*

```
> psubData$BT <-  factor(psubData$BT)
> levels(psubData$BT)

[1] "B"  "B1" "B2" "B3" "B4"

> psubData[["mol biol"]] <- factor(psubData[["mol biol"]])
> levels(psubData[["mol biol"]])

[1] "BCR/ABL" "NEG"
```

5 Compute summary statistics

R includes several functions that allows you to do a lot while writing only few lines of code. A good example is the `aggregate` function that splits the data into subsets, computes summary statistics for each section and returns the result in a convenient form. For more details regarding this function, please type in `help("aggregate")` into an R session. We will be making use of the `formula` and the `data.frame` methods for the `aggregate` function in this example. Another useful function for creating contingency tables that we will be using is the `xtabs` function.

We proceed to create some summary statistics on the meta data variable `psubData` using the `aggregate` and `xtabs` functions.

Exercise 4

- *Find the average age of male and females in our subsetted metadata variable* `psubData` *for the NEG and BCR/ABL groups using the* `data.frame` *interface for the* `aggregate` *function. The table generated by using the* `aggregate` *should look similar to the one found below. Hint: Try passing* `na.rm = TRUE` *to the aggregate function*

	sex	molBiol	age
1	F	BCR/ABL	39.94
2	M	BCR/ABL	40.50
3	F	NEG	29.75
4	M	NEG	24.86

- *Recalculate the average age of male and females in our subsetted metadata variable* `psubData` *for the NEG and BCR/ABL groups, this time using the* `formula` *interface for the* `aggregate` *function. Make sure that the results are identical to the one from the previous step.*
- *The column* `relapse` *in* `psubData` *is a logical vector indicating whether the patient had a relapse or not. Create a contingency table of the number of subjects that have had a relapse for the samples included in* `psubData` *using the* `xtabs` *function and the covariates* `relapse`, `mol biol` *and* `sex`*. The table generated by using the* `xtabs` *function should look similar to the one found below.*

	BCR/ABL	NEG
F	7.00	3.00
M	9.00	18.00

```
> aggregate( psubData[, "age", drop = FALSE], by= list("sex"= psubData$sex,
+              "molBiol"= psubData[["mol biol"]]),  FUN = mean,
+         na.rm = TRUE )

  sex molBiol      age
1   F BCR/ABL 39.93750
2   M BCR/ABL 40.50000
3   F     NEG 29.75000
4   M     NEG 24.85714

> aggregate(age ~ sex + `mol biol`, data = psubData, FUN = mean)

  sex mol biol      age
1   F  BCR/ABL 39.93750
2   M  BCR/ABL 40.50000
3   F      NEG 29.75000
4   M      NEG 24.85714

> xtabs( relapse ~ sex + `mol biol`, data = psubData)

   mol biol
sex BCR/ABL NEG
```

```
  F        7   3
  M        9  18
```

6 Data visualization

The lattice package implements the Trellis graphics system and provides high-level functions for visualization of multivariate data. A typical call to the lattice function `xyplot` is shown below.

```
> xyplot(y ~ x | c, data, groups = g)
```

The arguments to a lattice function can be summarized in terms of

1. lattice function: A lattice plotting function such as `xyplot`, `dotplot` etc.
2. formula: The first argument to a lattice method is a formula. The formula for our example is `y ~ x | c`. If the lattice method takes only a single vector as input, the formula can be expressed as `~ x | c`.
   - primary variables: Variables y (Y axis of the plot) and x (X axis of the plot) that defines the lattice display separated by the `~` character.
   - conditioning variable: Variable c in the example separated from the primary variables by the character `|`. The conditioning variable divides the plot into separate panels.
3. grouping variable: The variable g in the example. The grouping variable segregates data into subgroups within each panel.
4. data: A *data.frame* with column names corresponding to the variables y, x, c and g.

We proceed to create M-A plots for the first 20 samples in our subsetted expression intensity data.frame `exprsMat`, using the formula provided below and the `xyplot` function from the lattice package.

$$A = rowMeans(m) \tag{1}$$

$$M_i = m_i - A \tag{2}$$

where

- m is the expression `data.frame`.
- i is from 1 to number of samples in the expression `data.frame`
- M is variable for the Y axis of the M-A plot
- Mi is the column for sample i

- A is the variable for the X axis of the M-A plot

Exercise 5

- *Create a new variable* `mat` *by subsetting the* `exprsMat` *variable to include only the first 20 samples. Using the* `rowMeans` *function, create the variable* `A`
- *Create a variable* `M` *using the equation provided above*
- *Create a* `data.frame` `df` *with variables M, A and the sample names in the expression matrix. The column names of the data.frame should be "M", "A" and "Sample" respectively.*
- *Use the* `xyplot` *function from the lattice package to plot the variable M on the Y axis and the variable A on the x axis. Make use of the* `Sample` *variable as a conditioning variable to split the plot into sub plots one each for each sample.*

```
> library(lattice)
> mat <- exprsMat[,1:20]
> A <- rowMeans(log2(mat))
> M <- log2(unlist(mat)) - A
> Sample <- rep(colnames(mat), each=nrow(mat))
> df <- data.frame(M, A, Sample, row.names=NULL, check.names = FALSE)
> plt <-  xyplot(M ~ A | Sample, df, panel=panel.smoothScatter)
>
```

7 Session information

- R version 2.12.1 beta (2010-12-07 r53813), `i386-apple-darwin9.8.0`
- Locale: `C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: ALL 1.4.7, AnnotationDbi 1.12.0, Biobase 2.10.0, Biostrings 2.18.2, DBI 0.2-5, DESeq 1.2.1, GO.db 2.4.5, GenomicFeatures 1.2.3, GenomicRanges 1.2.2, IRanges 1.8.6, RCurl 1.4-3, RSQLite 0.9-4, Rsamtools 1.2.1, SeattleIntro2010 0.0.33, ShortRead 1.8.2, akima 0.5-4, biomaRt 2.6.0, bitops 1.0-4.1, genefilter 1.32.0, hgu95av2.db 2.4.5, lattice 0.19-13, locfit 1.5-6, org.Hs.eg.db 2.4.6, org.Sc.sgd.db 2.4.6, rtracklayer 1.10.5
- Loaded via a namespace (and not attached): BSgenome 1.18.2, KernSmooth 2.23-4, RColorBrewer 1.0-2, XML 3.2-0, annotate 1.28.0, geneplotter 1.28.0, grid 2.12.1, hwriter 1.3, splines 2.12.1, survival 2.36-2, tools 2.12.1, xtable 1.5-6

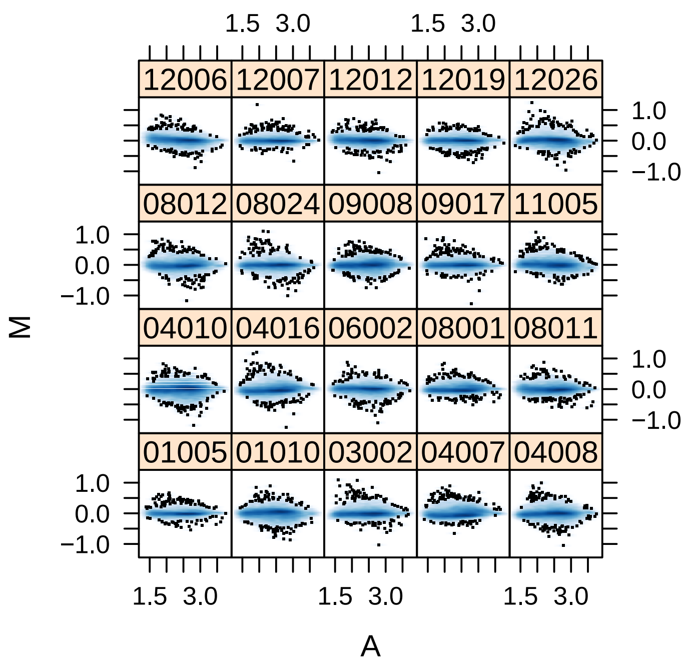

Figure 1: M-A plot for 20 Samples produced using `xyplot`