An introduction to Genomic Data Structures

Cavan Reilly

October 23, 2019

Table of contents

Object Oriented Programming

There has been extensive development of R based tools for managing and storing microarray data.

These approaches have relied heavily on *object oriented* approaches to computing.

As the name implies object oriented approaches are centered on various objects, and these objects have well described features and collections of functions for manipulation and information extraction.

Object Oriented Programming

There are also general high level functions that act in different ways for different types of objects: e.g. `summary`.

There will frequently be operators that have different functionality depending on the object(s) in question: overloaded operators, e.g. + adds numbers and matrices.

The ALL data set

Acute lymphoblastic leukemia (ALL) is a type of cancer.

There are 2 subtypes that we distinguish: a B cell type and a T cell type.

There are multiple types of these too: among the B cell type there are some that have the *Philadelphia chromosome* also called the BCR/ABL mutation.

The Philadelphia chromosome is a genetic abnormality in which a part of chromosome 22 gets switched with part of chromosome 9: this creates a protein that promotes uncontrolled cell division.

There are other known mutations at work in this cancer and some cancers don't have any observed cytogenic abnormality.

The ALL dataset

First we read in some libraries and set up commands to extract those samples that either have the Philadelphia chromosome or no observed cytogenic abnormality.

```
> library(Biobase)
> library(ALL)
> library(genefilter)
> data(ALL)

> bcell=grep("^B",as.character("ALL$BT"))
> types=c("NEG", "BCR/AB")
> moltyp=which(as.character(ALL$mol.biol) %in% types)
> ALL_bcrneg=ALL[,intersect(bcell,moltyp)]
```

The ALL dataset

Then we tidy things up by getting rid of levels that aren't used in our subset of data.

```
> ALL_bcrneg$mol.biol=factor(ALL_bcrneg$mol.biol)
> ALL_bcrneg$BT=factor(ALL_bcrneg$BT)
```

ExpressionSet Objects

If we type the name of the dataset we see that it is a special kind of object.

```
> ALL_bcrneg
ExpressionSet (storageMode: lockedEnvironment)
assayData: 12625 features, 42 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: 01010 04007 ... 68001 (42 total)
  varLabels: cod diagnosis ... date last seen (21 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
  pubMedIds: 14684422 16243790
Annotation: hgu95av2
```

Non-specific filtering

Not all genes are expressed in every cell type.

Some genes display so little variation that there is little hope of finding that such genes differ across groups.

For these reasons, genes are frequently dropped from further consideration during the analysis of microarray data.

There is really no rigorous way to do this, so fix these criteria prior to seeing the data.

If we don't then it will be tempting to try lots of different criteria and then there is concern for type I errors.

Non-specific filtering

Here is an example where we filter out genes based on the interquartile range.

```
> filt_bcrneg=varFilter(ALL_bcrneg, var.func=IQR,
+  var.cutoff=0.5, filterByQuantile=TRUE)

> dim(filt_bcrneg)
Features  Samples
    6312       79
> dim(ALL_bcrneg)
Features  Samples
   12625       79
```

Environments

We often access information in Bioconductor using *environments*.

An environment is like a list but the entries aren't numbered, there are simply key value pairs.

Here we use an environment that maps Affymetrix probe set identifiers to chromosomal locations.

```
> BiocManager::install("hgu95av2.db")
> library("hgu95av2.db")
> hgu95av2MAP$"1001_at"
[1] "1p34-p33"
```

Environments

We can use the `eapply` function like the other apply functions to apply an operation to an environment.

For example, suppose we wanted to find all probes that map to the p arm of chromosome 17.

```
> myPos=eapply(hgu95av2MAP, function(x)
+  grep("^17p",x,value=T))
> myPos=unlist(myPos)
> length(myPos)
```

Environments

One can create one's own environment with the command `new.env`.

Here we create a hash table, which is a common data structure that associates keys with values.

```
> e1=new.env(hash=T)
> e1$a=1:10
> e1$b=2:20
> e1$a
```

ExpressionSet Objects

Bioconductor has taken the approach that microarray experiments need a particular data structure to hold all of the relevant information.

This is unusual insofar as no other features of R are prescriptive about the manner in which you manage and store your data.

While unusual this helps to ensure that data is collected and stored in a format that is compliant with data reporting standards in the microarray analysis literature.

This required format is called MIAME for minimal information about a microarray experiment.

ExpressionSet Objects

We saw an ExpressionSet object above, and there we saw that these objects hold information about

1. `assayData`: this is the expression data
2. `metadata`: this is data about the biological source of the data and the microarray
3. `experimentData`: this is data about the experiment

To take advantage of the features that Bioconductor makes available for microarray analysis, the first task is to manipulate your data into an ExpressionSet object.

You will need the `Biobase` package to have access to these sorts of objects.

ExpressionSet Objects

If you've read your data into R and gotten it into a dataframe called `object`, sometimes you can convert to an ExpressionSet object via the following:

```
> library(convert)
> as(object, "ExpressionSet")
```

If this fails then you need to extract and assign the relevant information from object to the appropriate elements of an ExpressionSet object.

ExpressionSet Objects

Frequently microarray data resides in a tab delimited file where each row corresponds to a "gene" (or a feature in more general terms) and each column corresponds to a sample.

This is the transpose of a data matrix.

As we've seen, such data sets can be read into R using `read.delim` or `read.table`.

For example, here we read in some expression data as a matrix.

```
> dataDirectory=system.file("extdata",package="Biobase")
> exprsFile=file.path(dataDirectory, "exprsData.txt")
> exprs=as.matrix(read.table(exprsFile, header=T,
+  sep="\t",row.names=1, as.is=T))
```

ExpressionSet Objects

Covariate data is essential for analysis of microarray data so ExpressionSet objects collect this information too.

Here we read this data in as a dataframe and check that the row names match the column names of the expression data.

```
> pDataFile=file.path(dataDirectory, "pData.txt")
> pData=read.table(pDataFile, row.names=1, header=T,
+   sep="\t")
> rownames(pData)==colnames(exprs)
```

ExpressionSet Objects

An important component of good data management strategies is to create a *data dictionary* that provides details about the variables in one's data set.

ExpressionSet objects have slots to hold such information: here we will create a dataframe with this information for our example.

```
> metadata=data.frame(labelDescription=c("Patient gender",
  "Case/control status", "Tumor progression on XYZ scale"),
  row.names=c("gender", "type", "score"))
```

Annotated Data Frames

An annotated dataframe is a data structure that allows for easy manipulation of tabular data.

One can be created, and information can be extracted, as follows

```
> adf=new("AnnotatedDataFrame", data=pData,
+  varMetadata=metadata)

> pData(adf)

  gender    type score
A Female Control  0.75
B   Male    Case  0.40
C   Male Control  0.73
D   Male    Case  0.42
...
```

MIAME Objects

One can also directly construct objects to record information for proper annotation of microarray experiments and extract information as follows:

```
> expData=new("MIAME", name="Pierre Fermat", lab="G Lab",
+  contact="pfermat@lab.not.exist", title="Cancer Test",
+  abstract="An example MIAME object",
+  url="www.lab.not.exist",
+  other=list(notes="Created from text files"))
> abstract(expData)
[1] "An example MIAME object"
```

ExpressionSet Objects

We can use a similar approach to create our own ExpressionSet objects:

If we are using a standard microarray we can supply the name of the type of array when we create the ExpressionSet object.

This will allow easy access to information about the probes on the array.

```
> egSet=new("ExpressionSet", exprs=exprs, phenoData=adf,
+   experimentData=expData,
+   annotation="hgu95av2")
```

Note: this will return an error if the components are inadequate, e.g. your experiment data is not a MIAME object.

ExpressionSet Objects

Sometimes one just wants access to some of the functionality of Bioconductor packages without detailed data management-one can create ExpressionSet objects with empty slots.

```
> minSet=new("ExpressionSet", exprs=exprs)
```

ExpressionSet Objects

There are a number of functions that allow one to extract information from ExpressionSet objects:

1. `featureNames`
2. `sampleNames`
3. `varLabels`
4. `exprs`-extracts the matrix of expression vales
5. `phenoData`-extracts the phenotypic data

ExpressionSet Objects

A useful feature of these objects is that one can subset them and the result automatically inherits its class.

```
> males=egSet[,egSet$gender=="Male"]

> males
```