The oligo package

Kasper D. Hansen

Dependencies

This document has the following dependencies:

```
library(oligo)
library(GEOquery)
```

Use the following commands to install these packages in R.

```
source("http://www.bioconductor.org/biocLite.R")
biocLite(c("oligo", "GEOquery))
```

Overview

This document presents the *oligo* package for handling Affymetrix and Nimblegen microarrays, especially gene expression, exon expression and SNP arrays.

Other Resources

- The vignette from the oligo webpage.

Getting the data

We will use the dataset deposited as GEO accession number “GSE38792”. In this dataset, the experimenters profiled fat biopsies from two different conditions: 10 patients with obstructive sleep apnea (OSA) and 8 healthy controls.

The profiling was done using the Affymetrix Human Gene ST 1.0 array.

First we need to get the raw data; this will be a set of binary files in CEL format. There will be one file per sample. The CEL files are accessible as supplementary information from GEO; we get the files using *GEOquery*.

```
library(GEOquery)
getGEOSuppFiles("GSE38792")
list.files("GSE38792")
```

```
## [1] "CEL"               "filelist.txt"     "GSE38792_RAW.tar"
```

```
untar("GSE38792/GSE38792_RAW.tar", exdir = "GSE38792/CEL")
list.files("GSE38792/CEL")
```

```
##  [1] "GSM949164_Control1.CEL.gz" "GSM949166_Control2.CEL.gz"
##  [3] "GSM949168_Control3.CEL.gz" "GSM949169_Control4.CEL.gz"
##  [5] "GSM949170_Control5.CEL.gz" "GSM949171_Control6.CEL.gz"
##  [7] "GSM949172_Control7.CEL.gz" "GSM949173_Control8.CEL.gz"
##  [9] "GSM949174_OSA1.CEL.gz"     "GSM949175_OSA2.CEL.gz"
## [11] "GSM949176_OSA3.CEL.gz"     "GSM949177_OSA4.CEL.gz"
## [13] "GSM949178_OSA5.CEL.gz"     "GSM949179_OSA6.CEL.gz"
## [15] "GSM949180_OSA7.CEL.gz"     "GSM949181_OSA8.CEL.gz"
## [17] "GSM949182_OSA9.CEL.gz"     "GSM949183_OSA10.CEL.gz"
```

oligo and many other packages of its kind has convenience functions for reading in many files at once. In this case we construct a vector of filenames and feed it to `read.celfiles()`.

```
library(oligo)
celfiles <- list.files("GSE38792/CEL", full = TRUE)
rawData <- read.celfiles(celfiles)
```

```
## Reading in : GSE38792/CEL/GSM949164_Control1.CEL.gz
## Reading in : GSE38792/CEL/GSM949166_Control2.CEL.gz
## Reading in : GSE38792/CEL/GSM949168_Control3.CEL.gz
## Reading in : GSE38792/CEL/GSM949169_Control4.CEL.gz
## Reading in : GSE38792/CEL/GSM949170_Control5.CEL.gz
## Reading in : GSE38792/CEL/GSM949171_Control6.CEL.gz
## Reading in : GSE38792/CEL/GSM949172_Control7.CEL.gz
## Reading in : GSE38792/CEL/GSM949173_Control8.CEL.gz
## Reading in : GSE38792/CEL/GSM949174_OSA1.CEL.gz
## Reading in : GSE38792/CEL/GSM949175_OSA2.CEL.gz
## Reading in : GSE38792/CEL/GSM949176_OSA3.CEL.gz
## Reading in : GSE38792/CEL/GSM949177_OSA4.CEL.gz
## Reading in : GSE38792/CEL/GSM949178_OSA5.CEL.gz
## Reading in : GSE38792/CEL/GSM949179_OSA6.CEL.gz
## Reading in : GSE38792/CEL/GSM949180_OSA7.CEL.gz
## Reading in : GSE38792/CEL/GSM949181_OSA8.CEL.gz
## Reading in : GSE38792/CEL/GSM949182_OSA9.CEL.gz
## Reading in : GSE38792/CEL/GSM949183_OSA10.CEL.gz
```

```
rawData
```

```
## GeneFeatureSet (storageMode: lockedEnvironment)
## assayData: 1102500 features, 18 samples
##   element names: exprs
## protocolData
##   rowNames: GSM949164_Control1.CEL.gz GSM949166_Control2.CEL.gz
##     ... GSM949183_OSA10.CEL.gz (18 total)
##   varLabels: exprs dates
##   varMetadata: labelDescription channel
## phenoData
##   rowNames: GSM949164_Control1.CEL.gz GSM949166_Control2.CEL.gz
##     ... GSM949183_OSA10.CEL.gz (18 total)
##   varLabels: index
##   varMetadata: labelDescription channel
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation: pd.hugene.1.0.st.v1
```

This is in the form of an `GeneFeatureSet`; which is an `ExpressionSet`-like container. Knowing a bit of S4, we can see this through the class definition

```
getClass("GeneFeatureSet")
```

```
## Class "GeneFeatureSet" [package "oligoClasses"]
##
## Slots:
##
## Name:        manufacturer      intensityFile          assayData
## Class:          character          character          AssayData
##
## Name:           phenoData        featureData     experimentData
## Class: AnnotatedDataFrame AnnotatedDataFrame              MIAxE
##
## Name:          annotation       protocolData  .__classVersion__
## Class:          character AnnotatedDataFrame           Versions
##
## Extends:
## Class "FeatureSet", directly
## Class "NChannelSet", by class "FeatureSet", distance 2
## Class "eSet", by class "FeatureSet", distance 3
## Class "VersionedBiobase", by class "FeatureSet", distance 4
## Class "Versioned", by class "FeatureSet", distance 5
```

We see that this is a special case of a `FeatureSet` which is a special case of `NChannelSet` which is an `eSet`. We can see the intensity measures by

```
exprs(rawData)[1:4,1:3]
```

```
##   GSM949164_Control1.CEL.gz GSM949166_Control2.CEL.gz
## 1                      9411                      9917
## 2                       255                       200
## 3                      9171                      9202
## 4                       229                       220
##   GSM949168_Control3.CEL.gz
## 1                      8891
## 2                       181
## 3                      9266
## 4                       202
```

We see this is raw intensity data; the unit of measure is integer measurements on a 16 bit scanner, so we get values between 0 and $2^16 = 65, 536$. This is easily verifiable:

```
max(exprs(rawData))
```

```
## [1] 65534
```

Note the large number of features in this dataset, more than 1 million. Because of the manufacturing technology, Affymetrix can only make very short oligos (around 25bp) but can make them cheaply and at high quality. The short oligos means that the binding specificity of the oligo is not very good. To compensate for this, Affymetrix uses a design where a gene is being measured by many different probes simultaneously; this is called a probeset. As part of the preprocessing step for Affymetrix arrays, the measurements for all probes in a probeset needs to be combined into one expression measure.

Let us clean up the phenotype information for `rawData`.

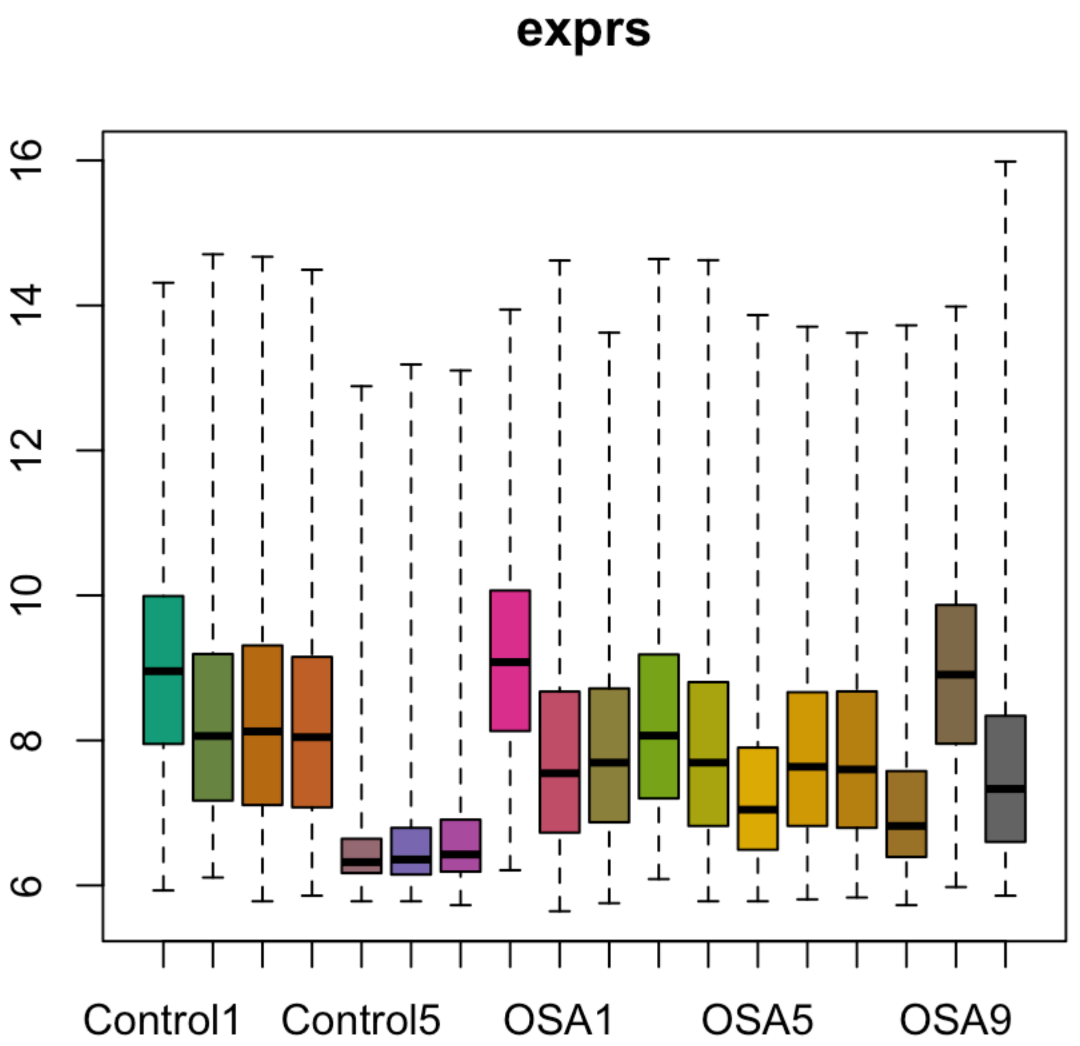
```
filename <- sampleNames(rawData)
pData(rawData)$filename <- filename
sampleNames <- sub(".*_", "", filename)
sampleNames <- sub(".CEL.gz$", "", sampleNames)
sampleNames(rawData) <- sampleNames
pData(rawData)$group <- ifelse(grepl("^OSA", sampleNames(rawData)),
                               "OSA", "Control")
pData(rawData)
```

```
##          index                  filename   group
## Control1     1 GSM949164_Control1.CEL.gz Control
## Control2     2 GSM949166_Control2.CEL.gz Control
## Control3     3 GSM949168_Control3.CEL.gz Control
## Control4     4 GSM949169_Control4.CEL.gz Control
## Control5     5 GSM949170_Control5.CEL.gz Control
## Control6     6 GSM949171_Control6.CEL.gz Control
## Control7     7 GSM949172_Control7.CEL.gz Control
## Control8     8 GSM949173_Control8.CEL.gz Control
## OSA1         9     GSM949174_OSA1.CEL.gz     OSA
## OSA2        10     GSM949175_OSA2.CEL.gz     OSA
## OSA3        11     GSM949176_OSA3.CEL.gz     OSA
## OSA4        12     GSM949177_OSA4.CEL.gz     OSA
## OSA5        13     GSM949178_OSA5.CEL.gz     OSA
## OSA6        14     GSM949179_OSA6.CEL.gz     OSA
## OSA7        15     GSM949180_OSA7.CEL.gz     OSA
## OSA8        16     GSM949181_OSA8.CEL.gz     OSA
## OSA9        17     GSM949182_OSA9.CEL.gz     OSA
## OSA10       18    GSM949183_OSA10.CEL.gz     OSA
```

Normalization

Let us look at the probe intensities across the samples, using the `boxplot()` function.

```
boxplot(rawData)
```



Boxplots are great for comparing many samples because it is easy to display many box plots side by side. We see there is a large difference in both location and spread between samples. There are three samples with very low intensities; almost all probes have intensities less than 7 on the log2 scale. From experience with Affymetrix microarrays, I know this is an extremely low intensity. Perhaps the array hybridization failed for these arrays. To determine this will require more investigation.

A classic and powerful method for preprocessing Affymetrix gene expression arrays is the RMA method. Experience tells us that RMA essentially always performs well so many people prefer this method; one can argue that it is better to use a method which always does well as opposed to a method which does extremely well on some datasets and poorly on others.

The RMA method was originally implemented in the *affy* package which has later been supplanted by the *oligo* package. The data we are analyzing comes from a “new” style Affymetrix array based on random priming; the *affy* package does not support these types of arrays. It is extremely easy to run RMA:

```
normData <- rma(rawData)
```

```
## Background correcting
## Normalizing
## Calculating Expression
```

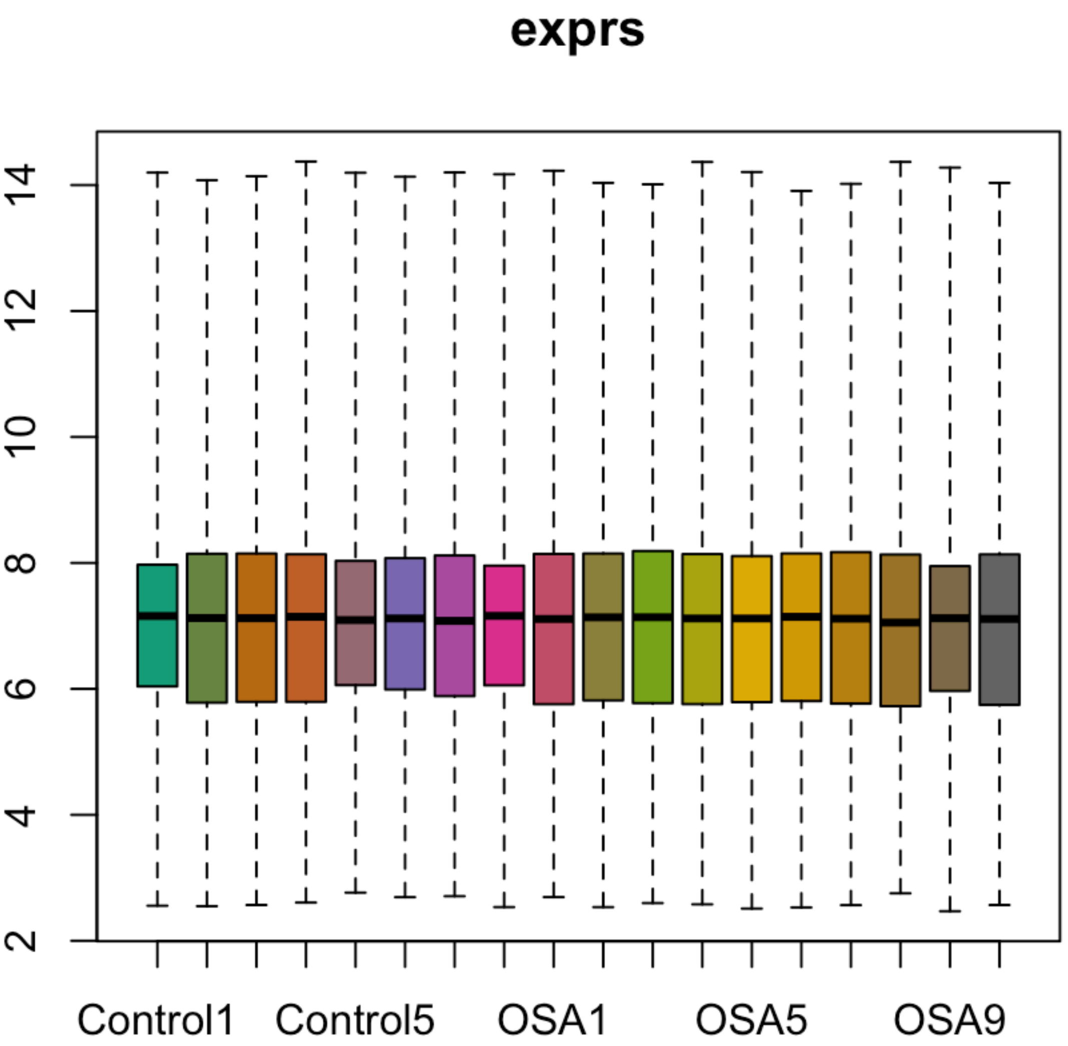
```
normData
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 33297 features, 18 samples
##   element names: exprs
## protocolData
##   rowNames: Control1 Control2 ... OSA10 (18 total)
##   varLabels: exprs dates
##   varMetadata: labelDescription channel
## phenoData
##   rowNames: Control1 Control2 ... OSA10 (18 total)
##   varLabels: index filename group
##   varMetadata: labelDescription channel
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation: pd.hugene.1.0.st.v1
```

Note how `normData` has on the order of 33k features which is closer to the number of genes in the human genome.

We can check the performance of RMA by looking at boxplots again.

```
boxplot(normData)
```



Here, it is important to remember that the first set of boxplots is at the probe level (~1M probes) whereas the second set of boxplots is at the probeset level (~33k probesets), so they display data at different summarization levels. However, what matters for analysis is that the probe distributions are normalized across samples and at a first glance it looks ok. One can see that the 3 suspicious samples from before still are slightly different, but that at least 2 more samples are similar to those.

For the normalization-interested person, note that while the distributions are similar, they are not identical despite the fact that RMA includes quantile normalization. This is because quantile normalization is done prior to probe summarization; if you quantile normalize different distributions they are guaranteed to have the same distribution afterwards.

The data is now ready for differential expression analysis.

SessionInfo

```
## R version 3.2.1 (2015-06-18)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  methods   stats     graphics  grDevices utils
## [8] datasets  base
##
## other attached packages:
##  [1] pd.hugene.1.0.st.v1_3.14.1 RSQLite_1.0.0
##  [3] DBI_0.3.1                  GEOquery_2.34.0
##  [5] oligo_1.32.0               Biostrings_2.36.4
##  [7] XVector_0.8.0              IRanges_2.2.7
##  [9] S4Vectors_0.6.5            Biobase_2.28.0
## [11] oligoClasses_1.30.0        BiocGenerics_0.14.0
## [13] BiocStyle_1.6.0            rmarkdown_0.8
##
## loaded via a namespace (and not attached):
##  [1] affxparser_1.40.0     knitr_1.11            magrittr_1.5
##  [4] splines_3.2.1         GenomicRanges_1.20.6  zlibbioc_1.14.0
##  [7] bit_1.1-12            foreach_1.4.2         stringr_1.0.0
## [10] GenomeInfoDb_1.4.2    tools_3.2.1           ff_2.2-13
## [13] htmltools_0.2.6       iterators_1.0.7       yaml_2.1.13
## [16] digest_0.6.8          preprocessCore_1.30.0 affyio_1.36.0
## [19] formatR_1.2           bitops_1.0-6          codetools_0.2-14
## [22] RCurl_1.95-4.7        evaluate_0.7.2        stringi_0.5-5
## [25] BiocInstaller_1.18.4  XML_3.98-1.3
```