The beadarrayFilter: An R Package to Filter Beads

by Anyiawung Chiara Forcheh, Geert Verbeke, Adetayo Kasim, Dan Lin, Ziv Shkedy, Willem Talloen, Hinrich W.H. Göhlmann, Lieven Clement

Abstract Microarrays enable the expression levels of thousands of genes to be measured simultaneously. However, only a small fraction of these genes are expected to be expressed under different experimental conditions. Nowadays, filtering has been introduced as a step in the microarray preprocessing pipeline. Gene filtering aims at reducing the dimensionality of data by filtering redundant features prior to the actual statistical analysis. Previous filtering methods focus on the Affymetrix platform and can not be easily ported to the Illumina platform. As such, we developed a filtering method for Illumina bead arrays. We developed an R package, **beadarrayFilter**, to implement the latter method. In this paper, the main functions in the package are highlighted and using many examples, we illustrate how **beadarrayFilter** can be used to filter bead arrays.

Introduction

Gene expression patterns are commonly assessed by microarrays that can measure thousands of genes simultaneously. However, in a typical microarray experiment, only a small fraction of the genes are informative which motivated the development of gene filtering methods. Gene filtering aims at reducing the dimensionality of data by filtering redundant features prior to the actual statistical analysis. This has been shown to improve differential expression analysis with Affymetrix microarrays (e.g., Talloen et al., 2007; Calza et al., 2007; Kasim et al., 2010).

Although different microarrays platforms share the same principle of hybridizing DNA to a complementary probe, they differ considerably by design. Unlike the Affymetrix microarrays which have sets of unique probes targeting a particular gene (resulting in a probe set for a targeted gene), Illumina microarrays have sets of identical probes. Thus, the existing filtering methods can not be readily ported to the Illumina platform. As a result, Forcheh et al. (2012) developed a filtering method for Illumina bead arrays. Forcheh et al. (2012) equally showed that filtering improves the analysis of differential expression. We provide the implementation of their method in the **beadarrayFilter** R software package. The **beadarrayFilter** package can take a normalized data frame or a normalized bead array ExpressionSetIllumina object (obtained using the `summarize` or `readBeadSummaryData` functions in the **beadarray** package of Dunning et al., 2007) or a normalized LumiBatch object as input and returns a list containing a filtered data frame or a filtered bead array ExpressionSetIllumina object or a filtered LumiBatch object, respectively. The package can also process summarized and normalized average intensities (eSet), their standard errors (seSet) and the number of beads used for summarization (nSet) as input and returns a list of components including the intra-cluster correlations (ICC), which can be used to assess different filtering strategies.

The paper contains a brief background of the filtering methodology followed by the introduction of the **beadarrayFilter** package with illustrative examples.

Filtering Illumina bead types

Let Y_{kij} be the bead-level expression intensity of bead j in sample (array) i in treatment group k, $i = 1, \ldots, n_k$, $j = 1, \ldots, m_{ki}$, $k = 1, \ldots, K$, then, Forcheh et al. (2012) proposed the following filtering model:

$$Y_{kij} = \mu + b_i + \epsilon_{kij}, \tag{1}$$

where μ represents the average expression for a bead type across all samples, b_i is the array specific random effect and ϵ_{kij} are the measurement errors, both normally distributed with mean zero and variance τ^2 and σ^2, respectively:

$$b_i \sim N(0, \tau^2), \epsilon_{kij} \sim N(0, \sigma^2).$$

The τ^2 captures the between array variability while σ^2 models the within array variability. Model (1) implies a common ICC, (Verbeke and Molenberghs, 2000), given by

$$\rho = \frac{\tau^2}{\tau^2 + \sigma^2}, \tag{2}$$

which is the correlation among the replicate probes on the array.

An informative bead type is expected to have a relatively high value of ρ since all corresponding beads have the same sequence. As such, the between array variability is the dominant variance component for informative bead types while the within array variability is the largest variance component for the non-informative bead types.

For Illumina bead arrays, within array variability is expected to vary across all arrays and treatment groups in the experiment. In this regard, Forcheh et al. (2012) included an array/treatment specific variance component to adjust the model for heteroscedasticity i.e., $\epsilon_{kij} \sim N(0, \sigma^2_{ki})$. Hence, the ICC becomes bead/array/group specific,

$$\rho_{ki} = \frac{\tau^2}{\tau^2 + \sigma^2_{ki}}, \tag{3}$$

and $\sum_{k=1}^{K} n_k$ ICCs are obtained for each bead type (more details can be found in Forcheh et al., 2012).

ICC based filtering of Affymetrix microarray data has been proposed in the literature (e.g, Talloen et al., 2007; Kasim et al., 2010). Typically, an ICC cut-off $\delta = 0.5$ has been used to declare a transcript informative. Forcheh et al. (2012) also relied on the ICC as a filtering criterion and proposed different thresholding schemes based on the five number summaries, i.e., a bead type can be called informative based on thresholding (1) the minimum ICC, (2) 25, (3) 50, (4) 75 quantiles or (5) the maximum ICC of all the arrays. Filtering is expected to become less stringent from strategy (1)–(5). These different thresholding strategies are implemented within the package.

beadarrayFilter package

The **beadarrayFilter** package can either take a normalized ExpressionSetIllumina object, a normalized LumiBatch object, a normalized data.frame or a normalized eSet, seSet and nSet as input and returns a list. We refer the user to the **beadarray** package and **lumi** package documentations for more details on generating ExpressionSetIllumina objects or LumiBatch object. For each bead type, the ICCs can be summarized using the 5 number summary or user specified quantiles. The corresponding ICC summaries are used for obtaining informative bead types. The package contains two major functions, which we refer to as: (1) a low level function `iccFun` and (2) a wrapper function `beadtypeFilter`. Model fitting is done using a modified version of the `MLM.beadarray` function of Kim and Lin (2011). Details on the functions can be obtained by using the help function in R (*?beadtypeFilter* or *?iccFun*).

Installation of beadarrayFilter

The **beadarrayFilter** package is available at CRAN, and can be installed using `install.packages("beadarrayFilter")`.

beadtypeFilter function

The `beadtypeFilter` function is a wrapper function for the `iccFun` function and is designed for users with a prime interest in obtaining filtered bead types. This function takes a normalized ExpressionSetIllumina object, a normalized LumiBatch object or a normalized data.frame and returns the names of the informative bead types. Optionally, the filtered ExpressionSetIllumina object or the filtered data.frame can also be returned. The filtered ExpressionSetIllumina object, or filtered LumiBatch object or the filtered data.frame can then be used for the downstream analysis.

iccFun function

`iccFun` is a low level function. It is designed for users that want to assess different filtering strategies. It takes a normalized eSet, seSet and nSet and the bead types identification variable (ProbeID), fits the filtering Model (1), calculates the ICC for each bead type on each array/treatment group, summaries the ICCs at the specified quantiles, and returns the ICC summaries, the within array variances, the between array variances as well as all ICCs. The ICCs output can later be used for filtering or to

assess different filtering strategies. Note the information printed as you execute the `beadtypeFilter` or `iccFun` functions:

- [1] "Number of converged transcripts: ... "
  This indicates the number of transcripts for which the filtering model has already converged while "Now ... remaining..." tells the number of transcripts still to be processed.
- [1] "Computing ICC for each array"
  This message is printed when the function begins to calculate the ICC for each array once the filtering model has been fitted to all the transcripts.
- [1] "Summarising the ICC at supplied quantile(s)"
  This message indicates the last stage of the filtering function where the ICCs are summarized at the supplied quantiles (see Forcheh et al., 2012).

The `emCDF` function within the **beadarrayFilter** package is used to plot the empirical cumulative density functions (edcfs) for the different threshold strategies discussed above. It processes the `iccFun` output and plots the empirical cumulative density functions (ecdf) for the different threshold strategies as discussed in Forcheh et al. (2012). It is expected that the filtering becomes more stringent as the ICC threshold increases and/or as the the thresholded ICC quantile decreases.

Informative bead types will have larger between array variances as compared to the within array variances. The `varianceplot` function takes the estimated between and within array variances from the `iccFun` function as inputs and plots them. The variances of noninformative bead types are plotted in blue while those of the informative bead types are displayed in red.

Upon filtering the `MLM.beadarray` function can be used for downstream analysis of single factor designs. For more details, see Kim and Lin (2011).

A summary of the functions within the **beadarrayFilter** package and their use is presented in Table 1.

Table 1: Main Functions within the **beadarrayFilter** package.

Function	Description
beadtypeFilter()	Fits the filtering model as in Forcheh et al. (2012), computes the ICC and filters the bead types.
iccFun()	Fits the filtering model as in Forcheh et al. (2012), and computes the ICC which can later be used for filtering or to assess different filtering strategies
MLM.beadarray()	Function to fit the filtering model or for the downstream analysis of single factor experimental designs
emCDF()	Plots the ecdf for the different threshold strategies
varianceplot()	Plots the the between-array and the within-array variances

Examples

The exampleSummaryData ExpressionSetIllumina object from the **beadarrayExampleData** package (Dunning et al., 2007), is used to illustrate filtering of ExpressionSetIllumina objects while the publicly available spike-in data (Dunning et al., 2008) are used to process data.frames. The $\log_2$ summarized expression intensities, obtained from Brain and Universal Human Reference RNA (UHRR) samples in the exampleSummaryData data, will be used. There are 12 arrays, 6 for the Brain samples and 6 for the UHRR samples. The spike-in dataset consists of 48 arrays and an array contained $\sim$ 48,000 bead types (non-spikes) and 33 bead types (spikes) targeting bacterial and viral genes absent in the mouse genome. We use the same subset of the spike-in data as in Kim and Lin (2011). This dataset includes 34,654 bead types targeting annotated genes with Genebank IDs and the 33 spikes. The data can be downloaded from `https://ephpublic.aecom.yu.edu/sites/rkim/Supplementary/files/KimLin2010.html` with the folder name "A normalized dataset for example analysis". We also show how the downstream analysis can be performed upon filtering using the spike-in data. Filtering of LumiBatch object is illustrated using the BeadStudio output used in the **beadarray** vignette and can be downloaded from `http://www.switchtoi.com/datasets.ilmn`.

beadtypeFilter function

This subsection shows how to use the `beadtypeFilter` function to filter normalized ExpressionSetIllumina objects, LumiBatch objects and data.frames. For an ExpressionSetIllumina, this is done using the exampleSummaryData ExpressionSetIllumina object from the **beadarrayExampleData** package.

```
# Normalize the  log2 transformed data
> library(beadarrayFilter)
> data(exampleSummaryData)
> exampleSummaryDataNorm <-
      normaliseIllumina(channel(exampleSummaryData, "G"),
                        method = "quantile", transform = "none")

# Filter the ExpressionSetIllumina
> iccResults <- beadtypeFilter(exampleSummaryDataNorm, Quantile = 1,
                               keepData = TRUE, delta = 0.5)
```

By specifying `iccQuant = 1` and `delta = 0.5`, the bead types are filtered using the maximum ICC at a cutoff of 50%. **The output of the** `beadtypeFilter` **function can then be observed as follows:**

```
> head(iccResults$InformProbeNames)
[1] "ILMN_1802380" "ILMN_1736104" "ILMN_1792389" "ILMN_1705423" "ILMN_1697642" "ILMN_1788184"

> exprs(iccResults$informData)[1:6, 1:6]
             4613710017_B 4613710052_B 4613710054_B 4616443079_B 4616443093_B 4616443115_B
ILMN_1802380     8.216547     8.229713     8.097047     8.343822     8.249190     8.347416
ILMN_1736104     5.317065     5.470957     5.054653     5.100678     5.446530     5.172984
ILMN_1792389     6.725049     7.003632     6.783809     7.214921     7.257032     7.286421
ILMN_1705423     5.496207     4.845898     5.394206     5.422772     5.479191     5.597819
ILMN_1697642     7.977234     7.912246     7.668253     7.850134     7.758535     7.447480
ILMN_1788184     5.291988     5.614500     5.565426     5.473346     5.573395     5.289350

> head(fData(iccResults$informData))
           ArrayAddressID   IlluminaID  Status
ILMN_1802380          10008 ILMN_1802380 regular
ILMN_1736104          10017 ILMN_1736104 regular
ILMN_1792389          10019 ILMN_1792389 regular
ILMN_1705423          10039 ILMN_1705423 regular
ILMN_1697642          10044 ILMN_1697642 regular
ILMN_1788184          10048 ILMN_1788184 regular

> dim(exampleSummaryDataNorm)
Features  Samples Channels
   49576       12        1

> dim(iccResults$informData)
Features  Samples Channels
   23419       12        1
```

23419 out of the 49576 bead types were declared informative using the maximum ICC at a cutoff point of 50%.

For a LumiBatch object, filtering is illustrated using the non-normalized data, an output of BeadStudio used in the **beadarray** vignette. The data file, AsuragenMAQC_BeadStudioOutput.zip, can be downloaded from http://www.switchtoi.com/datasets.ilmn. Once the file has been downloaded, unzip its content to your R working directory.

```
> require(lumi)

# Set the working directory to the directory where the unzipped data file was saved.
> setwd("C:/Multi_level_Illumina_feb2011/RPackageFinal/beadstudiooutputData")

# Read in the data using \code{lumiR} to obtain a LumiBatch object
> x.lumi <- lumiR("AsuragenMAQC-probe-raw.txt")
# Normalize the data without any further transformation step
> lumi.N <- lumiN(x.lumi, "rsn")
# Filter the LumiBatch
```

```
> iccResult <- beadtypeFilter(lumi.N, Quantile = 1, keepData = TRUE,
                              delta = 0.5)
```

By specifying `iccQuant = 1` and `delta = 0.5`, the bead types are filtered using the maximum ICC at a cutoff of 50%.

```
> dim(lumi.N)
Features  Samples
   48701        6

> dim(iccResult$informData)
Features  Samples
    1195        6
```

Only 1195 of the 48701 bead types were declared informative using the maximum ICC at a cutoff of 50%. This may be due to the way the data was summarized and normalized. How the processing of bead array data affects bead types filtering is a topic of future research.

For a data.frame, the `beadtypeFilter` function is illustrated using the Illumina spike-in data. Read the data from the file location where the data had been downloaded, unzipped and saved.

```
> filepath <- "C:/Multi_level_Illumina_feb2011/log2scale.normalized.txt"
> dt <- read.delim(filepath, header = TRUE, as.is = TRUE, row.names = NULL)[,-1]
> dt[1:6,1:5]
   ProbeID X1377192001_A.AVG_Signal X1377192001_A.Detection.Pval X1377192001_A.Avg_NBEADS
1   50014                6.150486                  0.579207900                      27
2   50017                6.616132                  0.074257430                      40
3   50019                8.164317                  0.000000000                      25
4   50020                7.414991                  0.001856436                      27
5   50022                5.804593                  0.974628700                      38
6   50025                6.412067                  0.173267300                      28
  X1377192001_A.BEAD_STDERR
1                0.09889349
2                0.05644992
3                0.06384269
4                0.07853792
5                0.08098911
6                0.08153830
```

Note, that the data.frame supplied to the `beadtypeFilter` function should contain the summarized intensities (`eSet`), standard errors (`seSet`) and the number of beads used for the summarization (`nSet`). When using a data frame, column names should be conform to BeadStudio output, i.e., the column names for eSet should end on "Signal", those for seSet on "STDERR" and the columns corresponding to nSet should end on "NBEADS". It is preferable to use an identification variable with a unique ID for each bead type. In the spike-in data, the spikes all have the same `TargetID`, thus the `ProbeID` is preferred. Similar to the ExpressionSetIllumina example, the `beadtypeFilter()` function is used for filtering.

```
> library(beadarrayFilter)
> iccResults <- beadtypeFilter(dt, Quantile = 0.5, keepData = TRUE, delta = 0.5)
```

By specifying `Quantile = 0.5`, bead types are filtered using the median ICC.

```
> head(iccResults$InformProbeNames)
[1]  50280  50440  70594 110138 110685 130402

> dim(dt)
[1] 34687   193
> dim(iccResults$informData)
[1] 238 193
```

238 of the 34687 bead types were declared informative based on thresholding the median ICC at a cutoff of 50%. A large number of bead types have been filtered out. It should be noted that this is probably due to the artificial nature of the spike-in data and we would expect lesser bead types to be filtered out in real life data.

iccFun function

The examples in this section show how the `iccFun` function can be used to process different data types, observe its output and assess the filtering strategies.

Processing data using the `iccFun` function

1. **Processing a data.frame**

```
> filepath <- "C:/Multi_level_Illumina_feb2011/log2scale.normalized.txt"
> dt <- read.delim(filepath, header = TRUE, as.is = TRUE, row.names = NULL)[,-1]
> eSet <- dt[, grep("Signal", names(dt))]
> seSet <- dt[, grep("STDERR", names(dt))]
> nSet <- dt[, grep("NBEADS", names(dt))]
> ProbeID <- dt[, 2]
> library(beadarrayFilter)
> iccResults <- iccFun(eSet, seSet, nSet, ProbeID = ProbeID,
                       iccQuant = c(0, 0.25, 0.5, 0.75, 0.8, 1),
                       diffIcc = TRUE, keepData = TRUE)
```

2. **Processing a LumiBatch object**

```
> setwd("C:/Multi_level_Illumina_feb2011/RPackageFinal/beadstudiooutputData")

# Read in the data using \code{lumiR} to obtain a LumiBatch object
> x.lumi <- lumiR("AsuragenMAQC-probe-raw.txt")
> lumi.N <- lumiN(x.lumi, "rsn")
> eSet <- exprs(lumi.N)
> seSet <- se.exprs(lumi.N)
> nSet <- beadNum(lumi.N)
> group <- c(1:dim(eSet)[2])
> ProbeID = fData(lumi.N)$ProbeID
> library(beadarrayFilter)
> iccResults <- iccFun(eSet, seSet, nSet, ProbeID = ProbeID,
                       iccQuant = c(0, 0.25, 0.5, 1),
                       diffIcc = TRUE, keepData = TRUE)
```

3. **Processing an ExpressionSetIllumina object**

```
> library(beadarrayFilter)
> data(exampleSummaryData)
> exampleSummaryDataNorm <-
      normaliseIllumina(channel(exampleSummaryData, "G"),
                        method = "quantile", transform = "none")
> aaa <-
      na.omit(data.frame(I(rownames(exprs(exampleSummaryDataNorm))),
                         exprs(exampleSummaryDataNorm)))
> ProbeID <- aaa[, 1]
> eSet <- na.omit(exprs(exampleSummaryDataNorm))
> stddev <- na.omit(se.exprs(exampleSummaryDataNorm))
> nSet <- na.omit(attributes(exampleSummaryDataNorm)$assayData$nObservations)
> seSet <- stddev/sqrt(nSet)
> iccResults <- iccFun(eSet, seSet, nSet, ProbeID = ProbeID, iccQuant = c(0, 0.25, 0.5, 1))
```

The output of the `iccFun` function

In this subsection, we illustrate how the output of the `iccFun` function can be observed. Note that we display the results for the exampleSummaryData, the output for the spike-in data can be found in Forcheh et al. (2012).

```
> head(iccResults$betweenvar)
      ProbeID    fit1.tau2
1 ILMN_1802380 1.3154886475
2 ILMN_1893287 0.0202718744
3 ILMN_1736104 0.7883136626
4 ILMN_1792389 0.5374776179
```

```
5 ILMN_1854015 0.0000000000
6 ILMN_1904757 0.0004272419

> iccResults$withinvar[1:6, 1:6]
             ProbeID      sigma2.4613710017_B sigma2.4613710052_B sigma2.4613710054_B
ILMN_1802380 ILMN_1802380          0.08024396           0.1133679          0.07562057
ILMN_1893287 ILMN_1893287          0.15510050           0.1495736          0.31645854
ILMN_1736104 ILMN_1736104          0.22109680           0.2449570          0.16237022
ILMN_1792389 ILMN_1792389          0.16305881           0.2232660          0.25316536
ILMN_1854015 ILMN_1854015          0.31302729           0.1367953          0.29684239
ILMN_1904757 ILMN_1904757          0.11065525           0.2427457          0.35319329
             sigma2.4616443079_B sigma2.4616443093_B
ILMN_1802380           0.1715118           0.1282700
ILMN_1893287           0.4629203           0.1956166
ILMN_1736104           0.2781976           0.2364219
ILMN_1792389           0.1187983           0.1560972
ILMN_1854015           0.2776505           0.4338320
ILMN_1904757           0.2621982           0.4215812

> head(iccResults$iccAll)
            [,1]        [,2]        [,3]        [,4]        [,5]        [,6]        [,7]
[1,] 0.942507641 0.920658321 0.945640089 0.884659197 0.911155532 0.945550561 0.813348960
[2,] 0.115593318 0.119354803 0.060202088 0.041954058 0.093899753 0.076838800 0.146496212
[3,] 0.780964425 0.762930437 0.829206926 0.739151757 0.769284994 0.672257600 0.862368982
[4,] 0.767237213 0.706516107 0.679798133 0.818981163 0.774938161 0.794058983 0.795226413
[5,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
[6,] 0.003846168 0.001756947 0.001208193 0.001626811 0.001012401 0.003354805 0.002726754
             [,8]        [,9]       [,10]       [,11]        [,12]
[1,] 0.2851197747 0.924933768 0.921820518 0.953747680 0.9672588084
[2,] 0.0086476707 0.079882094 0.046679979 0.083112294 0.0577318494
[3,] 0.4039671466 0.788168517 0.745235220 0.651296881 0.5468796285
[4,] 0.2446849630 0.818919242 0.889560842 0.859993713 0.7440351650
[5,] 0.0000000000 0.000000000 0.000000000 0.000000000 0.0000000000
[6,] 0.0005326228 0.001815729 0.005423024 0.002744552 0.0009235276

> head(iccResults$icc)
 ProbeID           q0       q0.25        q0.5          q1
1 ILMN_1802380 0.2851197747 0.904531448 0.923377143 0.967258808
2 ILMN_1893287 0.0086476707 0.054968882 0.078360447 0.146496212
3 ILMN_1736104 0.4039671466 0.667017420 0.754082829 0.862368982
4 ILMN_1792389 0.2446849630 0.734655400 0.784498572 0.889560842
5 ILMN_1854015 0.0000000000 0.000000000 0.000000000 0.000000000
6 ILMN_1904757 0.0005326228 0.001159245 0.001786338 0.005423024
```

If desired, the `iccResults$icc` output from the `iccFun` function can be used for filtering and assessing the different filtering strategies.

```
# Obtaining the number of informative bead types at each of the specified ICC quantiles
> apply(iccResults$icc[, -1], 2, function(x, thres) sum(x >= thres), thres = 0.5)
q0 q0.25  q0.5    q1
 4699 15784 17757 23419

# Obtaining the informative bead types using the minimum ICC
> filterDataNorm <- exampleSummaryDataNorm[subset(iccResults$icc,
                                                  iccResults$icc[, 2] >= 0.5)[, 1], ]
>  dim(filterDataNorm)
Features  Samples Channels
    4699       12        1
```

Assessing the filtering strategies

This is done using the `emCDF` function (Figure 1).

```
> emCDF(iccResults, iccQuant = c(0, 0.25, 0.5, 1))
```

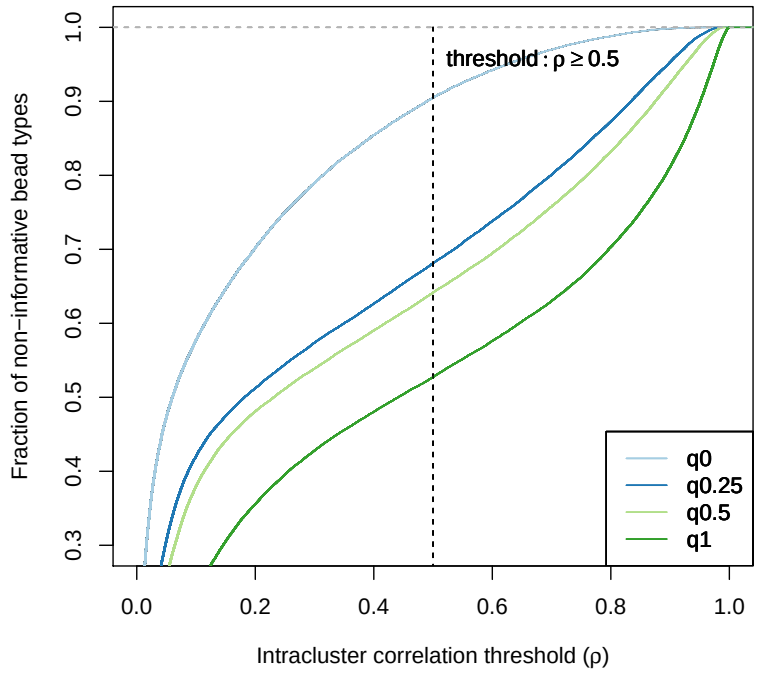

Figure 1: The empirical cumulative density function of the ICCs at the specified quantiles.

Further, the within and between array variances at the minimum ICC can be observed using the `varianceplot` function (Figure 2).

```
> varianceplot(iccResults, q = 1, delta = 0.8)
```

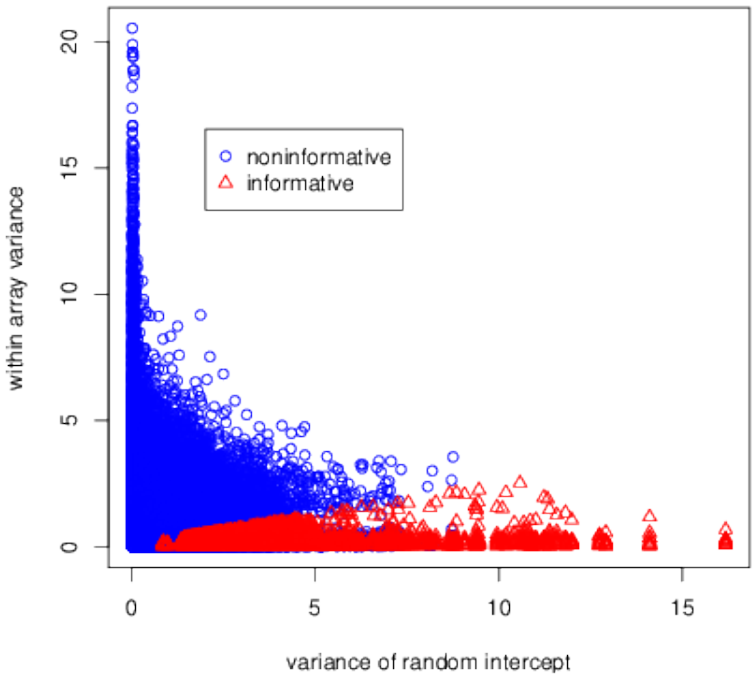

Figure 2: Variance of random intercept versus variance of measurement error. Bead types declared informative based on thresholding the minimum ICC at a cutoff point of 80%: noninformative bead types (○) and informative bead types (△).

By specifying `q = 1` and `delta = 0.8`, the informative beads (displayed in red) are obtained using the minimum ICC at a cutoff of 80%.

Analysis of differential expression

Once the data have been filtered, they can be used for downstream analysis. Here, we assess differential expression using a filtered data.frame. For the example 608 bead types were declared informative based on the maximum ICC. We refer to the help file of the `MLM.beadarray` function in the **beadarrayFilter** package for an example on a ExpressionSetIllumina object.

```
> iccResults <- beadtypeFilter(dt, Quantile = 1, keepData = TRUE,
                               delta = 0.5)
> dim(iccResults$informData)
> dat <- iccResults$informData
> eSet <- dat[, grep("Signal", names(dat))]
> seSet <- dat[, grep("STDERR", names(dat))]
> nSet <- dat[, grep("NBEADS", names(dat))]
```

We define the group variable to compare concentrations 0.3 and 0.1 pM in the spike-in data. This is done by selecting the column numbers of the arrays corresponding to the concentrations of interest.

```
> group1 <- c(26, 32, 38, 44)
> group2 <- c(27, 33, 39, 45)
> fit1 <-
      MLM.beadarray(eSet, seSet, nSet, list(group1, group2),
              var.equal = TRUE, max.iteration = 20, method = "ML")
```

The output of the `MLM.beadarray` **function can then be used to test** for equality of mean expression between the two concentrations

```
> df <- length(group1) + length(group2) - 2
> fit1$pvalue <- 2 * (1-pt(abs(fit1$t.statistics), df))
> fit1$pvalAdjust <- p.adjust(fit1$pvalue, method = "fdr",
                              n = length(fit1$pvalue))
> length(which(fit1$pvalAdjust < 0.05))
[1] 29
```

i.e., 29 bead types were found to be differentially expressed between concentrations 0.3 and 0.1 pM. Note that 22 of these 29 bead types are true positives (spikes).

Discussion

The **beadarrayFilter** package can be used to filter Illumina bead array data. The `beadtypeFilter` function can filter normalized ExpressionSetIllumina objects, normalized LumiBatch objects as well as normalized data.frames and returns the names of the informative bead types. Optionally, the user can also obtain the filtered data. This, however, does not return the required outputs to assess different filtering strategies nor the variances using the `emCDF` or the `varianceplot` functions, respectively. The `iccFun` function can be used to customize filtering strategies. It returns the required outputs for assessing different filtering strategies and the between and within array variances.

Acknowledgements

We acknowledge the support from IAP research network grant nr. P6/03 of the Belgian government (Belgian Science Policy), SymBioSys, the Katholieke Universiteit Leuven center of Excellence on Computational Systems Biology, (EF/05/007), and Bioframe of the institute for the Promotion of Innovation by Science and technology in Flanders (IWT: 060045/KUL-BIO-M$S-PLANT).

We are also grateful to Kim and Lin (2011) for making the `MLM.beadarray` function available.

Bibliography

S. Calza, W. Raffelsberger, A. Ploner, J. Sahel, T. Leveillard, and Y. Pawitan. Filtering genes to improve sensitivity in oligonucleotide microarray data analysis. *Nucleic Acids Research*, 35(16), 2007. [p1]

M. Dunning, N. B. Morais, A. Lynch, S. Tavaré, and M. Ritchie. Statistical issues in the analysis of Illumina data. *BMC Bioinformatics*, 9(1):85+, 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-85. URL http://dx.doi.org/10.1186/1471-2105-9-85. [p3]

M. J. Dunning, M. L. Smith, M. E. Ritchie, and S. Tavaré. beadarray: R classes and methods for Illumina bead-based data. *Bioinformatics*, 23(16):2183–2184, Aug. 2007. ISSN 1367-4811. doi: 10.1093/bioinformatics/btm311. URL http://dx.doi.org/10.1093/bioinformatics/btm311. [p1, 3]

A. C. Forcheh, G. Verbeke, A. Kasim, D. Lin, Z. Shkedy, W. Talloen, H. W. Göhlmann, and L. Clement. Gene filtering in the analysis of illumina microarray experiments. *Journal of Statistical Applications in Genetics and Molecular Biology*, 11(1), 2012. [p1, 2, 3, 6]

A. Kasim, D. Lin, S. Van Sanden, D.-A. Clevert, L. Bijnens, H. Göhlmann, D. Amaratunga, S. Hochreiter, Z. Shkedy, and W. Talloen. Informative or noninformative calls for gene expression: a latent variable approach. *Statistical applications in genetics and molecular biology*, 9(1), Jan. 2010. ISSN 1544-6115. doi: 10.2202/1544-6115.1460. URL http://dx.doi.org/10.2202/1544-6115.1460. [p1, 2]

R. S. Kim and J. Lin. Multi-level mixed effects models for bead arrays. *Bioinformatics*, 27:633–640, March 2011. ISSN 1367-4803. doi: http://dx.doi.org/10.1093/bioinformatics/btq708. URL http://dx.doi.org/10.1093/bioinformatics/btq708. [p2, 3, 9]

W. Talloen, D.-A. Clevert, S. Hochreiter, D. Amaratunga, L. Bijnens, S. Kass, and H. W. H. Göhlmann. I/NI-calls for the exclusion of non-informative genes: a highly effective filtering tool for microarray data. *Bioinformatics*, 23(21):btm478–2902, Oct. 2007. doi: 10.1093/bioinformatics/btm478. URL http://dx.doi.org/10.1093/bioinformatics/btm478. [p1, 2]

G. Verbeke and G. Molenberghs. *Linear mixed models for longitudinal data*. Springer Series in Statistics. Springer-Verlag, New-York, 2000. [p2]

Anyiawung Chiara Forcheh
Interuniversity Institute for Biostatistics and statistical Bioinformatics, Katholieke Universiteit Leuven, Kapucijnenvoer 35, Blok D, bus 7001, B3000 Leuven, Belgium,
and Universiteit Hasselt,
Belgium
forcheh@yahoo.com

Geert Verbeke and Lieven Clement
Interuniversity Institute for Biostatistics and statistical Bioinformatics Katholieke Universiteit Leuven, Kapucijnenvoer 35, Blok D, bus 7001, B3000 Leuven,
Belgium and Universiteit Hasselt, Belgium
Geert.Verbeke@med.kuleuven.be
lieven.clement@med.kuleuven.be

Dan Lin and Ziv Shkedy
Interuniversity Institute for Biostatistics and statistical Bioinformatics Universiteit Hasselt, Agoralaan 1, B3590 Diepenbeek, Belgium, and Katholieke Universiteit Leuven,
Belgium
dan.lin2@pfizer.com
ziv.shkedy@uhasselt.be

Adetayo Kasim
Wolfson Research Institute, Durham University,
Queen's Campus, University Boulevard, Thornaby,
Stockton-on-Tees,TS17 6BH, United Kingdom
a.s.kasim@durham.ac.uk

Willem Talloen and Hinrich W.H. Göhlmann
Janssen Pharmaceutica N.V.,

Beerse, Belgium
wtalloen@its.jnj.com
hinrich@goehlmann.info